KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Automatische verificatie en kwaliteitscontrole van UML-diagrammen met FO(.)

Thomas Vochten

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdoptie Gedistribueerde systemen

Promotor:
Prof. dr. Marc Denecker

Assessor:
TBD

Begeleider:
Matthias van der Hallen

Academiejaar 2017 – 2018

Voorwoord

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Thomas Vochten

Inhoudsopgave

Samenvatting

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Hoofdstuk 1

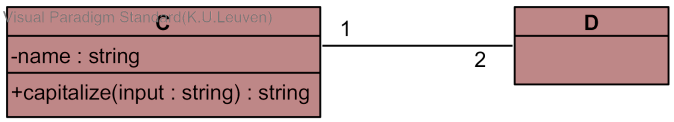
Inleiding

Binnen software engineering is UML een veelgebruikt gereedschap om het domein waarin de software die ontworpen wordt alsook de structuur van de software zelf grafisch weer te geven. Het is voor de ontwerper interessant om uit een tekstuele beschrijving van wat de voorgestelde software moet kunnen de relevante concepten en procedures te halen, die neer te zetten in een diagram en door middel van de verscheidene symbolen aangeboden door UML uit te drukken hoe die concepten en procedures met elkaar interageren. Op deze manier kan een team snel duidelijkheid scheppen in welke doelen ze precies moeten bereiken.

Het is echter makkelijk om het overzicht te verliezen als de gebruikte diagrammen omvangrijk worden. Dit kan een probleem zijn omdat fouten die worden gemaakt in de ontwerpfase en pas laat in het productieproces ontdekt worden kostbaar zijn om recht te zetten. Het komt ook voor dat een ontwerper per vergissing overbodige informatie toevoegt aan een diagram en dat daardoor het diagram minder duidelijk wordt.

In deze masterproef worden in het bijzonder UML-klassediagrammen beschouwd. Een klassediagram beschrijft welke concepten (in deze tekst verder *klasses* genoemd) er bestaan binnen de software. Elk van die klasses kan attributen en operaties hebben. Verder geeft een klassediagram ook weer welke klasses in relatie staan tot elkaar. Deze relaties leggen vast aan welke beperkingen alle mogelijke toestanden van de beschreven software moeten voldoen om beschouwd te worden als correct.

Beschouw volgend klassediagram:



Figuur 1.1: Een voorbeeld van een klassediagram

Dit klassediagram drukt uit dat er twee klasses bestaan: C en D. C heeft één attribuut, *name*, dat van type *string* is. Het heeft ook één operatie *capitalize* dat *input*, van type *string*, als parameter heeft. *capitalize* geeft een resultaat terug dat

ook van type *string* is. Voorts drukt de lijn tussen C en D uit dat er een relatie bestaat tussen de twee klasses. Beschouw klasse C. Als we vanuit die klasse de lijn volgen, zien we dat er aan het ander uiteinde staat dat elke C-object in relatie moet staan tot exact twee D-objecten. Zo ook zien we dat, als we vertrekken vanuit D, elk D-object in relatie moet staan tot exact één C-object.

Met het voorgaande in het achterhoofd beschouwen we in deze masterproef twee categorieën van gebreken in een klassediagram:

- **Inconsistenties:** Het klassediagram is zo opgebouwd dat geen enkele mogelijke toestand van de software kan beantwoorden aan de voorwaarden die worden opgelegd. Dit betekent dat het stuk van de software dat wordt beschreven in het diagram onmogelijk kan werken.
- **Kwaliteitsgebreken:** Deze gebreken hebben een negatieve impact op de kwaliteit van het softwareontwerp. Zo kunnen ze bijvoorbeeld onduidelijkheden in het ontwerp introduceren of het onderhoud van de software éénmaal ingezet in productie bemoeilijken.

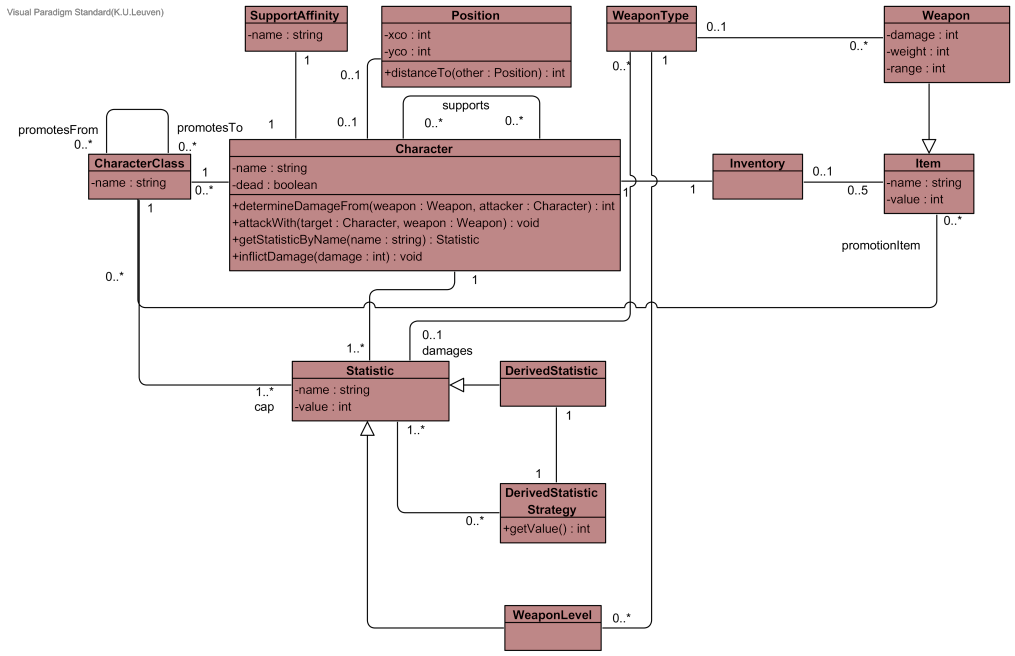
Deze masterproef heeft tot doel om automatisch op een gestructureerde manier uit te drukken welke informatie een UML-klassediagram juist bevat. Die informatie willen we op zijn beurt terug gebruiken om inconsistenties en kwaliteitsgebreken te detecteren. Concreter willen we predikatenlogica gebruiken om de informatie neer te schrijven en om aan detectie van gebreken te doen. De volgende hoofdstukken beschrijven hoe we dit exact willen bereiken.

relevante structuur van document beschrijven

Hoofdstuk 2

Controleren van consistentie

In dit hoofdstuk treden we meer in detail over hoe we de consistentie van een diagram willen controleren. Daarvoor willen we een specifieke vorm van logische theorie automatisch laten genereren. In deze theorieën staan **objecten** centraal. Deze objecten zijn instanties van een klasse die voorkomt in het beschouwde diagram, hebben exact de attributen en operaties van die klasse en maken deel uit van exact die relaties die het diagram voorschrijft voor die klasse. Aan de hand van volgend voorbeeld zullen we illustreren welke regels we gebruiken om zulk een theorie op te bouwen:



Figuur 2.1: Leidend voorbeeld van een klassediagram

Meer bepaald willen we uitdrukken welke **klasses** er bestaan in het diagram waarvan een object een instantie kan zijn, welke **attributen** en **operaties** elke

klasse bevat, welke **associaties** er bestaan tussen de verscheidene klasses en welke **klassehiërarchieën** er bestaan.

2.0.1 Logisch type *Object*

Zoals gezegd staan in deze theoriën objecten centraal, dus is het vanzelfsprekend om een logisch type *Object* te voorzien. Dit logisch type bevat dus softwareobjecten.

2.0.2 Logisch type *ClassObject* en predicaat *StaticClass*\2

Dit logisch type bevat exact de klasses die worden weergegeven in het diagram — niet meer en niet minder. *Character*, *Inventory*, *Item* enz. zijn dus *ClassObject*s. We drukken uit dat een *Object* een instantie is van een bepaald *ClassObject* door middel van het predicaat *StaticClass*\2. *StaticClass(o1,Character)* zegt dus uit dat het object *o1* een instantie is van klasse *Character*.

2.0.3 Voorstellen van attributen

Voor elk attribuut voegen we een binair predicaat toe waarvan de naam beantwoordt aan het patroon: *Klassenaamattribuutnaam*. Voor klasse *Character* en attribuut *name* resulteert dit dus in het predicaat *Charactername*\2. Het eerste argument van dit predicaat is een *Object*. Het type van het tweede argument hangt af van wat er in het diagram staat: Als het een primitief type is zoals *string* of *int*, zal dat ook het type zijn van het tweede predicaat; in het andere geval is het type van het tweede argument ook *Object*. De signatuur van *Charactername*\2 is daarom *Charactername(Object, string)*. Voor elk attribuut worden ook een aantal andere regels afgeleid:

- Als het tweede argument van het attribuutpredicaat *Object* is, wordt er een regel toegevoegd van de vorm:

$$\forall o1[Object]\forall o2[Object](Klassenaamattribuutnaam(o1, o2) \Rightarrow \\ StaticClass(classObj, o1) \wedge StaticClass(attrClassObj, o2))$$

  waarbij *classObj* het logisch object van type *ClassObject* dat het attribuut bevat en *attrClassObj* het logisch object van type *ClassObject* dat dient als mogelijke waarde van dit attribuut. Deze regel verzekert dat de attribuuthouder en de attribuutwaarde van de juiste klasse zijn. In dit diagram komt dit geval nergens voor en wordt deze regel dus niet toegepast.

- Als het tweede argument van het attribuutpredicaat van een primitief type is, wordt een regel toegevoegd van de vorm:

$$\forall o[Object]\forall x[primitiveType](Klassenaamattribuutnaam(o, x) \Rightarrow \\ StaticClass(classObj, o))$$

waarbij primitiveType het type van de attribuutwaarde. De signatuur van het predicaat verzekert dat de attribuutwaarde van het juiste type is, dus moet dit niet explicit worden neergeschreven. Deze regel zorgt ervoor dat de volgende zin wordt toegevoegd aan de theorie:

$$\forall o[Object]\forall x[primitiveType](Charactername(o,x) \Rightarrow \\ StaticClass(Character,o))$$

- De multipliciteit van het attribuut wordt ook in rekening gebracht. Zij *lowerBound* de ondergrens en *upperBound* de bovengrens. Dan is de meest algemene vorm van deze regel als volgt:

$$\forall o1[Object](StaticClass(classObj,o1) \Rightarrow lowerBound \geq \\ \#\{o2 : Klasseattribuutnaam(o1,o2)\} \geq upperBound$$

waarbij *lowerBound* wordt weggelaten als deze 0 is en *upperBound* wordt weggelaten als deze $*$ is. Indien beide van deze voorwaarden gelden, wordt er geen regel afgeleid betreffende de multipliciteit van het attribuut. Als $lowerBound = upperBound$, wordt deze regel in de plaats:

$$\forall o1[Object](StaticClass(classObj,o1) \Rightarrow \\ \exists_{=upperBound} o2(Klassenaamattribuutnaam(o1,o2))$$

Voor *Charactername*\2 wordt daarom afgeleid:

$$\forall o[Object](StaticClass(Character,o) \Rightarrow \exists_{=1} x(Charactername(o,x))$$

2.0.4 Voorstellen van operaties

Voor elke operatie voegen we een predicaat toe dat beantwoordt aan volgend patroon: *Klassenaamoperatienaam*\$(m+2)$, waarbij m het aantal argumenten dat als invoer wordt meegegeven aan de operatie. De signatuur ziet eruit als $Klasseoperatienaam(o,p_1,\ldots,p_m,r)$, waarbij o het object van logisch type *Object* waarop de operatie wordt opgeroepen, $p_1 \ldots p_m$ de argumenten en r het resultaat van de oproep van de operatie op het object o met de gegeven argumenten. Indien er geen argumenten zijn, ziet de signatuur eruit als *Klassenaamoperatienaam(o,r)*. Voor *determineDamageWeaponFrom(Weapon)* van *Character* wordt dit dus *CharacterdetermineDamageFrom(Object,Object,int)*.

Voor elke operatie worden de volgende regels afgeleid:

- Het object waarop de operatie wordt opgeroepen (zijnde o), de parameters (zijnde $p_1 \dots p_m$) en het resultaat van de oproep (zijnde r) moeten allemaal van de juiste klasse zijn. Daarom wordt een regel toegevoegd van de vorm:

$$\begin{aligned}&\forall o[Object]\forall p_1[Object]\dots\forall p_m[Object]\forall r[Object]\\&(Klassenaamoperatienaam(o,p_1,\dots,p_m,r)\Rightarrow\\&StaticClass(classObj,o)\wedge StaticClass(p_1ClassObj,p_1)\wedge\dots\wedge\\&StaticClass(p_mClassObj,p_m)\wedge StaticClass(resultClassObj,r))\end{aligned}$$

  Voor elke p waarvoor geldt dat het van een primitief type is wordt de corresponderende *StaticClass(p_l, p_lClassObj)* (met $1 \leq l \leq m$) weggelaten; hetzelfde geldt voor r. De invulling voor *CharacterdetermineDamageFrom(o,p1,r)* wordt dus:

$$\begin{aligned}\forall o[Object]\forall p_1[Object](&CharacterdetermineDamageFrom(o,p_1,r)\Rightarrow\\&StaticClass(Character,o)\wedge StaticClass(Weapon,p_1))\end{aligned}$$

- Voor elke combinatie van Object waar de operatie wordt opgeroepen en invoerparameters moet gelden dat er exact één resultaat is:

$$\begin{aligned}&\forall o[Object]\forall p_1[Object]\dots\forall p_m[Object]\\&(StaticClass(classObj,o)\wedge StaticClass(p_1ClassObj,p_1)\wedge\dots\wedge\\&StaticClass(p_m,p_mClassObj)\Rightarrow\\&\exists!r[Object](Klassenaamoperatienaam(o,p_1,\dots,p_m,r)))\end{aligned}$$

  Opnieuw geldt dat voor primitieve types de bijhorende conjuncten weggelaten worden. De invulling voor *CharacterdetermineDamageFrom(Object,Object,int)* wordt:

$$\begin{aligned}&\forall o[Object]\forall p_1[Object](StaticClass(Character,o)\wedge\\&StaticClass(Weapon,p_1)\Rightarrow\exists!r(CharacterdetermineDamageFrom(o,p1,r))\end{aligned}$$

2.0.5 Voorstellen van associaties

Voor elke associatie voegen we een predicaat toe dat beantwoordt aan volgend patroon: $ClassOneand\dots andClassM\backslash m$, waarbij m de ariteit van de associatie. Voor de associatie tussen *Inventory* en *Item* wordt dit dus *InventoryandItem(Object,Object)*. We leiden regels van de volgende vormen af voor elke associatie:

- De deelnemende Objects moeten allemaal van de juiste klasse zijn. Daarom wordt een regel toegevoegd van de vorm:

$$\forall o_1[Object] \ldots \forall o_m[Object](ClassOne \ldots andClassM(o_1, \ldots, o_m)$$
$$\Rightarrow StaticClass(o_1 ClassObj, o_1) \wedge \ldots \wedge StaticClass(o_m ClassObj, o_m))$$

  Voor *InventoryandItem(Object,Object)* wordt dit:

$$\forall o_1[Object] \forall o_2[Object](InventoryandItem(o1, o2) \Rightarrow$$
$$StaticClass(Inventory, o1) \wedge StaticClass(Item, o2))$$

- De multipliciteit voor elke rol moet worden uitgedrukt. Voor alle o_l waarvoor $1 \leq l \leq m$ wordt een regel toegevoegd van de volgende vorm:
  Zij $lowerBound_l$ de ondergrens en $upperBound_l$ de bovengrens:

$$\forall c_1[Object] \ldots \forall c_m[Object](StaticClass(c_1 ClassObj, c_1) \wedge \ldots \wedge$$
$$StaticClass(c_m ClassObj, c_m) \Rightarrow lowerBound_l \leq$$
$$\#o_l : ClassOneand \ldots ClassM(c_1, \ldots, o_1, \ldots, c_m) \leq upperBound_l)$$

  waarbij de c met index l overgeslagen wordt. Indien de ondergrens gelijk is aan 0 of de bovengrens gelijk is aan $*$ worden dezen weggelaten. Als beide voorwaarden gelden, wordt voor deze l geen regel afgeleid. Indien $lowerBound_l = upperBound_l$ wordt in de plaats afgeleid:

$$\forall c_1[Object] \ldots \forall c_m[Object](StaticClass(c_1 ClassObj, c_1) \wedge \ldots \wedge$$
$$StaticClass(c_m ClassObj, c_m) \Rightarrow$$
$$\exists_{=upperbound_l} o_l(ClassOneand \ldots andClassM(c_1, \ldots, o_l, \ldots, c_m)))$$

  Voor $InventoryandItem \backslash m$ worden de volgende regels afgeleid:

$$\forall o_2[Object](StaticClass(Item, o_2) \Rightarrow$$
$$\#o_1 : InventoryandItem(o_1, o_2) \leq 1)$$

$$\forall o_1[Object](StaticClass(Inventory, o_1) \Rightarrow$$
$$\#o_2 : InventoryandItem(o_1, o_2) \leq 5)$$

2.0.6 Voorstellen van klassehiërarchiëen

Stel dat voor een object o van logisch type *Object* gegeven is dat *StaticClass(oClassObject,o)*. Ons doel is dat *StaticClass(superClassObject,o)* geldt voor alle objecten van logisch type *ClassObject* die volgens het diagram superklasses zijn van *oClassObject* — niet meer en niet minder. Daartoe introduceren we het predikaat *IsDirectSupertypeOf(ClassObject,ClassObject)* dat ingevuld wordt door alle directe subklasseringen van het diagram en het predikaat *IsSupertypeOf(ClassObject,ClassObject)*, hetgeen de transitieve sluiting is van *IsDirectSupertypeOf*. Om zowel de invulling van *IsDirectSupertypeOf*\2 te doen als de transitieve sluiting te berekenen maken we gebruik van **inductieve definities** voor twee redenen:

1. In predikatenlogica is het onmogelijk om op een universeel geldige manier de transitieve sluiting uit te drukken.

2. Als men in een inductieve definitie een lijst feiten opsomt, drukt men tegelijk ook uit dat exact die feiten waar zijn — niet meer of niet minder.

In één definitie lijsten we dus de feiten die we kunnen aflezen van het diagram op:

$$\begin{aligned}\{IsDirectSupertypeOf(Statistic, Weaponlevel) \leftarrow\\ IsDirectSupertypeOf(Statistic, DerivedStatistic) \leftarrow\\ IsDirectSupertypeOf(Item, Weapon) \leftarrow\\ \}\end{aligned}$$

In een andere definitie drukken we de transitieve sluiting uit en gebruiken we die ook meteen om het gewenste resultaat voor *StaticClass*\2 uit te komen:

$$\begin{aligned}&\{\forall x[ClassObject]\forall y[ClassObject](IsSupertypeOf(x,y) \leftarrow\\ &\ IsDirectSupertypeOf(x,y))\\ &\forall x[ClassObject]\forall y[ClassObject](IsSupertypeOf(y,x) \leftarrow\\ &\exists z(IsSupertypeOf(y,z) \wedge IsSupertypeOf(z,x)))\\ \\ &\forall x[ClassObject]\forall o[Object](StaticClass(x,o) \leftarrow RuntimeClass(x,o))\\ &\forall x[ClassObject]\forall y[ClassObject]\forall o[Object](StaticClass(y,o) \leftarrow\\ &RuntimeClass(x,o) \wedge IsSupertypeOf(y,x))\}\end{aligned}$$

waarbij *RuntimeClass(ClassObject,Object)* een predikaat is dat uitdrukt wat de uniek dynamisch bepaalde klasse is van een *Object* (de veronderstelling is dat in een geldige toestand van een programma in uitvoering ieder object exact één *runtime* klasse heeft).
In hoofdstuk 4 wordt de logische theorie die automatisch gegenereerd werd volgens

de regels opgelijst in dit hoofdstuk weergegeven en wordt ook uitgelegd hoe die theorie wordt gebruikt om de consistentie van het diagram te controleren.

Hoofdstuk 3

Controleren op kwaliteitsgebreken

Waar in hoofdstuk 2 *Object*s centraal stonden, doen we daar hier afstand van: we abstraheren *Object*s weg en concentreren ons in de plaats op *ClassObject*s. We gebruiken het diagram in figuur ?? weer als begeleidend voorbeeld. In de volgende subsecties overlopen we hoe we de theorie die we gebruiken voor dit probleem opbouwen.

3.0.1 Gebruikte logische types en predikaten

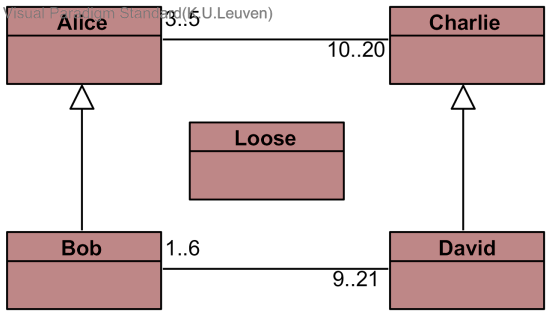
We bewaren het logisch type *ClassObject* en het predikaat *IsSupertypeOf(ClassObject, ClassObject)* exact zoals ze zijn in hoofdstuk 2 (en berekenen de transitieve sluiting horende bij *IsSupertypeOf*\2 op dezelfde manier) en gebruiken daar bijkomend volgende predikaten:

- ***BiAssoc(ClassObject,ClassObject)***: drukt uit dat er een binaire associatie bestaat tussen de twee klasses.
- ***BiAssocLow(ClassObject,ClassObject,ClassObject,nat)***: voor *BiAssocLow(x,y,x,n1)* geldt dat voor de binaire associatie tussen klasse x en klasse y de ondergrens voor de multipliciteit aan de x-kant gelijk is aan *n1*; een gelijkaardige interpretatie geldt voor *BiAssocLow(x,y,y,n2)*.
- ***BiAssocHigh(ClassObject,ClassObject,ClassObject,nat)***: gelijkaardig aan *BiAssocLow*\4, maar dan voor de bovengrens van de multipliciteit.

Deze predikaten worden ingevuld met een lijst van feiten die af te lezen zijn van het diagram. In hoofdstuk 4 wordt de logische theorie die het resultaat is van dit proces weergegeven.

3.0.2 Kwaliteitsgebreken detecteren

Er zijn drie kwaliteitsgebreken waarnaar wordt gezocht in de resulterende theorie:



Figuur 3.1: Voorbeeld van meer permissieve multipliciteiten in een klassehiërarchie en van een losstaande klasse (zijnde *Loose*)

- **Many-to-many associaties**: Dit zijn associaties waar dat de bovengrens van de multipliciteiten aan beide kanten gelijk is aan $*$. Het voorkomen van een many-to-many associatie is doorgaans een teken dat er een klasse ontbreekt in het ontwerp. Het is dus van groot belang dat dit wordt opgespoord en opgelost.
- **Losstaande klasse**: Concreet is een losstaande klasse een klasse die geen associatie heeft met een andere klasse in het ontwerp. Zulk een klasse is nutteloos en moet ofwel verbonden worden met een andere klasse of verwijderd worden.
- **Overbodige associaties in een klassehiërarchie**: Beschouw figuur ??. Daar is te zien dat de grenzen van de multipliciteiten in de associatie *Alice—Charlie* strengere voorwaarden opleggen dan die van de associatie *Bob—David*, en dat daarom de associatie *Bob—David* overbodig is. Om de verstaanbaarheid van het diagram te verbeteren, wordt die associatie best verwijderd.

We definiëren deze respectievelijke gebreken in de logische theorie door middel van de volgende logische zinnen:

$$\forall x[ClassObject]\forall y[ClassObject](ManyToMany(x,y) \Leftrightarrow BiAssoc(x,y) \wedge \neg\exists z[nat](BiAssocHigh(x,y,x,z)) \wedge \neg\exists z[nat](BiAssocHigh(x,y,y,z)))$$

$$\forall x[ClassObject](LooseClass(x) \Leftrightarrow \neg(\exists y[ClassObject](BiAssoc(x,y)) \vee \exists s[ClassObject]\exists y[ClassObject](IsSupertypeOf(s,y) \wedge BiAssoc(s,x))))$$

oplossing vinden voor de derde: gewoon verwijzen naar IDP-bestand?

Deze regels worden meteen toegevoegd aan de theorie die wordt gegenereerd zoals uitgelijnd eerder in dit hoofdstuk. In hoofdstuk 4 wordt uitgelegd hoe deze theorie wordt gebruik om kwaliteitsgebreken te vinden.

Hoofdstuk 4

De rol van IDP

4.0.1 Korte inleiding in IDP

IDP is een kennisbanksysteem. Deze kennisbanken zijn opgesteld in FO(·), een uitbreiding van predikatenlogica. De basisblokken van een specificatie in IDP zijn als volgt:

- **Vocabularium**: Hier specificeert de ontwerper de logische types die bestaan in het beschouwd domein, predikaten en functies
- **Theorie**: Hier schrijft de ontwerper zinnen in FO(·) die bepalen welke structuren over het beschouwde vocabularium modellen zijn (waarbij natuurlijk niet wordt uitgesloten dat de ontwerper een inconsistente theorie ontwerpt).
- **Structuur**: De ontwerper vult hier de logische types gedefinieerd in het vocabularium in (waar nodig) en geeft voor één of meerdere predikaten aan welke tupels wel of geen lid zijn, als hij dat wil.

De ontwerper kan meerdere vocabularia, theorieën en structuren neerschrijven. Elke theorie kan wel maar de symbolen van één vocabularium gebruiken en men kan in een structuur alleen spreken over één theorie.

IDP gebruikt zijn eigen symbolen voor universele kwantoren, existentiële kwantoren en logische connectieven. Voor meer informatie over IDP, zie

verwijzing naar relevant document

4.0.2 Gebruik van modeluitbreiding

Gegeven een structuur over een bepaalde theorie en vocabularium kan de gebruiker de opdracht geven aan IDP om een uitbreiding te vinden van deze structuur die ervoor zorgt de structuur een model is van de theorie. Dit is een vorm van inferentie die men **modeluitbreiding** noemt. Het kan echter het geval zijn dat IDP antwoordt dat zulk een uitbreiding niet bestaat of dat de uitvoering nooit eindigt.

Controleren van consistentie

In bijlage A staat de logische theorie die werd gegenereerd volgens de regels uitgelijnd in hoofdstuk 2. Als men dit geeft als invoer aan IDP, is het besluit dat er een model bestaat voor de theorie en dat het diagram inderdaad consistent is.

4.0.3 Detecteren van kwaliteitsgebreken

In bijlage B staat de logische theorie die werd gegenereerd uit een combinatie van de diagrammen uit figuren **??** en **??** volgens de regels uitgelijnd in hoofdstuk 3. IDP vindt alle many-to-many associaties, besluit dat *Loose* een losstaande klasse is en dat de associatie *B*—overbodig is door de samenloop van de klassehiërarchie en de opgelegde multipliciteiten.

Hoofdstuk 5

Simuleren van gedrag op basis van een sequentiediagram

Een ander populair type van UML-diagram is het sequentiediagram. Waar klassediagrammen de informatie bevat in klasses en de verbanden tussen klasses benoemen, beschrijven sequentiediagrammen het gedrag van instanties van de klasses. Deze instanties communiceren via berichten. Doorgaans zijn deze berichten ofwel een oproep van een methode gedefinieerd voor de klasse van een instantie ofwel een instantiatie van een nieuwe instantie. De berichten zijn genummerd volgens een bepaalde volgorde en samen modelleren ze het gedrag van een stuk van de software.

Figuur ?? geeft een voorbeeld van een sequentiediagram gebaseerd op het klassediagram voorgesteld in figuur ??.

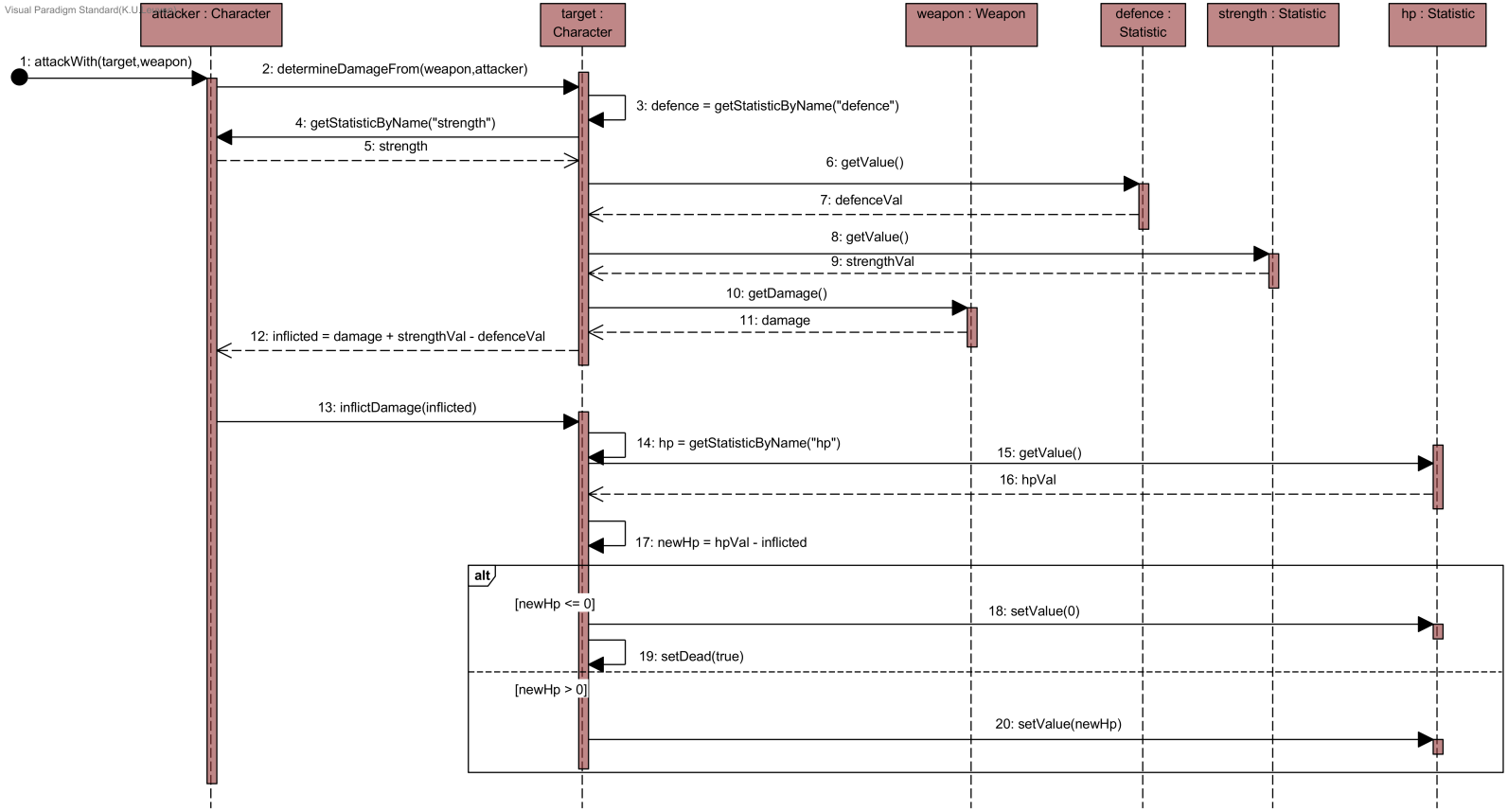
Een instantie wordt voorgesteld door een kader met daarin tekst volgens het patroon *instantienaam : klassenaam*. Dit wil zeggen dat bijvoorbeeld *attacker* een instantie is van de klasse *Character*. Vanuit elk kader vertrekt ook een streepjeslijn: de **levenslijn**. Deze levenslijn kan ingevuld worden door gekleurde balken, welke de duur van een oproep van een methode aan een instantie voorstellen. Verder zijn er ook kaders die berichten omsluiten. Deze kaders duiden **gecombineerde fragmenten** aan, en in deze tekst beschouwen we twee soorten:

1. Het **altfragment**: Deze soort duidt een *if-else*-constructie aan. Het bestaat uit twee delen, namelijk het if-deel en het *else*-deel, en er staat aangeduid onder welke voorwaarden welk deel wordt uitgevoerd. Figuur ?? bevat een voorbeeld van een altfragment.

2. Het **lusfragment**: Deze soort duidt een lusconstructie aan. Er staat aangeduid onder welke voorwaarden er een iteratie wordt uitgevoerd. Deze voorwaarde wordt gecontroleerd zowel vóór de eerste keer dat er mogelijks een iteratie wordt uitgevoerd als elke keer dat een iteratie ten einde komt. Indien de voorwaarde niet geldt, wordt de lus overgeslagen.

referentie naar figuur met loop fragment

In dit hoofdstuk beschouwen we hoe we het vocabularium en de logische theorie die we hebben opgebouwd eerder in de tekst kunnen uitbreiden om het gedrag voorgesteld in een sequentiediagram te modelleren.



Figuur 5.1: Sequentiediagram gebaseerd op het klassediagram van figuur ??

5.1 De keuze voor lineaire tijdscalculus

UML-diagrammen schrijven mogelijke toestanden van, en acties, op softwaresystemen voor. Die systemen kunnen van toestand veranderen tussen tijdstappen. Sequentiediagrammen zijn een manier om te beschrijven hoe zulke veranderingen teweeg kunnen gebracht worden. Tijdens de uitvoering van een sequentiediagram mag het systeem enkel veranderen zoals beschreven door de huidige actie. Daarom hebben we een mechanisme nodig binnen FO(.) dat dynamische systemen en acties op deze kan beschrijven. Tegelijk moet dat mechanisme garanderen dat eigenschappen van het systeem die niet worden beïnvloed door de huidige beschouwde actie van het sequentiediagram niet veranderen. Lineaire tijdscalculus[1], oftewel LTC, voldoet aan deze voorwaarden. Daarom zullen we om sequentiediagrammen uitvoerbaar te maken binnen FO(.) het generatieproces voor het vocabularium en de theorie dat we bekomen zijn in hoofdstuk 2 uitbreiden volgens de principes van LTC.

In de volgende secties werken we deze uitbreiding uit voor het sequentiediagram in figuur ??.

5.2 Uitbreiding van het vocabularium

In LTC is tijd een centraal concept, dus daarom introduceren we allereerst een logisch type $Time \subset \mathbb{N}$. Verder definiëren we een partiële functie *Next(Time)* dat voor alle tijdpunten het volgende tijdpunt geeft behalve voor het laatst mogelijke tijdpunt. We definiëren ook een constante *Start*, wat het eerst mogelijke tijdpunt aanduidt.

Voor elk tijdpunt is het mogelijk dat er een bepaalde instructie van het sequentiediagram wordt uitgevoerd. We duiden deze instructie aan met zijn volgnummer. Deze volgnummers gebruiken we als instructieteller, en daarvoor definiëren we een logisch type $SDPoint \subset \mathbb{N}$.

Om te garanderen dat de instructievolgorde opgelegd door het sequentiediagram gevolgd wordt, maken we deze instructieteller inertieel en introduceren we deze symbolen:

- Het toestandspredicaat: *SDPointAt(Time, SDPoint)*
- Het begintoestandspredicaat: *I_SDPointAt(SDPoint)*
- Het causatiepredicaat: *C_SDPointAt(Time, SDPoint)*

We moeten ook de instanties waarop gehandeld wordt in het sequentiediagram kunnen benoemen. Om te garanderen dat de instanties die vernoemd worden altijd verwijzen naar hetzelfde object, maken we ook de instanties inertieel. Voor *attacker* verkrijgen we dan bijvoorbeeld:

- *AttackerT(Time, Character)*
- *I_AttackerT(Character)*

- *C_AttackerT(Time, Character)*

Het is ook mogelijk dat in een instructie een variabele intern aan het sequentiediagram wordt gedefinieerd. Zo is er instructie 7 waar een return-instructie *defenceVal* definieert en ook instructie 12 die de waarde van *inflicted* definieert als een som van andere variabelen. Deze variabelen willen we ook kunnen benoemen en maken we inertieel. Voor alle zulke variabelen definiëren we ook predicaten zoals hierboven voor *attacker*.

We passen ook de predicaten die overeenkomen met klasseattributen aan. Het kan immers zijn dat de waarde van een attribuut wordt aangepast, zoals in instructie 18 die de waarde van *value* van object *hp* van klasse *Statistic* verandert naar 0. Klasseattributen maken we ook inertieel. Voor *value* in *Statistic* krijgen we dan:

- *Statisticvalue(Time, Statistic, LimitedInt)*
- *I_Statisticvalue(Statistic, LimitedInt)*
- *C_Statisticvalue(Time, Statistic, LimitedInt)*
- En het oncausatiepredicaat: *Cn_Statisticvalue(Time, Statistic, LimitedInt)*

Hier voegen we een oncausatiepredicaat toe omdat het mogelijk is dat een attribuut meer dan één waarde heeft op een bepaald tijdstip. Met dit predicaat geven we aan dat bepaalde waardes die voor een bepaalde tijdstap gelden ongedaan moeten worden gemaakt in de volgende tijdstap.

5.3 Uitbreiden van de theorie

Voor elke inertiële eigenschap van het systeem moeten er twee dingen gebeuren: Toestandszinnen opstellen en voorwaardes voor causatiezinnen en oncausatiezinnen specificeren. Het resultaat is een inductieve definitie die de inertiële predicaten definieert en een inductieve definitie die de causatiepredicaten en oncausatiepredicaten definieert.

5.3.1 Toestandszinnen opstellen

Toestandszinnen worden geschreven in termen van begintoestandspredicaten, causatiepredicaten en oncausatiepredicaten. Ze garanderen dat inertiële eigenschappen enkel veranderen wanneer het ook echt de bedoeling is dat ze veranderen.

Als eerste kijken we naar toestandszinnen voor *SDPointAt*. *I_SDPointAt* geeft aan welke de eerste instructie is die we willen uitvoeren, en daarom schrijven we een definitie die deze overeenkomst uitdrukt:

$$\forall s[SDPoint](SDPointAt(Start, s) \leftarrow I\_SDPointAt(s)). \tag{5.1}$$

De volgende definities gebruiken het causatiepredicaat:

$$\forall t[Time]\forall s[SDPoint](SDPointAt(Next(t), s) \leftarrow C\_SDPointAt(Next(t), s)). \tag{5.2}$$

$$\forall t[Time]\forall s[SDPoint](SDPointAt(Next(t), s) \leftarrow SDPointAt(t, s) \\ \wedge\neg(\exists s1[SDPoint](C\_SDPointAt(Next(t), s1)))). \tag{5.3}$$

Zin 5.2 zorgt ervoor dat de huidige waarde van *SDPointAt* wordt behouden tenzij er een oorzaak is voor verandering.

We schrijven gelijkaardige definities voor de predicaten die overeenkomen met instanties die vernoemd worden in het sequentiediagram (zoals *attacker*).

Voor klasseattributen verloopt dit ook gelijkaardig, maar we wijken af van het formaat van zin 5.3 door als volgt het oncausatiepredicaat te gebruiken:

$$\forall t[Time]\forall s[Statistic]\forall i[LimitedInt](Statisticvalue(Next(t), s, i) \\ \leftarrow Statisticvalue(t, s, i) \wedge \neg Cn\_Statisticvalue(Next(t), s, i)).$$

Voorwaardes voor causatie en oncausatie

We kijken eerst naar klasseattributen. Een aantal ervan worden niet aangepast, wat we bijvoorbeeld neerschrijven voor *range* in *Weapon* als volgt:

$$\forall t[Time]\forall w[Weapon]\forall i[LimitedInt](C\_Weaponrange(t, w, i) \leftarrow false).$$

$$\forall t[Time]\forall w[Weapon]\forall i[LimitedInt](Cn\_Weaponrange(t, w, i) \leftarrow false).$$

Voor de klasseattributen die wel worden aangepast, kijken we naar de instructies die zulke aanpassingen doorvoeren. Voor *value* in *Statistic* zijn dit instructie 18 en 20. We kijken eerst naar de causatiezin en oncausatiezin die volgen uit instructie 18:

$$\forall t[Time]\forall s[Statistic](C\_Statisticvalue(t, s, 0) \leftarrow SDPointAt(t, 18) \wedge HpT(t, s).$$

$$\forall t[Time]\forall s[Statistic]\forall i[LimitedInt](Cn\_Statisticvalue(Next(t), s, v) \\ \leftarrow SDPointAt(Next(t), 18) \wedge HpT(t, s) \wedge Statisticvalue(t, s, i) \wedge \neg(i = 0).$$

Aangezien in instructie 18 de instantie *hp* wordt aangesproken, gebruiken we *HpT* om te verzekeren dat de waarde van het juiste logisch object wordt veranderd. *value* kan ook maar één waarde tegelijk hebben, en daarom schrijven we een oncausatiezin om te verzekeren dat de vorige waarde wordt gewist.

Kijken we nu naar de definities die voortvloeien uit instructie 20:

$$\forall t[Time]\forall s[Statistic]\forall i[LimitedInt](C\_Statisticvalue(t,s,i) \leftarrow SDPointAt(t,20) \wedge HpT(t,s) \wedge NewHpT(t,i)).$$

$$\forall t[Time]\forall s[Statistic]\forall i[LimitedInt](Cn\_Statisticvalue(Next(t),s,i) \leftarrow SDPointAt(Next(t),20) \wedge HpT(t,s) \wedge Statisticvalue(t,s,i) \wedge \neg NewHpT(Next(t),i)).$$

Het verschil hier is dat we *NewHpT* erbij betrekken omdat we de waarde van *hp* veranderen naar de waarde van *newHp* in plaats van het te veranderen naar 0.

Het volgende waar we naar kijken zijn de causatiezinnen voor *SDPointAt*. Wat we hier willen uitdrukken is dat normaal gezien tussen instructies de instructieteller telkens met één wordt verhoogd, tenzij een grens van een *if-else*-constructie of een lus is bereikt. In dat geval kan het zijn dat de instructieteller verspringt afhankelijk van de voorwaarde die vernoemd wordt voor zulke constructies.

Voor deze sequentiediagram krijgen we:

$$\forall t[Time]\forall s[SDPoint](C\_SDPointAt(Next(t),(s+1) \leftarrow SDPointAt(t,s) \wedge \neg((s=17) \vee (s=19))). \tag{5.4}$$

$$\forall t[Time](C\_SDPointAt(Next(t),18) \leftarrow SDPointAt(t,17) \wedge (\exists i[LimitedInt](NewHpT(t,i) \wedge i <= 0))). \tag{5.5}$$

$$\forall t[Time](C\_SDPointAt(Next(t),20) \leftarrow SDPointAt(t,17) \wedge (\exists i[LimitedInt](NewHpT(t,i) \wedge i > 0))). \tag{5.6}$$

$$\forall t[Time](C\_SDPointAt(Next(t),21) \leftarrow SDPointAt(t,19) \vee SDPointAt(t,20)). \tag{5.7}$$

Zin 5.4 verzekert het juiste gedrag van de instructieteller, namelijk dat hij doorgaans met één wordt verhoogd tussen tijdstappen. De uitzonderingen worden hier ook opgelijst; in dit geval verspringt de teller wanneer men het begin van de *if-else*-constructie tegenkomt en wanneer het einde van het *if*-deel is bereikt. Zinnen 5.5 en 5.6 controleren de voorwaarde voor de uitvoering van het *if*- en *else*-deel en selecteren wat correct is. Zin 5.7 zegt dat zowel het *if*-deel als het *else*-deel uitkomen op de instructie die direct volgt op de *if-else*-constructie.

Als laatste zijn er de causatiezinnen voor de verscheidene variabelen die worden aangemaakt en aangesproken in het sequentiediagram. Een aantal van deze variabelen veranderen niet doorheen de uitvoering van het sequentiediagram en er wordt verondersteld dat deze al bekend zijn vóór de uitvoering begint. Deze variabelen zijn diegenen die betrokken zijn bij de eerste instructie: *attacker*, de instantie die de eerste oproep ontvangt, en *target* en *weapon*, die als parameter worden opgegeven.

Voor de andere variabelen wordt er een causatiezin toegevoegd voor elke instructie die een waarde toekent aan die variabele. Als voorbeeld bekijken we instructie 3:

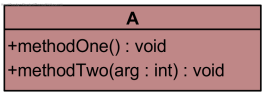
$$\forall t[Time]\forall d[Statistic](C\_DefenceT(t,d) \leftarrow SDPointAt(t,3) \wedge \exists c[Character](TargetT(t,c) \wedge CharacterandStatistic(c,d) \wedge Statisticname(t,d,"defence"))).$$

De zin drukt uit dat de getter wordt opgeroepen op *target* en dat er wordt gevraagd naar een instantie van *Statistic* dat in verband staat met *target* en als naam "defence"heeft. Die instantie wordt dan als waarde toegekend aan de variabele *defence*.

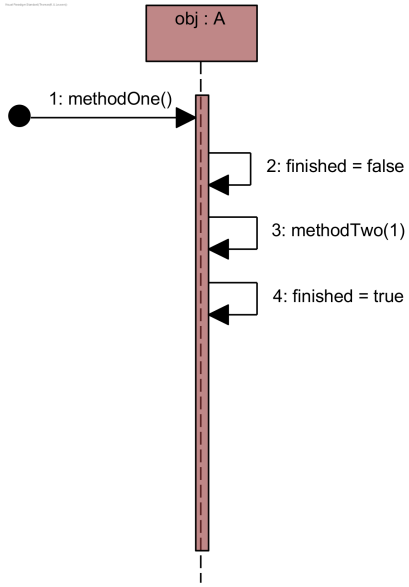
Zie appendix C voor het volledig model voor het gedrag van het sequentiediagram in figuur ??.

5.4 Interactie tussen meerdere sequentiediagrammen

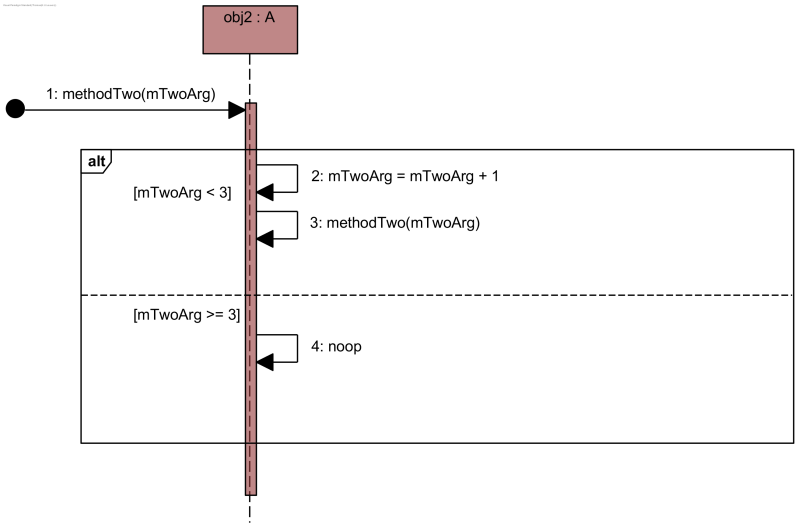
In een project stelt men doorgaans meerdere sequentiediagrammen op die elkaar ook kunnen oproepen. Men gebruikt soms ook recursie in deze diagrammen. In deze sectie beschrijven we hoe we het oproepen van andere sequentiediagrammen en een recursiemechanisme ondersteunen.



Figuur 5.2: Klasse gebruikt in voorbeeld over recursie



(a) Sequentiediagram voor methodOne()



(b) Sequentiediagram voor methodTwo()

Figuur 5.3: Sequentiediagrammen voor klasse A in figuur 5.2

In de volgende subsecties gebruiken we het voorbeeld uitgebeeld in figuren 5.2 en 5.3 om de gebruikte principes te illustreren.

5.4.1 Aanpassingen aan *SDPoint*

Het is niet meer voldoende om *SDPoint*s te modelleren als natuurlijke getallen aangezien elk diagram zijn eigen *SDPoint*s heeft. Om de *SDPoint*s horende bij elk diagram van elkaar te kunnen onderscheiden, maken we van het logisch type *SDPoint* nu een constructed type in IDP. We gebruiken als patroon voor de naamgeving van elk logisch object $< diagramnaam > \_ < instructienummer >$. Voor het voorbeeld krijgen we o.a. *methodOne*_2 en *methodTwo*_3. We voegen ook een nieuwe functie toe aan het vocabularium dat gegeven een *SDPoint* het volgende *SDPoint* teruggeeft, namelijk $NextSD(SDPoint) : SDPoint$.

Een andere aanpassing is dat we een virtuele *SDPoint* inpassen na elke instructie voor een oproep. Het naamgevingspatroon hiervoor is $< diagramnaam > \_ < instructienummer > post$. Aangezien de derde instructie in figuur 5.3a een oproep is, krijgen we *methodOne*_3*post*. Met deze toevoeging ontkoppelen we twee zaken die bij een oproep komen kijken: enerzijds dat het mogelijk is dat het resultaat van een oproep wordt toegekend aan een variabele; en anderzijds dat na een oproep er mogelijks een alt-fragment of lusfragment volgt, en dat het dus vóór de oproep niet duidelijk is welke de volgende instructie is na de uitvoering van de oproep. Het zou niet mogelijk zijn om deze twee zaken op een correcte manier af te handelen zonder een oproepinstructie op deze manier op te splitsen.

5.4.2 Het stapelmechanisme

Om oproepen van andere sequentiediagrammen correct uit te voeren, moet de theorie bijhouden welke variabelen er bestaan tijdens een bepaalde oproep. Bovendien moet de theorie voor recursieve oproepen ook de waardes van een set variabelen kunnen bewaren vóór een oproep en die waardes herstellen na een oproep. Hiertoe ontwerpen we een stapelmechanisme. Er gebeuren volgende aanpassingen aan het vocabularium:

- Toevoeging van het logisch type $StackLevel \subset \mathbb{N}$: Dit stelt de oproepdiepte van oproep voor.
- Toevoeging van een inertiële functie $CurrentStackLevel(Time) : StackLevel$: De oproepdiepte op een bepaald tijdstip.
- Toevoeging van een intertieel predicaaat $ReturnPoint(Time, StackLevel, SDPoint)$: Op een bepaald tijdstip, de *SDPoint* naar waar de uitvoering moet terugkeren wanneer de laatste instructie voor de gegeven *StackLevel* bereikt is.
- Alle diagramvariabelen worden nu gemodelleerd door een ternair predicaat dat nu ook de oproepdiepte in rekening neemt. Voor het voorbeeld krijgen we dus bijvoorbeeld $FinishedT(Time, StackLevel, bool)$.

De volgende subsecties beschrijven hoe we in het definitieblok voor de causatiezinnen dit stapelmechanisme gebruiken.

Causatiezinnen voor *SDPointAt/2*

Oproepinstructies betekenen bijkomende uitzonderingen op het normale verloop van *SDPoints* naast deze die voortkomen uit alt– en lusfragmenten. In dit geval zijn $methodOne\_3$, $methodTwo\_3$, $methodOne\_5$, $methodTwo\_5$ en $finished$ de nieuwe uitzonderingen. $methodOne\_5$ en $methodTwo\_5$ zijn ook *SDPoint*s die niet in het diagram terug te vinden zijn, maar die we zelf toevoegen. Daarmee markeren we het einde van elk van de diagrammen. $finished$ is een speciale *SDPoint* die het einde van de uitvoering aanduidt. De zin die het normale verloop van *SDPoint*s regelt wordt dus:

$$\begin{aligned}&\forall t[Time]\forall s[SDPoint](C\_SDPointAt(Next(t), NextSD(s) \leftarrow \\ &SDPointAt(t,s) \wedge \neg((s = methodOne\_3) \vee (s = methodOne\_5) \\ &\vee (s = methodTwo\_1) \vee (s = methodTwo\_3) \vee (s = methodTwo\_3post) \\ &\vee (methodTwo\_4) \vee (methodTwo\_5) \vee (s = finished)).\end{aligned} \tag{5.8}$$

De uitzonderingen als resultaat van een oproep worden als volgt gemodelleerd:

$$\forall t[Time](C\_SDPointAt(Next(t), methodTwo\_1) \leftarrow SDPointAt(t, methodOne\_3). \tag{5.9}$$

$$\forall t[Time](C\_SDPointAt(Next(t), methodTwo\_1) \leftarrow SDPointAt(t, methodOne\_3). \tag{5.10}$$

Zin 5.9 resulteert uit instructie 3 van het diagram voor *methodOne* en zin 5.10 resulteert uit instructie 3 van het diagram voor *methodTwo*.

De tweede aanpassing is dat er een terugkeer moet gebeuren wanneer het einde van een sequentiediagram is bereikt. Hiervoor maken we gebruik van $ReturnPoint/3$:

$$\begin{aligned}&\forall t[Time]\forall s[SDPoint](C\_SDPointAt(Next(t), s) \leftarrow \\ &ReturnPoint(t, CurrentStackLevel(t), s) \wedge (SDPointAt(t, methodOne\_5) \\ &\vee SDPointAt(t, methodTwo\_5))).\end{aligned} \tag{5.11}$$

Deze zin drukt uit dat het terugkeerpunt dat is genoteerd voor deze oproepdiepte wordt genomen als de volgende *SDPoint* wanneer het einde van een sequentiediagram is bereikt, in dit geval $methodOne\_5$ of $methodTwo\_5$. We schrijven geen nieuwe zin voor $finished$ omdat er niets op volgt.

Causatiezinnen voor *ReturnPoint/3*

Wanneer de uitvoering een oproep bereikt, willen we voor de nieuwe oproepdiepte dat het terugkeerpunt wordt gezet naar de *SDPoint* direct na de oproepinstructie. Daarmee krijgen we de volgende twee zinnen:

$$\forall t[Time]\forall st[StackLevel](C\_ReturnPoint(Next(t), st, methodOne\_3post) \leftarrow (CurrentStackLevel(t) = (st - 1)) \wedge SDPointAt(t, methodOne\_3)). \quad (5.12)$$

$$\forall t[Time]\forall st[StackLevel](C\_ReturnPoint(Next(t), st, methodTwo\_3post) \leftarrow (CurrentStackLevel(t) = (st - 1)) \wedge SDPointAt(t, methodOne\_3)). \quad (5.13)$$

We willen ook dat een terugkeerpunt verdwijnt eenmaal dat het wordt gebruikt aan het einde van een diagram. Daarom schrijven we de volgende voorwaarde neer voor het oncausatiepredicaat voor *ReturnPoint/3*:

$$\forall t[Time]\forall st[StackLevel]\forall sd[SDPoint](Cn\_ReturnPoint(Next(t), st, sd) \leftarrow (CurrentStackLevel(t) = st) \wedge ReturnPoint(t, st, sd) \wedge (SDPointAt(t, methodOne\_5) \vee SDPointAt(t, methodTwo\_5))). \quad (5.14)$$

Zinnen 5.11 en 5.14 samen garanderen dat terugkeerpunten gebruikt worden en verdwijnen wanneer het einde van een diagram is bereikt.

Causatiezinnen voor *CurrentStackLevel(Time) : StackLevel*

De oproepdiepte moet toenemen wanneer een oproepinstructie wordt uitgevoerd en afnemen wanneer het einde van een diagram is bereikt. Deze respectievelijke gevallen modelleren we als volgt:

$$\forall t[Time]\forall st[StackLevel](C\_CurrentStackLevel(Next(t), st) \leftarrow (CurrentStackLevel(t) = (st - 1)) \wedge (SDPointAt(t, methodOne\_3) \vee SDPointAt(t, methodTwo\_3))). \quad (5.15)$$

$$\forall t[Time]\forall st[StackLevel](C\_CurrentStackLevel(Next(t), st) \leftarrow (CurrentStackLevel(t) = (st + 1)) \wedge (SDPointAt(t, methodOne\_5) \vee SDPointAt(t, methodTwo\_5))). \quad (5.16)$$

Causatiezinnen voor oproepobjecten en parameters

Er komen twee nieuwe soorten variabelen bij: Objecten waarvan een methode wordt opgeroepen en parameters van een methode. In het sequentiediagram voor *methodTwo* is *obj2* de naam van het object dat het eerste bericht ontvangt. Wanneer het diagram voor *methodOne* deze methode oproept, moet *obj2* dus gezet worden naar de juiste waarde, in dit geval *obj* omdat *obj* de methode oproept op zichzelf.

Een gelijkaardig geval doet zich voor bij de recursieve oproep in het diagram voor *methodTwo*. Daarom krijgen we de volgende zinnen voor *C_Obj2T/3*:

$$\begin{aligned}&\forall t[Time]\forall s[StackLevel]\forall obj[A](C\_Obj2T(Next(t), s, obj) \leftarrow \\ &(CurrentStackLevel(t) = (s-1)) \wedge SDPointAt(t, methodOne\_3) \\ &\wedge ObjT(t, (s-1), obj)). \end{aligned} \tag{5.17}$$

$$\begin{aligned}&\forall t[Time]\forall s[StackLevel]\forall obj[A](C\_Obj2T(Next(t), s, obj) \leftarrow \\ &(CurrentStackLevel(t) = (s-1)) \wedge SDPointAt(t, methodTwo\_3) \\ &\wedge Obj2T(t, (s-1), obj)). \end{aligned} \tag{5.18}$$

mTwoArg in het diagram voor *methodTwo* is een parameter van *methodTwo* dat ook aangesproken wordt in het diagram zelf. In *methodOne* wordt *mTwoArg* gelijkgesteld aan 1 terwijl in *methodTwo* deze eerst met één wordt verhoogd. Daarom krijgen we de drie volgende zinnen:

$$\begin{aligned}&\forall t[Time]\forall s[StackLevel](C\_MTwoArgT(Next(t), s, 1) \leftarrow \\ &(CurrentStackLevel(t) = (s-1)) \wedge SDPointAt(t, methodOne\_3)). \end{aligned} \tag{5.19}$$

$$\begin{aligned}&\forall t[Time]\forall s[StackLevel]\forall n[int](C\_MTwoArgT(Next(t), s, 1) \leftarrow \\ &(CurrentStackLevel(t) = (s-1)) \wedge SDPointAt(t, methodTwo\_3) \\ &\wedge MTwoArg(t, (s-1), n)). \end{aligned} \tag{5.20}$$

$$\begin{aligned}&\forall t[Time]\forall s[StackLevel]\forall n[int](C\_MTwoArgT(Next(t), s, 1) \leftarrow \\ &(CurrentStackLevel(t) = s) \wedge SDPointAt(t, methodTwo\_2) \\ &\wedge (\exists n1[int](MTwoArg(t, s, n1) \wedge (n = n1 + 1)))). \end{aligned} \tag{5.21}$$

Zin 5.21 demonstreert dat ook buiten oproepinstructies of een terugkeer uit een diagram wordt gekeken naar het oproepniveau. Er wordt immers enkel gekeken of geschreven naar de waarde van de 'versie' van de variabele die overeenkomt met het huidige oproepniveau. Dit komt ook terug bij de variabelen die niet het oproepobject of een parameter van een methode voorstellen.

Uitkomst van de beschreven procedure

Er zijn geen noemenswaardige veranderingen aan hoe we de toestandszinnen opstellen. Bijlage D bevat de uitkomst van de procedure die we hebben beschreven in deze sectie.

Bijlagen

Bijlage A

IDP-bestand resulterend uit de procedure beschreven in hoofdstuk 2

```
vocabulary V {
    type LimitedInt = { 1..18 } isa int
    type LimitedFloat = { 0.0; 0.5; 1.0; 1.5; 2.0; 2.5; 3.0; 3.5; 4.0; 4.5;
        5.0; 5.5; 6.0; 6.5; 7.0; 7.5; 8.0; 8.5 } isa float
    type LimitedString  = { "SVObSVpOwNsLeM1TCYkx"; "s0mJOUkvFuOYxoypf0e2"; "
        ucPRTRrfWBdnx8IbebrH"; "IIsx8mkhNB6tFKXhIh01"; "c8FoPQm8gzGloJi352R6";
         "QOwcTPcuxqohdJOOoYI5"; "nNhaMFI1sNq4FM9g9PFK"; "THmoxPvd1k7axZ9Rx3Vo
        "; "psOJIEnr6CUFa2S1shdP"; "2ykrKTZkDAopHEMGzBgp"; "
        YrUOvIjCy2OZLs39PE5t"; "LXDiF7O705qElFzEF3WJ"; "NW3yPaSUa5NJERB5bpd0";
         "NzD42F9XGUvbUNaZHU0q"; "s9Iudo9RU7iwdNeSJi8t"; "iNl5Hkr0r9krSOlg2KER
        "; "bkl5G0Ix9OUrFJfV1H7P"; "kC4BfZXQ4VDHxUmJ1O5G" } isa string
    type bool constructed from { true, false }
    type void constructed from { void }
    type Object
    type ClassObject constructed from { DerivedStatisticStrategy,
        DerivedStatistic, WeaponType, CharacterClass, Inventory, Weapon,
        Character, SupportAffinity, WeaponLevel, Statistic, Item }
    RuntimeClass(ClassObject, Object)
    StaticClass(ClassObject, Object)
    IsDirectSupertypeOf(ClassObject, ClassObject)
    IsSupertypeOf(ClassObject, ClassObject)

    CharacterClassname(Object,LimitedString)
    Weapondamage(Object,LimitedInt)
    Weaponweight(Object,LimitedInt)
    Weaponrange(Object,LimitedInt)
    Charactername(Object,LimitedString)
    SupportAffinityname(Object,LimitedString)
    Statisticname(Object,LimitedString)
    Itemvalue(Object,LimitedInt)
    Itemname(Object,LimitedString)


    DerivedStatisticStrategygetValue(Object,LimitedInt)
    CharacterdetermineDamageFrom(Object,Object,LimitedInt)
    StatisticgetValue(Object,LimitedInt)

```

```
30      StatisticandCharacterClass(Object,Object)
31      DerivedStatisticStrategyandStatistic(Object,Object)
32      WeaponTypeandWeaponLevel(Object,Object)
33      CharacterandStatistic(Object,Object)
34      CharacterandInventory(Object,Object)
35      CharacterandCharacter(Object,Object)
36      WeaponTypeandWeapon(Object,Object)
37      DerivedStatisticStrategyandDerivedStatistic(Object,Object)
38      StatisticandWeaponType(Object,Object)
39      ItemandInventory(Object,Object)
40      ItemandCharacterClass(Object,Object)
41      CharacterandCharacterClass(Object,Object)
42      CharacterClassandCharacterClass(Object,Object)
43      CharacterandSupportAffinity(Object,Object)
44
45  }
46
47  theory T:V {
48      {
49          ! x y : IsDirectSupertypeOf(x, y) <- x = Statistic & y = WeaponLevel.
50          ! x y : IsDirectSupertypeOf(x, y) <- x = Item & y = Weapon.
51          ! x y : IsDirectSupertypeOf(x, y) <- x = Statistic & y =
                DerivedStatistic.
52      }
53
54      ! o : ?1 x : RuntimeClass(x, o).
55
56      {
57          ! x y : IsSupertypeOf(x, y) <- IsDirectSupertypeOf(x, y).
58          ! x y : IsSupertypeOf(y, x) <- ? z : IsSupertypeOf(y, z) &
                IsSupertypeOf(z, x).
59
60          ! x o : StaticClass(x, o) <- RuntimeClass(x, o).
61          ! x y o: StaticClass(y, o) <- RuntimeClass(x, o) & IsSupertypeOf(y, x)
                .
62      }
63
64      ! o x : CharacterClassname(o, x) => StaticClass(CharacterClass, o).
65      ! o : StaticClass(CharacterClass, o) => ?1 x : CharacterClassname(o, x).
66
67      ! o x : Weapondamage(o, x) => StaticClass(Weapon, o).
68      ! o : StaticClass(Weapon, o) => ?1 x : Weapondamage(o, x).
69
70      ! o x : Weaponweight(o, x) => StaticClass(Weapon, o).
71      ! o : StaticClass(Weapon, o) => ?1 x : Weaponweight(o, x).
72
73      ! o x : Weaponrange(o, x) => StaticClass(Weapon, o).
74      ! o : StaticClass(Weapon, o) => ?1 x : Weaponrange(o, x).
75
76      ! o x : Charactername(o, x) => StaticClass(Character, o).
77      ! o : StaticClass(Character, o) => ?1 x : Charactername(o, x).
78
79      ! o x : SupportAffinityname(o, x) => StaticClass(SupportAffinity, o).
80      ! o : StaticClass(SupportAffinity, o) => ?1 x : SupportAffinityname(o, x).
81
82      ! o x : Statisticname(o, x) => StaticClass(Statistic, o).
83      ! o : StaticClass(Statistic, o) => ?1 x : Statisticname(o, x).
84
85      ! o x : Itemvalue(o, x) => StaticClass(Item, o).
86      ! o : StaticClass(Item, o) => ?1 x : Itemvalue(o, x).
87
88      ! o x : Itemname(o, x) => StaticClass(Item, o).
89      ! o : StaticClass(Item, o) => ?1 x : Itemname(o, x).
```

```
90
91      ! o r : DerivedStatisticStrategygetValue(o, r) => (StaticClass(
            DerivedStatisticStrategy, o)).
92      ! o : (StaticClass(DerivedStatisticStrategy, o)) => (?1 r :
            DerivedStatisticStrategygetValue(o, r)).
93
94      ! o p1 r : CharacterdetermineDamageFrom(o, p1, r) => (StaticClass(
            Character, o) & (StaticClass(Weapon, p1))).
95      ! o p1 : (StaticClass(Character, o) & (StaticClass(Weapon, p1))) => (?1 r
            : CharacterdetermineDamageFrom(o, p1, r)).
96
97      ! o r : StatisticgetValue(o, r) => (StaticClass(Statistic, o)).
98      ! o : (StaticClass(Statistic, o)) => (?1 r : StatisticgetValue(o, r)).
99
100
101     ! o1 o2 : StatisticandCharacterClass(o1,o2) => ((StaticClass(Statistic, o1
            )) & (StaticClass(CharacterClass, o2))).
102     ! o2 : ((StaticClass(CharacterClass, o2))) => (1 =< #{o1 :
            StatisticandCharacterClass(o1,o2)}).
103     ! o1 : ((StaticClass(Statistic, o1))) => ?1 o2 :
            StatisticandCharacterClass(o1,o2).
104
105     ! o1 o2 : DerivedStatisticStrategyandStatistic(o1,o2) => ((StaticClass(
            DerivedStatisticStrategy, o1)) & (StaticClass(Statistic, o2))).
106     ! o1 : ((StaticClass(DerivedStatisticStrategy, o1))) => (1 =< #{o2 :
            DerivedStatisticStrategyandStatistic(o1,o2)}).
107
108     ! o1 o2 : WeaponTypeandWeaponLevel(o1,o2) => ((StaticClass(WeaponType, o1)
            ) & (StaticClass(WeaponLevel, o2))).
109     ! o2 : ((StaticClass(WeaponLevel, o2))) => ?1 o1 :
            WeaponTypeandWeaponLevel(o1,o2).
110
111     ! o1 o2 : CharacterandStatistic(o1,o2) => ((StaticClass(Character, o1)) &
            (StaticClass(Statistic, o2))).
112     ! o2 : ((StaticClass(Statistic, o2))) => ?1 o1 : CharacterandStatistic(o1,
            o2).
113     ! o1 : ((StaticClass(Character, o1))) => (1 =< #{o2 :
            CharacterandStatistic(o1,o2)}).
114
115     ! o1 o2 : CharacterandInventory(o1,o2) => ((StaticClass(Character, o1)) &
            (StaticClass(Inventory, o2))).
116     ! o2 : ((StaticClass(Inventory, o2))) => ?1 o1 : CharacterandInventory(o1,
            o2).
117     ! o1 : ((StaticClass(Character, o1))) => ?1 o2 : CharacterandInventory(o1,
            o2).
118
119     ! o1 o2 : CharacterandCharacter(o1,o2) => ((StaticClass(Character, o1)) &
            (StaticClass(Character, o2))).
120
121     ! o1 o2 : WeaponTypeandWeapon(o1,o2) => ((StaticClass(WeaponType, o1)) & (
            StaticClass(Weapon, o2))).
122     ! o2 : ((StaticClass(Weapon, o2))) => (#{o1 : WeaponTypeandWeapon(o1,o2)}
            =< 1).
123
124     ! o1 o2 : DerivedStatisticStrategyandDerivedStatistic(o1,o2) => ((
            StaticClass(DerivedStatisticStrategy, o1)) & (StaticClass(
            DerivedStatistic, o2))).
125     ! o2 : ((StaticClass(DerivedStatistic, o2))) => ?1 o1 :
            DerivedStatisticStrategyandDerivedStatistic(o1,o2).
126     ! o1 : ((StaticClass(DerivedStatisticStrategy, o1))) => ?1 o2 :
            DerivedStatisticStrategyandDerivedStatistic(o1,o2).
127
```

```
128     ! o1 o2 : StatisticandWeaponType(o1,o2) => ((StaticClass(Statistic, o1)) &
            (StaticClass(WeaponType, o2))).
129     ! o2 : ((StaticClass(WeaponType, o2))) => (#{o1 : StatisticandWeaponType(
            o1,o2)} =< 1).
130
131     ! o1 o2 : ItemandInventory(o1,o2) => ((StaticClass(Item, o1)) & (
            StaticClass(Inventory, o2))).
132     ! o2 : ((StaticClass(Inventory, o2))) => (#{o1 : ItemandInventory(o1,o2)}
            =< 5).
133     ! o1 : ((StaticClass(Item, o1))) => (#{o2 : ItemandInventory(o1,o2)} =< 1)
            .
134
135     ! o1 o2 : ItemandCharacterClass(o1,o2) => ((StaticClass(Item, o1)) & (
            StaticClass(CharacterClass, o2))).
136
137     ! o1 o2 : CharacterandCharacterClass(o1,o2) => ((StaticClass(Character, o1
            )) & (StaticClass(CharacterClass, o2))).
138     ! o1 : ((StaticClass(Character, o1))) => ?1 o2 :
            CharacterandCharacterClass(o1,o2).
139
140     ! o1 o2 : CharacterClassandCharacterClass(o1,o2) => ((StaticClass(
            CharacterClass, o1)) & (StaticClass(CharacterClass, o2))).
141
142     ! o1 o2 : CharacterandSupportAffinity(o1,o2) => ((StaticClass(Character,
            o1)) & (StaticClass(SupportAffinity, o2))).
143     ! o2 : ((StaticClass(SupportAffinity, o2))) => ?1 o1 :
            CharacterandSupportAffinity(o1,o2).
144     ! o1 : ((StaticClass(Character, o1))) => ?1 o2 :
            CharacterandSupportAffinity(o1,o2).
145 }
146
147 structure thestruct : V {
148     Object = { 1..18}
149 }
150
151 procedure main() {
152     print(modelexpand(T,thestruct)[1])
153 }
```

Codebestand 0.4.3

Bijlage B

IDP-bestand resulterend uit de procedure beschreven in hoofdstuk 3

```
vocabulary V {
    type ClassObject constructed from { DerivedStatisticStrategy,
        DerivedStatistic, WeaponType, CharacterClass, Inventory, Weapon,
        Character, SupportAffinity, WeaponLevel, Statistic, Item, Loose, A, B,
         C, D}
    type PrimitiveType constructed from { boolean, byte, character, double,
        floating, integer, long, short, astring, void}
    IsSupertypeOf(ClassObject, ClassObject)

    type LimitedInt = {1 .. 21} isa int

    BiAssoc(ClassObject, ClassObject)
    BiAssocLow(ClassObject, ClassObject, ClassObject, LimitedInt)
    BiAssocHigh(ClassObject, ClassObject, ClassObject, LimitedInt)

    ManyToMany(ClassObject, ClassObject)
    LooseClass(ClassObject)
    SubclassMorePermissiveMult(ClassObject, ClassObject, ClassObject)
}

theory T:V {

    // ------- BAD DESIGN THAT MAY OCCUR IN UML -------

    // many-to-many associations
    ! x [ClassObject] y [ClassObject] : ManyToMany(x, y) <=> (BiAssoc(x, y) &
        ~ (? z [LimitedInt] : BiAssocHigh(x, y, x, z)) & ~ (? z [LimitedInt] :
         BiAssocHigh(x, y, y, z))).

    // classes that are not associated with any other class
    ! x [ClassObject] : LooseClass(x) <=> ~ ((? y [ClassObject] : BiAssoc(x, y
        )) | (? s [ClassObject] y [ClassObject] : IsSupertypeOf(s, x) & (
        BiAssoc(s, y)))).

    ! x [ClassObject] y [ClassObject] : SubclassMorePermissiveMult(x, y, x)
        <=> ((? sx [ClassObject] : IsSupertypeOf(sx, x)
                                                                          &
```

```
((
BiAsso
(
sx
,

y
)

&

((?

z1

[
Limite
]

z2

[
Limite
]

:

BiAsso
(
x
,

y
,

x
,

z1
)

&

BiAsso
(
sx
,

y
,

sx
,

z2
)

&

z1

<
```

```
z2
)
```

29

```
30                                                                    |
31                                                                      (?
                                                                            sy
                                                                            [
                                                                            Cl
                                                                            ]
                                                                            :
                                                                            Is
                                                                            (
                                                                            sy
                                                                            ,
                                                                            y
                                                                            )
32                                                                          &
```

```
                                                                                &
                                                                                BiAsso
                                                                                (
                                                                                sx
                                                                                ,
                                                                                sy
                                                                                ,
                                                                                sx
                                                                                ,
                                                                                z2
                                                                                )
                                                                                &
                                                                                z1
                                                                                <
                                                                                z2
                                                                                )
33
```

```
34                                                                  |
                                                                        (?
                                                                        sy
                                                                        [
                                                                        ClassObjec
                                                                        ]
                                                                        :
                                                                        IsSupertyp
                                                                        (
                                                                        sy
                                                                        ,
                                                                        y
                                                                        )
35                                                                          &
                                                                                Bi
                                                                                (
                                                                                x
                                                                                ,
                                                                                sy
                                                                                )
                                                                                &
                                                                                ((
                                                                                z1
                                                                                [
                                                                                Li
                                                                                ]
```

```
z2
[
LimitedInt
]
:
BiAssocLow
(
x
,
y
,
x
,
z1
)
&
BiAssocLow
(
x
,
sy
,
x
,
z2
)
&
z1
<
z2
)
```

```
37      // ------- INFORMATION FROM DIAGRAM -------
38
39      // class hierarchy
40      {
41          IsSupertypeOf(Statistic, WeaponLevel) <- .
42          IsSupertypeOf(Item, Weapon) <- .
43          IsSupertypeOf(Statistic, DerivedStatistic) <- .
44          IsSupertypeOf(A,B) <- .
45          IsSupertypeOf(C,D) <- .
46          IsSupertypeOf(A,E) <- .
47          IsSupertypeOf(C,F) <- .
48      }
49
50      // associations between classes
51      {
52          BiAssoc(A, C) <- .
53          BiAssocLow(A, C, A, 3) <- .
```

```
54          BiAssocHigh(A, C, A, 5) <- .
55          BiAssocLow(A, C, C, 10) <- .
56          BiAssocHigh(A, C, C, 20) <- .
57
58          BiAssoc(B, D) <- .
59          BiAssocLow(B, D, B, 1) <- .
60          BiAssocHigh(B, D, B, 6) <- .
61          BiAssocLow(B, D, D, 9) <- .
62          BiAssocHigh(B, D, D, 21) <- .
63
64          BiAssoc(Statistic, CharacterClass) <- .
65          BiAssocLow(Statistic, CharacterClass, Statistic, 1) <- .
66          BiAssocLow(Statistic, CharacterClass, CharacterClass, 1) <- .
67          BiAssocHigh(Statistic, CharacterClass, CharacterClass, 1) <- .
68
69          BiAssoc(DerivedStatisticStrategy, Statistic) <- .
70          BiAssocLow(DerivedStatisticStrategy, Statistic, Statistic, 1) <- .
71          BiAssoc(WeaponType, WeaponLevel) <- .
72
73          BiAssocLow(WeaponType, WeaponLevel, WeaponType, 1) <- .
74          BiAssocHigh(WeaponType, WeaponLevel, WeaponType, 1) <- .
75
76          BiAssoc(Character, Statistic) <- .
77          BiAssocLow(Character, Statistic, Character, 1) <- .
78          BiAssocHigh(Character, Statistic, Character, 1) <- .
79          BiAssocLow(Character, Statistic, Statistic, 1) <- .
80
81          BiAssoc(Character, Inventory) <- .
82          BiAssocLow(Character, Inventory, Character, 1) <- .
83          BiAssocHigh(Character, Inventory, Character, 1) <- .
84          BiAssocLow(Character, Inventory, Inventory, 1) <- .
85          BiAssocHigh(Character, Inventory, Inventory, 1) <- .
86
87          BiAssoc(Character, Character) <- .
88
89          BiAssoc(WeaponType, Weapon) <- .
90          BiAssocHigh(WeaponType, Weapon, WeaponType, 1) <- .
91
92          BiAssoc(DerivedStatistic, DerivedStatisticStrategy) <- .
93          BiAssocLow(DerivedStatistic, DerivedStatisticStrategy,
                DerivedStatistic, 1) <- .
94          BiAssocHigh(DerivedStatistic, DerivedStatisticStrategy,
                DerivedStatistic, 1) <- .
95          BiAssocLow(DerivedStatistic, DerivedStatisticStrategy,
                DerivedStatisticStrategy, 1) <- .
96          BiAssocHigh(DerivedStatistic, DerivedStatisticStrategy,
                DerivedStatisticStrategy, 1) <- .
97
98          BiAssoc(Statistic, WeaponType) <- .
99          BiAssocHigh(Statistic, WeaponType, Statistic, 1) <- .
100
101         BiAssoc(Item, Inventory) <- .
102         BiAssocHigh(Item, Inventory, Item, 5) <- .
103         BiAssocHigh(Item, Inventory, Inventory, 1) <- .
104
105         BiAssoc(Item, CharacterClass) <- .
106
107         BiAssoc(Character, CharacterClass) <- .
108         BiAssocLow(Character, CharacterClass, CharacterClass, 1) <- .
109         BiAssocHigh(Character, CharacterClass, CharacterClass, 1) <- .
110
111         BiAssoc(CharacterClass, CharacterClass) <- .
112
```

```
113         BiAssoc(Character, SupportAffinity) <- .
114         BiAssocLow(Character, SupportAffinity, Character, 1) <- .
115         BiAssocHigh(Character, SupportAffinity, Character, 1) <- .
116         BiAssocLow(DerivedStatistic, SupportAffinity, SupportAffinity, 1) <- .
117         BiAssocHigh(DerivedStatistic, SupportAffinity, SupportAffinity, 1) <-
                .
118
119         ! x [ClassObject] y [ClassObject] : BiAssoc(y, x) <- BiAssoc(x, y).
120         ! x [ClassObject] y [ClassObject] z [nat] : BiAssocHigh(y, x, x, z) <-
                BiAssocHigh(x, y, x, z).
121         ! x [ClassObject] y [ClassObject] z [nat] : BiAssocHigh(y, x, y, z) <-
                BiAssocHigh(x, y, y, z).
122         ! x [ClassObject] y [ClassObject] z [nat] : BiAssocLow(y, x, x, z) <-
                BiAssocLow(x, y, x, z).
123         ! x [ClassObject] y [ClassObject] z [nat] : BiAssocLow(y, x, y, z) <-
                BiAssocLow(x, y, y, z).
124     }
125
126
127         //! x [ClassObject] y [ClassObject] : BiAssocDef(x, y) <=> BiAssocDef(
                y, x).
128 }
129
130 theory U:V {
131     ~ (? z [ClassObject] : BiAssoc(Loose, z)).
132 }
133
134 structure thestruct:V {
135 }
136
137 procedure main() {
138     print(modelexpand(T,thestruct)[1])
139 }
```

chap–rol–idp/defs.idp

Bijlage C

IDP-bestand voor sequentiediagram van het spelvoorbeeld

```
include<LTC>

LTCvocabulary V {
    type Time isa nat
    Start: Time
    partial Next(Time) : Time

    type SDPoint = { 1..21 } isa nat

    SDPointAt(Time,SDPoint)
    I_SDPointAt(SDPoint)
    C_SDPointAt(Time,SDPoint)

    type LimitedInt isa int
    type LimitedFloat isa float
    type LimitedString isa string
    type boolean constructed from { T, F }
    type void constructed from { null }

    type Statistic
    type Item
    type Position
    type DerivedStatisticStrategy
    type DerivedStatistic isa Statistic
    type WeaponType
    type CharacterClass
    type Inventory
    type Weapon isa Item
    type Character
    type SupportAffinity
    type WeaponLevel isa Statistic

    DamageT(Time, LimitedInt)
    I_DamageT(LimitedInt)
    C_DamageT(Time, LimitedInt)

    StrengthT(Time, Statistic)
    I_StrengthT(Statistic)
```

```
39      C_StrengthT(Time, Statistic)
40
41      DefenceT(Time, Statistic)
42      I_DefenceT(Statistic)
43      C_DefenceT(Time, Statistic)
44
45      DefenceValT(Time, LimitedInt)
46      I_DefenceValT(LimitedInt)
47      C_DefenceValT(Time, LimitedInt)
48
49      HpT(Time, Statistic)
50      I_HpT(Statistic)
51      C_HpT(Time, Statistic)
52
53      AttackerT(Time, Character)
54      I_AttackerT(Character)
55      C_AttackerT(Time, Character)
56
57      TargetT(Time, Character)
58      I_TargetT(Character)
59      C_TargetT(Time, Character)
60
61      WeaponT(Time, Weapon)
62      I_WeaponT(Weapon)
63      C_WeaponT(Time, Weapon)
64
65      HpValT(Time, LimitedInt)
66      I_HpValT(LimitedInt)
67      C_HpValT(Time, LimitedInt)
68
69      NewHpT(Time, LimitedInt)
70      I_NewHpT(LimitedInt)
71      C_NewHpT(Time, LimitedInt)
72
73      StrengthValT(Time, LimitedInt)
74      I_StrengthValT(LimitedInt)
75      C_StrengthValT(Time, LimitedInt)
76
77      InflictedT(Time, LimitedInt)
78      I_InflictedT(LimitedInt)
79      C_InflictedT(Time, LimitedInt)
80
81      Positionyco(Time, Position, LimitedInt)
82      I_Positionyco(Position, LimitedInt)
83      C_Positionyco(Time, Position, LimitedInt)
84      Cn_Positionyco(Time, Position, LimitedInt)
85
86      Positionxco(Time, Position, LimitedInt)
87      I_Positionxco(Position, LimitedInt)
88      C_Positionxco(Time, Position, LimitedInt)
89      Cn_Positionxco(Time, Position, LimitedInt)
90
91      CharacterClassname(Time, CharacterClass, LimitedString)
92      I_CharacterClassname(CharacterClass, LimitedString)
93      C_CharacterClassname(Time, CharacterClass, LimitedString)
94      Cn_CharacterClassname(Time, CharacterClass, LimitedString)
95
96      Weapondamage(Time, Weapon, LimitedInt)
97      I_Weapondamage(Weapon, LimitedInt)
98      C_Weapondamage(Time, Weapon, LimitedInt)
99      Cn_Weapondamage(Time, Weapon, LimitedInt)
100
101     Weaponweight(Time, Weapon, LimitedInt)
```

```
    I_Weaponweight(Weapon, LimitedInt)
    C_Weaponweight(Time, Weapon, LimitedInt)
    Cn_Weaponweight(Time, Weapon, LimitedInt)

    Weaponrange(Time, Weapon, LimitedInt)
    I_Weaponrange(Weapon, LimitedInt)
    C_Weaponrange(Time, Weapon, LimitedInt)
    Cn_Weaponrange(Time, Weapon, LimitedInt)

    Characterdead(Time, Character, boolean)
    I_Characterdead(Character, boolean)
    C_Characterdead(Time, Character, boolean)
    Cn_Characterdead(Time, Character, boolean)

    Charactername(Time, Character, LimitedString)
    I_Charactername(Character, LimitedString)
    C_Charactername(Time, Character, LimitedString)
    Cn_Charactername(Time, Character, LimitedString)

    SupportAffinityname(Time, SupportAffinity, LimitedString)
    I_SupportAffinityname(SupportAffinity, LimitedString)
    C_SupportAffinityname(Time, SupportAffinity, LimitedString)
    Cn_SupportAffinityname(Time, SupportAffinity, LimitedString)

    Statisticvalue(Time, Statistic, LimitedInt)
    I_Statisticvalue(Statistic, LimitedInt)
    C_Statisticvalue(Time, Statistic, LimitedInt)
    Cn_Statisticvalue(Time, Statistic, LimitedInt)

    Statisticname(Time, Statistic, LimitedString)
    I_Statisticname(Statistic, LimitedString)
    C_Statisticname(Time, Statistic, LimitedString)
    Cn_Statisticname(Time, Statistic, LimitedString)

    Itemvalue(Time, Item, LimitedInt)
    I_Itemvalue(Item, LimitedInt)
    C_Itemvalue(Time, Item, LimitedInt)
    Cn_Itemvalue(Time, Item, LimitedInt)

    Itemname(Time, Item, LimitedString)
    I_Itemname(Item, LimitedString)
    C_Itemname(Time, Item, LimitedString)
    Cn_Itemname(Time, Item, LimitedString)

    CharacterandPosition(Character, Position)
    WeaponTypeandWeaponLevel(WeaponType, WeaponLevel)
    CharacterClassandCharacterClass(CharacterClass, CharacterClass)
    CharacterandSupportAffinity(Character, SupportAffinity)
    DerivedStatisticStrategyandDerivedStatistic(DerivedStatisticStrategy,
        DerivedStatistic)
    CharacterandCharacter(Character, Character)
    StatisticandCharacterClass(Statistic, CharacterClass)
    DerivedStatisticStrategyandStatistic(DerivedStatisticStrategy, Statistic)
    CharacterandStatistic(Character, Statistic)
    CharacterandCharacterClass(Character, CharacterClass)
    ItemandCharacterClass(Item, CharacterClass)
    StatisticandWeaponType(Statistic, WeaponType)
    CharacterandInventory(Character, Inventory)
    WeaponTypeandWeapon(WeaponType, Weapon)
    ItemandInventory(Item, Inventory)
}
theory T:V {
    {
```

```
164             ! t [Time] x [Weapon] v [LimitedInt] : C_Weaponrange(t, x, v) <-
                    false.
165             ! t [Time] x [Weapon] v [LimitedInt] : Cn_Weaponrange(t, x, v) <-
                    false.
166
167             ! t [Time] x [Position] v [LimitedInt] : C_Positionxco(t, x, v) <-
                     false.
168             ! t [Time] x [Position] v [LimitedInt] : Cn_Positionxco(t, x, v)
                    <- false.
169
170             ! t [Time] x [Position] v [LimitedInt] : C_Positionyco(t, x, v) <-
                     false.
171             ! t [Time] x [Position] v [LimitedInt] : Cn_Positionyco(t, x, v)
                    <- false.
172
173                     ! t [Time] x [Statistic] : C_Statisticvalue(t, x, 0) <-
                            SDPointAt(t, 18) & HpT(t, x).
174         ! t [Time] x [Statistic] v [LimitedInt] : C_Statisticvalue(t, x, v) <-
                SDPointAt(t, 20) & HpT(t, x) & NewHpT(t, v).
175
176                     ! t [Time] x [Statistic] v [LimitedInt] :
                            Cn_Statisticvalue(Next(t), x, v) <- SDPointAt(Next(t),
                             18) & HpT(t, x) & Statisticvalue(t, x, v) & ~(v = 0).
177         ! t [Time] x [Statistic] v [LimitedInt] : Cn_Statisticvalue(Next(t), x
                , v) <- SDPointAt(Next(t), 20) & HpT(t, x) & Statisticvalue(t, x,
                v) & ~NewHpT(Next(t), v).
178
179
180             ! t [Time] x [Weapon] v [LimitedInt] : C_Weaponweight(t, x, v) <-
                    false.
181             ! t [Time] x [Weapon] v [LimitedInt] : Cn_Weaponweight(t, x, v) <-
                     false.
182
183             ! t [Time] x [Item] v [LimitedString] : C_Itemname(t, x, v) <-
                    false.
184             ! t [Time] x [Item] v [LimitedString] : Cn_Itemname(t, x, v) <-
                    false.
185
186             ! t [Time] x [CharacterClass] v [LimitedString] :
                    C_CharacterClassname(t, x, v) <- false.
187             ! t [Time] x [CharacterClass] v [LimitedString] :
                    Cn_CharacterClassname(t, x, v) <- false.
188
189             ! t [Time] x [Character] v [LimitedString] : C_Charactername(t, x,
                     v) <- false.
190             ! t [Time] x [Character] v [LimitedString] : Cn_Charactername(t, x
                    , v) <- false.
191
192             ! t [Time] x [Item] v [LimitedInt] : C_Itemvalue(t, x, v) <- false
                    .
193             ! t [Time] x [Item] v [LimitedInt] : Cn_Itemvalue(t, x, v) <-
                    false.
194
195             ! t [Time] x [Weapon] v [LimitedInt] : C_Weapondamage(t, x, v) <-
                    false.
196             ! t [Time] x [Weapon] v [LimitedInt] : Cn_Weapondamage(t, x, v) <-
                     false.
197
198                     ! t [Time] x [Character] : C_Characterdead(t, x, T) <-
                            SDPointAt(t, 19) & TargetT(t, x).
199
200                     ! t [Time] x [Character] v [boolean] : Cn_Characterdead(
                            Next(t), x, v) <- SDPointAt(Next(t), 19) & TargetT(t,
```

```
                        x) & Characterdead(t, x, v) & ~(v = T).
201
202
203             ! t [Time] x [Statistic] v [LimitedString] : C_Statisticname(t, x,
                     v) <- false.
204             ! t [Time] x [Statistic] v [LimitedString] : Cn_Statisticname(t, x
                    , v) <- false.
205
206             ! t [Time] x [SupportAffinity] v [LimitedString] :
                    C_SupportAffinityname(t, x, v) <- false.
207             ! t [Time] x [SupportAffinity] v [LimitedString] :
                    Cn_SupportAffinityname(t, x, v) <- false.
208         ! t [Time] s [SDPoint] : C_SDPointAt(Next(t), (s+1)) <- SDPointAt(t, s
                )& ~((s = 17) | (s = 19) | (s = 21)).
209         ! t [Time] : C_SDPointAt(Next(t), 18) <- SDPointAt(t, 17) & ( ? newHp
                [LimitedInt] : NewHpT(t, newHp) & newHp =< 0).
210         ! t [Time] : C_SDPointAt(Next(t), 20) <- SDPointAt(t, 17) & ( ? newHp
                [LimitedInt] : NewHpT(t, newHp) & newHp > 0).
211         ! t [Time] : C_SDPointAt(Next(t), 21) <- SDPointAt(t, 19) | SDPointAt(
                t, 20).
212         ! t [Time] : C_SDPointAt(Next(t), 1) <- SDPointAt(t, 21).
213
214          ! t [Time] defence [Statistic] : C_DefenceT(t, defence) <- SDPointAt(
                 t, 3) & (? target [Character] : TargetT(t, target) &
                 CharacterandStatistic(target, defence) & Statisticname(t, defence
                 , "defence")).
215          ! t [Time] strength [Statistic] : C_StrengthT(Next(t), strength) <-
                 SDPointAt(t, 4) & (? attacker [Character] : AttackerT(t, attacker
                 ) & CharacterandStatistic(attacker, strength) & Statisticname(t,
                 strength, "strength")).
216         ! t [Time] x [LimitedInt] : C_DefenceValT(Next(t), x) <- SDPointAt(t,
                6) & (? o [Statistic] : DefenceT(t, o) & Statisticvalue(t, o, x)).
217         ! t [Time] x [LimitedInt] : C_StrengthValT(Next(t), x) <- SDPointAt(t,
                 8) & (? o [Statistic] : StrengthT(t, o) & Statisticvalue(t, o, x)
                ).
218         ! t [Time] x [LimitedInt] : C_DamageT(Next(t), x) <- SDPointAt(t, 10)
                & (? o [Weapon] : WeaponT(t, o) & Weapondamage(t, o, x)).
219         ! t [Time] inflicted [LimitedInt] : C_InflictedT(t, inflicted) <-
                SDPointAt(t, 12) & (? damage [LimitedInt] strengthVal [LimitedInt]
                 defenceVal [LimitedInt] : DamageT(t, damage) & StrengthValT(t,
                strengthVal) & DefenceValT(t, defenceVal) & inflicted=damage+
                strengthVal-defenceVal).
220          ! t [Time] hp [Statistic] : C_HpT(t, hp) <- SDPointAt(t, 14) & (?
                 target [Character] : TargetT(t, target) & CharacterandStatistic(
                 target, hp) & Statisticname(t, hp, "hp")).
221         ! t [Time] x [LimitedInt] : C_HpValT(Next(t), x) <- SDPointAt(t, 15) &
                 (? o [Statistic] : HpT(t, o) & Statisticvalue(t, o, x)).
222         ! t [Time] newHp [LimitedInt] : C_NewHpT(t, newHp) <- SDPointAt(t, 17)
                 & (? hpVal [LimitedInt] inflicted [LimitedInt] : HpValT(t, hpVal)
                 & InflictedT(t, inflicted) & newHp=hpVal-inflicted).
223     }
224     {
225             ! s [SDPoint] : SDPointAt(Start, s) <- I_SDPointAt(s).
226             ! t [Time] s [SDPoint] : SDPointAt(Next(t), s) <- C_SDPointAt(Next
                    (t), s).
227             ! t [Time] s [SDPoint] : SDPointAt(Next(t), s) <- SDPointAt(t, s)
                    & ~(? s1 [SDPoint] : C_SDPointAt(Next(t), s1)).
228
229             ! x [LimitedInt] : DamageT(Start, x) <- I_DamageT(x).
230             ! t [Time] x [LimitedInt] : DamageT(t, x) <- C_DamageT(t, x).
231             ! t [Time] x [LimitedInt] : DamageT(Next(t), x) <- DamageT(t, x) &
                     ~( ? x1 [LimitedInt] : C_DamageT(Next(t), x1) & ~(x = x1)).
232
```

```
233             ! x [Statistic] : StrengthT(Start, x) <- I_StrengthT(x).
234             ! t [Time] x [Statistic] : StrengthT(t, x) <- C_StrengthT(t, x).
235             ! t [Time] x [Statistic] : StrengthT(Next(t), x) <- StrengthT(t, x
                    ) & ~( ? x1 [Statistic] : C_StrengthT(Next(t), x1) & ~(x = x1)
                    ).
236
237             ! x [Statistic] : DefenceT(Start, x) <- I_DefenceT(x).
238             ! t [Time] x [Statistic] : DefenceT(t, x) <- C_DefenceT(t, x).
239             ! t [Time] x [Statistic] : DefenceT(Next(t), x) <- DefenceT(t, x)
                    & ~( ? x1 [Statistic] : C_DefenceT(Next(t), x1) & ~(x = x1)).
240
241             ! x [LimitedInt] : DefenceValT(Start, x) <- I_DefenceValT(x).
242             ! t [Time] x [LimitedInt] : DefenceValT(t, x) <- C_DefenceValT(t,
                    x).
243             ! t [Time] x [LimitedInt] : DefenceValT(Next(t), x) <- DefenceValT
                    (t, x) & ~( ? x1 [LimitedInt] : C_DefenceValT(Next(t), x1) &
                    ~(x = x1)).
244
245             ! x [Statistic] : HpT(Start, x) <- I_HpT(x).
246             ! t [Time] x [Statistic] : HpT(t, x) <- C_HpT(t, x).
247             ! t [Time] x [Statistic] : HpT(Next(t), x) <- HpT(t, x) & ~( ? x1
                    [Statistic] : C_HpT(Next(t), x1) & ~(x = x1)).
248
249             ! x [Character] : AttackerT(Start, x) <- I_AttackerT(x).
250             ! t [Time] x [Character] : AttackerT(t, x) <- C_AttackerT(t, x).
251             ! t [Time] x [Character] : AttackerT(Next(t), x) <- AttackerT(t, x
                    ) & ~( ? x1 [Character] : C_AttackerT(Next(t), x1) & ~(x = x1)
                    ).
252
253             ! x [Character] : TargetT(Start, x) <- I_TargetT(x).
254             ! t [Time] x [Character] : TargetT(t, x) <- C_TargetT(t, x).
255             ! t [Time] x [Character] : TargetT(Next(t), x) <- TargetT(t, x) &
                    ~( ? x1 [Character] : C_TargetT(Next(t), x1) & ~(x = x1)).
256
257             ! x [Weapon] : WeaponT(Start, x) <- I_WeaponT(x).
258             ! t [Time] x [Weapon] : WeaponT(t, x) <- C_WeaponT(t, x).
259             ! t [Time] x [Weapon] : WeaponT(Next(t), x) <- WeaponT(t, x) & ~(
                    ? x1 [Weapon] : C_WeaponT(Next(t), x1) & ~(x = x1)).
260
261             ! x [LimitedInt] : HpValT(Start, x) <- I_HpValT(x).
262             ! t [Time] x [LimitedInt] : HpValT(t, x) <- C_HpValT(t, x).
263             ! t [Time] x [LimitedInt] : HpValT(Next(t), x) <- HpValT(t, x) &
                    ~( ? x1 [LimitedInt] : C_HpValT(Next(t), x1) & ~(x = x1)).
264
265             ! x [LimitedInt] : NewHpT(Start, x) <- I_NewHpT(x).
266             ! t [Time] x [LimitedInt] : NewHpT(t, x) <- C_NewHpT(t, x).
267             ! t [Time] x [LimitedInt] : NewHpT(Next(t), x) <- NewHpT(t, x) &
                    ~( ? x1 [LimitedInt] : C_NewHpT(Next(t), x1) & ~(x = x1)).
268
269             ! x [LimitedInt] : StrengthValT(Start, x) <- I_StrengthValT(x).
270             ! t [Time] x [LimitedInt] : StrengthValT(t, x) <- C_StrengthValT(t
                    , x).
271             ! t [Time] x [LimitedInt] : StrengthValT(Next(t), x) <-
                    StrengthValT(t, x) & ~( ? x1 [LimitedInt] : C_StrengthValT(
                    Next(t), x1) & ~(x = x1)).
272
273             ! x [LimitedInt] : InflictedT(Start, x) <- I_InflictedT(x).
274             ! t [Time] x [LimitedInt] : InflictedT(t, x) <- C_InflictedT(t, x)
                    .
275             ! t [Time] x [LimitedInt] : InflictedT(Next(t), x) <- InflictedT(t
                    , x) & ~( ? x1 [LimitedInt] : C_InflictedT(Next(t), x1) & ~(x
                    = x1)).
276
```

```
277             ! x [Position] y [LimitedInt] : Positionyco(Start, x, y) <-
                    I_Positionyco(x, y).
278             ! t [Time] x [Position] y [LimitedInt] : Positionyco(t, x, y) <-
                    C_Positionyco(t, x, y).
279             ! t [Time] x [Position] y [LimitedInt] : Positionyco(Next(t), x, y
                    ) <- Positionyco(t, x, y) & ~Cn_Positionyco(Next(t), x, y).
280
281             ! x [Position] y [LimitedInt] : Positionxco(Start, x, y) <-
                    I_Positionxco(x, y).
282             ! t [Time] x [Position] y [LimitedInt] : Positionxco(t, x, y) <-
                    C_Positionxco(t, x, y).
283             ! t [Time] x [Position] y [LimitedInt] : Positionxco(Next(t), x, y
                    ) <- Positionxco(t, x, y) & ~Cn_Positionxco(Next(t), x, y).
284
285             ! x [CharacterClass] y [LimitedString] : CharacterClassname(Start,
                     x, y) <- I_CharacterClassname(x, y).
286             ! t [Time] x [CharacterClass] y [LimitedString] :
                    CharacterClassname(t, x, y) <- C_CharacterClassname(t, x, y).
287             ! t [Time] x [CharacterClass] y [LimitedString] :
                    CharacterClassname(Next(t), x, y) <- CharacterClassname(t, x,
                    y) & ~Cn_CharacterClassname(Next(t), x, y).
288
289             ! x [Weapon] y [LimitedInt] : Weapondamage(Start, x, y) <-
                    I_Weapondamage(x, y).
290             ! t [Time] x [Weapon] y [LimitedInt] : Weapondamage(t, x, y) <-
                    C_Weapondamage(t, x, y).
291             ! t [Time] x [Weapon] y [LimitedInt] : Weapondamage(Next(t), x, y)
                     <- Weapondamage(t, x, y) & ~Cn_Weapondamage(Next(t), x, y).
292
293             ! x [Weapon] y [LimitedInt] : Weaponweight(Start, x, y) <-
                    I_Weaponweight(x, y).
294             ! t [Time] x [Weapon] y [LimitedInt] : Weaponweight(t, x, y) <-
                    C_Weaponweight(t, x, y).
295             ! t [Time] x [Weapon] y [LimitedInt] : Weaponweight(Next(t), x, y)
                     <- Weaponweight(t, x, y) & ~Cn_Weaponweight(Next(t), x, y).
296
297             ! x [Weapon] y [LimitedInt] : Weaponrange(Start, x, y) <-
                    I_Weaponrange(x, y).
298             ! t [Time] x [Weapon] y [LimitedInt] : Weaponrange(t, x, y) <-
                    C_Weaponrange(t, x, y).
299             ! t [Time] x [Weapon] y [LimitedInt] : Weaponrange(Next(t), x, y)
                    <- Weaponrange(t, x, y) & ~Cn_Weaponrange(Next(t), x, y).
300
301             ! x [Character] y [boolean] : Characterdead(Start, x, y) <-
                    I_Characterdead(x, y).
302             ! t [Time] x [Character] y [boolean] : Characterdead(t, x, y) <-
                    C_Characterdead(t, x, y).
303             ! t [Time] x [Character] y [boolean] : Characterdead(Next(t), x, y
                    ) <- Characterdead(t, x, y) & ~Cn_Characterdead(Next(t), x, y)
                    .
304
305             ! x [Character] y [LimitedString] : Charactername(Start, x, y) <-
                    I_Charactername(x, y).
306             ! t [Time] x [Character] y [LimitedString] : Charactername(t, x, y
                    ) <- C_Charactername(t, x, y).
307             ! t [Time] x [Character] y [LimitedString] : Charactername(Next(t)
                    , x, y) <- Charactername(t, x, y) & ~Cn_Charactername(Next(t),
                     x, y).
308
309             ! x [SupportAffinity] y [LimitedString] : SupportAffinityname(
                    Start, x, y) <- I_SupportAffinityname(x, y).
310             ! t [Time] x [SupportAffinity] y [LimitedString] :
                    SupportAffinityname(t, x, y) <- C_SupportAffinityname(t, x, y)
```

```
            .
311         ! t [Time] x [SupportAffinity] y [LimitedString] :
                SupportAffinityname(Next(t), x, y) <- SupportAffinityname(t, x
                , y) & ~Cn_SupportAffinityname(Next(t), x, y).
312
313         ! x [Statistic] y [LimitedInt] : Statisticvalue(Start, x, y) <-
                I_Statisticvalue(x, y).
314         ! t [Time] x [Statistic] y [LimitedInt] : Statisticvalue(t, x, y)
                <- C_Statisticvalue(t, x, y).
315         ! t [Time] x [Statistic] y [LimitedInt] : Statisticvalue(Next(t),
                x, y) <- Statisticvalue(t, x, y) & ~Cn_Statisticvalue(Next(t),
                 x, y).
316
317         ! x [Statistic] y [LimitedString] : Statisticname(Start, x, y) <-
                I_Statisticname(x, y).
318         ! t [Time] x [Statistic] y [LimitedString] : Statisticname(t, x, y
                ) <- C_Statisticname(t, x, y).
319         ! t [Time] x [Statistic] y [LimitedString] : Statisticname(Next(t)
                , x, y) <- Statisticname(t, x, y) & ~Cn_Statisticname(Next(t),
                 x, y).
320
321         ! x [Item] y [LimitedInt] : Itemvalue(Start, x, y) <- I_Itemvalue(
                x, y).
322         ! t [Time] x [Item] y [LimitedInt] : Itemvalue(t, x, y) <-
                C_Itemvalue(t, x, y).
323         ! t [Time] x [Item] y [LimitedInt] : Itemvalue(Next(t), x, y) <-
                Itemvalue(t, x, y) & ~Cn_Itemvalue(Next(t), x, y).
324
325         ! x [Item] y [LimitedString] : Itemname(Start, x, y) <- I_Itemname
                (x, y).
326         ! t [Time] x [Item] y [LimitedString] : Itemname(t, x, y) <-
                C_Itemname(t, x, y).
327         ! t [Time] x [Item] y [LimitedString] : Itemname(Next(t), x, y) <-
                 Itemname(t, x, y) & ~Cn_Itemname(Next(t), x, y).
328
329
330     }
331         ! x [Position] : ?1 v [LimitedInt] : I_Positionyco(x, v).
332         ! t [Time] x [Position] : ?1 v [LimitedInt] : Positionyco(t, x, v).
333
334         ! x [Position] : ?1 v [LimitedInt] : I_Positionxco(x, v).
335         ! t [Time] x [Position] : ?1 v [LimitedInt] : Positionxco(t, x, v).
336
337         ! x [CharacterClass] : ?1 v [LimitedString] : I_CharacterClassname(x,
                v).
338         ! t [Time] x [CharacterClass] : ?1 v [LimitedString] :
                CharacterClassname(t, x, v).
339
340         ! x [Weapon] : ?1 v [LimitedInt] : I_Weapondamage(x, v).
341         ! t [Time] x [Weapon] : ?1 v [LimitedInt] : Weapondamage(t, x, v).
342
343         ! x [Weapon] : ?1 v [LimitedInt] : I_Weaponweight(x, v).
344         ! t [Time] x [Weapon] : ?1 v [LimitedInt] : Weaponweight(t, x, v).
345
346         ! x [Weapon] : ?1 v [LimitedInt] : I_Weaponrange(x, v).
347         ! t [Time] x [Weapon] : ?1 v [LimitedInt] : Weaponrange(t, x, v).
348
349         ! x [Character] : ?1 v [boolean] : I_Characterdead(x, v).
350         ! t [Time] x [Character] : ?1 v [boolean] : Characterdead(t, x, v).
351
352         ! x [Character] : ?1 v [LimitedString] : I_Charactername(x, v).
353         ! t [Time] x [Character] : ?1 v [LimitedString] : Charactername(t, x,
                v).
```

```
354
355         ! x [SupportAffinity] : ?1 v [LimitedString] : I_SupportAffinityname(x
                , v).
356         ! t [Time] x [SupportAffinity] : ?1 v [LimitedString] :
                SupportAffinityname(t, x, v).
357
358         ! x [Statistic] : ?1 v [LimitedInt] : I_Statisticvalue(x, v).
359         ! t [Time] x [Statistic] : ?1 v [LimitedInt] : Statisticvalue(t, x, v)
                .
360
361         ! x [Statistic] : ?1 v [LimitedString] : I_Statisticname(x, v).
362         ! t [Time] x [Statistic] : ?1 v [LimitedString] : Statisticname(t, x,
                v).
363
364         ! x [Item] : ?1 v [LimitedInt] : I_Itemvalue(x, v).
365         ! t [Time] x [Item] : ?1 v [LimitedInt] : Itemvalue(t, x, v).
366
367         ! x [Item] : ?1 v [LimitedString] : I_Itemname(x, v).
368         ! t [Time] x [Item] : ?1 v [LimitedString] : Itemname(t, x, v).
369
370         ! x2 [Position] : #{ x1 [Character] : CharacterandPosition(x1, x2)} =<
                1.
371         ! x1 [Character] : #{ x2 [Position] : CharacterandPosition(x1, x2)} =<
                1.
372
373         ! x2 [WeaponLevel] : ?1 x1 [WeaponType] : WeaponTypeandWeaponLevel(x1,
                x2).
374
375
376         ! x2 [SupportAffinity] : ?1 x1 [Character] :
                CharacterandSupportAffinity(x1, x2).
377         ! x1 [Character] : ?1 x2 [SupportAffinity] :
                CharacterandSupportAffinity(x1, x2).
378
379         ! x2 [DerivedStatistic] : ?1 x1 [DerivedStatisticStrategy] :
                DerivedStatisticStrategyandDerivedStatistic(x1, x2).
380         ! x1 [DerivedStatisticStrategy] : ?1 x2 [DerivedStatistic] :
                DerivedStatisticStrategyandDerivedStatistic(x1, x2).
381
382
383         ! x2 [CharacterClass] : #{ x1 [Statistic] : StatisticandCharacterClass
                (x1, x2)} >= 1.
384         ! x1 [Statistic] : ?1 x2 [CharacterClass] : StatisticandCharacterClass
                (x1, x2).
385
386         ! x1 [DerivedStatisticStrategy] : #{ x2 [Statistic] :
                DerivedStatisticStrategyandStatistic(x1, x2)} >= 1.
387
388         ! x2 [Statistic] : ?1 x1 [Character] : CharacterandStatistic(x1, x2).
389         ! x1 [Character] : #{ x2 [Statistic] : CharacterandStatistic(x1, x2)}
                >= 1.
390
391         ! x1 [Character] : ?1 x2 [CharacterClass] : CharacterandCharacterClass
                (x1, x2).
392
393
394         ! x2 [WeaponType] : #{ x1 [Statistic] : StatisticandWeaponType(x1, x2)
                } =< 1.
395
396         ! x2 [Inventory] : ?1 x1 [Character] : CharacterandInventory(x1, x2).
397         ! x1 [Character] : ?1 x2 [Inventory] : CharacterandInventory(x1, x2).
398
```

```
399         ! x2 [Weapon] : #{ x1 [WeaponType] : WeaponTypeandWeapon(x1, x2)} =<
                1.
400
401         ! x2 [Inventory] : #{ x1 [Item] : ItemandInventory(x1, x2)} =< 5.
402         ! x1 [Item] : #{ x2 [Inventory] : ItemandInventory(x1, x2)} =< 1.
403
404 }
405 structure S:V {
406     Time = { 0..40 }
407     Start = 0
408     Next = { 0->1; 1->2; 2->3; 3->4; 4->5; 5->6; 6->7; 7->8; 8->9; 9->10;
            10->11; 11->12; 12->13; 13->14; 14->15; 15->16; 16->17; 17->18;
            18->19; 19->20; 20->21; 21->22; 22->23; 23->24; 24->25; 25->26;
            26->27; 27->28; 28->29; 29->30; 30->31; 31->32; 32->33; 33->34;
            34->35; 35->36; 36->37; 37->38; 38->39; 39->40 }
409
410     I_SDPointAt = { 1 }
411
412     LimitedInt = { -3..3 }
413     LimitedFloat = { 0.0; 0.5; -0.5; 0.5; -0.5}
414     LimitedString = { "FcPBe4HTw3ZpeLBKRbR6"; "UNsSbSrxxg21BWTZuV41"; "
            iEcKqyJxivjFU0w1E6uH"; "bruGumhm1weHaeDf5zVh"; "hE4a0GaH2xRHShwIASw7";
             "damage"; "strength"; "defence"; "defenceVal"; "hp"; "attacker"; "
            target"; "weapon"; "hpVal"; "newHp"; "strengthVal"; "inflicted"}
415
416     Position = { Position1; Position2; Position3; Position4; Position5}
417     DerivedStatisticStrategy = { DerivedStatisticStrategy1;
            DerivedStatisticStrategy2; DerivedStatisticStrategy3;
            DerivedStatisticStrategy4; DerivedStatisticStrategy5}
418     DerivedStatistic = { DerivedStatistic1; DerivedStatistic2;
            DerivedStatistic3; DerivedStatistic4; DerivedStatistic5}
419     WeaponType = { WeaponType1; WeaponType2; WeaponType3; WeaponType4;
            WeaponType5}
420     CharacterClass = { CharacterClass1; CharacterClass2; CharacterClass3;
            CharacterClass4; CharacterClass5}
421     Inventory = { Inventory1; Inventory2; Inventory3; Inventory4; Inventory5}
422     Weapon = { Weapon1; Weapon2; Weapon3; Weapon4; Weapon5}
423     Character = { Character1; Character2; Character3; Character4; Character5}
424     SupportAffinity = { SupportAffinity1; SupportAffinity2; SupportAffinity3;
            SupportAffinity4; SupportAffinity5}
425     WeaponLevel = { WeaponLevel1; WeaponLevel2; WeaponLevel3; WeaponLevel4;
            WeaponLevel5}
426     Statistic = { Statistic1; Statistic2; Statistic3; Statistic4; Statistic5;
            WeaponLevel1; WeaponLevel2; WeaponLevel3; WeaponLevel4; WeaponLevel5;
            DerivedStatistic1; DerivedStatistic2; DerivedStatistic3;
            DerivedStatistic4; DerivedStatistic5}
427     Item = { Item1; Item2; Item3; Item4; Item5; Weapon1; Weapon2; Weapon3;
            Weapon4; Weapon5}
428
429     I_Statisticname<ct> = {Statistic1,"strength";Statistic2,"defence";
            Statistic3,"hp"}
430     I_Statisticvalue<ct> = {Statistic1,2;Statistic2,1;Statistic3,3}
431     I_Weapondamage = {Weapon1,1;Weapon2,1;Weapon3,1;Weapon4,1;Weapon5,1}
432
433     I_TargetT = { Character2 }
434     I_AttackerT = { Character1 }
435     I_WeaponT = { Weapon1 }
436     I_DefenceT = {}
437     I_StrengthT = {}
438     I_DefenceValT = {}
439     I_StrengthValT = {}
440     I_DamageT = {}
441     I_InflictedT = {}
```

```
442     I_HpT = {}
443     I_HpValT = {}
444     I_NewHpT = {}
445
446     C_WeaponT = {}
447     C_TargetT = {}
448     C_AttackerT = {}
449
450     CharacterandStatistic<ct> = { Character1,Statistic1; Character2,Statistic2
            ; Character2,Statistic3 }
451     I_Characterdead<ct> = { Character2,F() }
452 }
453 procedure main() {
454     print(modelexpand(T,S)[1])
455 }
```

Modellering van het gedrag van het sequentiediagram in figuur ??

Bijlage D

IDP-bestand voor sequentiediagram voor het voorbeeld over recursie

```
vocabulary V {

    type Time isa nat
    Start: Time
    partial Next(Time) : Time

    type SDPoint constructed from { methodOne_1, methodOne_2, methodOne_3,
        methodOne_3post, methodOne_4, methodOne_5, methodTwo_1, methodTwo_2,
        methodTwo_3, methodTwo_3post, methodTwo_4, methodTwo_5, finished }
    partial NextSD(SDPoint) : SDPoint

    SDPointAt(Time,SDPoint)
    I_SDPointAt(SDPoint)
    C_SDPoint(Time,SDPoint)

    type LimitedInt = { 0..5 } isa int
    type StackLevel = { 1..10 } isa nat
    type bool constructed from { T,F }

    type A

    ObjT(Time,StackLevel,A)
    I_ObjT(StackLevel, A)
    C_ObjT(Time, StackLevel, A)

    Obj2T(Time,StackLevel,A)
    I_Obj2T(StackLevel, A)
    C_Obj2T(Time, StackLevel, A)

    FinishedT(Time,StackLevel,bool)
    I_FinishedT(StackLevel, bool)
    C_FinishedT(Time, StackLevel, bool)

    MTwoArgT(Time,StackLevel,LimitedInt)
    I_MTwoArgT(StackLevel, LimitedInt)
    C_MTwoArgT(Time, StackLevel, LimitedInt)

    CurrentStackLevel(Time) : StackLevel
```

```
37      I_CurrentStackLevel : StackLevel
38      C_CurrentStackLevel(Time, StackLevel)
39
40      ReturnPoint(Time,StackLevel,SDPoint)
41      I_ReturnPoint(StackLevel, SDPoint)
42      C_ReturnPoint(Time, StackLevel, SDPoint)
43      Cn_ReturnPoint(Time, StackLevel, SDPoint)
44  }
45
46  theory T:V {
47
48      {
49      ! t [Time] s [SDPoint] : C_SDPoint(Next(t), NextSD(s)) <- SDPointAt(t, s)
            & ~((s = methodOne_3) | (s = methodOne_5) | (s = methodTwo_1) | (s =
            methodTwo_3) | (s = methodTwo_3post) | (s = methodTwo_4) | (s =
            methodTwo_5) | (s = finished)).
50      ! t [Time] : C_SDPoint(Next(t), methodTwo_2) <- SDPointAt(t, methodTwo_1)
            & (? n [LimitedInt] : MTwoArgT(t, CurrentStackLevel(t), n) & n < 3).
51      ! t [Time] : C_SDPoint(Next(t), methodTwo_4) <- SDPointAt(t, methodTwo_1)
            & (? n [LimitedInt] : MTwoArgT(t, CurrentStackLevel(t), n) & n >= 3).
52      ! t [Time] : C_SDPoint(Next(t), methodTwo_5) <- SDPointAt(t,
            methodTwo_3post) | SDPointAt(t, methodTwo_4).
53      ! t [Time] : C_SDPoint(Next(t), methodTwo_1) <- SDPointAt(t, methodOne_3).
54      ! t [Time] : C_SDPoint(Next(t), methodTwo_1) <- SDPointAt(t, methodTwo_3).
55
56      ! t [Time] s [SDPoint] : C_SDPoint(Next(t), s) <- ReturnPoint(t,
            CurrentStackLevel(t), s) & (SDPointAt(t, methodOne_5) | SDPointAt(t,
            methodTwo_5)).
57
58      ! t [Time] st [StackLevel] : C_ReturnPoint(Next(t), st, methodOne_3post)
            <- (CurrentStackLevel(t) = (st-1)) & SDPointAt(t, methodOne_3).
59      ! t [Time] st [StackLevel] : C_ReturnPoint(Next(t), st, methodTwo_3post)
            <- (CurrentStackLevel(t) = (st-1)) & SDPointAt(t, methodTwo_3).
60      ! t [Time] st [StackLevel] sd [SDPoint] : Cn_ReturnPoint(Next(t), st, sd)
            <- (CurrentStackLevel(t) = st) & ReturnPoint(t, st, sd) & (SDPointAt(t
            , methodOne_5) | SDPointAt(t, methodTwo_5)).
61
62      ! t [Time] st [StackLevel] : C_CurrentStackLevel(Next(t), st) <- (
            CurrentStackLevel(t) = (st - 1)) & (SDPointAt(t, methodOne_3) |
            SDPointAt(t, methodTwo_3)).
63      ! t [Time] st [StackLevel] : C_CurrentStackLevel(Next(t), st) <- (
            CurrentStackLevel(t) = (st + 1)) & (SDPointAt(t, methodOne_5) |
            SDPointAt(t, methodTwo_5)).
64
65
66      ! t [Time] s [StackLevel] : C_FinishedT(t, s, F) <- (s = CurrentStackLevel
            (t)) & SDPointAt(t, methodOne_2).
67      ! t [Time] s [StackLevel] : C_FinishedT(t, s, T) <- (s = CurrentStackLevel
            (t)) & SDPointAt(t, methodOne_4).
68
69      ! t [Time] s [StackLevel] obj [A] : C_Obj2T(Next(t), s, obj) <- (
            CurrentStackLevel(t) = (s-1)) & SDPointAt(t, methodOne_3) & ObjT(t, (s
            -1), obj).
70      ! t [Time] s [StackLevel] obj [A] : C_Obj2T(Next(t), s, obj) <- (
            CurrentStackLevel(t) = (s-1)) & SDPointAt(t, methodTwo_3) & Obj2T(t, (
            s-1), obj).
71
72      ! t [Time] s [StackLevel] : C_MTwoArgT(Next(t), s, 1) <- (
            CurrentStackLevel(t) = (s-1)) & SDPointAt(t, methodOne_3).
73      ! t [Time] s [StackLevel] n [LimitedInt] : C_MTwoArgT(Next(t), s, n) <- (
            CurrentStackLevel(t) = (s-1)) & SDPointAt(t, methodTwo_3) & MTwoArgT(t
            , (s-1), n).
```

```
74      ! t [Time] s [StackLevel] n [LimitedInt] : C_MTwoArgT(Next(t), s, n) <- (
            CurrentStackLevel(t) = s) & SDPointAt(t, methodTwo_2) & (? n1 [
            LimitedInt] : MTwoArgT(t, s, n1) & (n = n1 + 1)).
75      }
76
77      {
78      ! s [SDPoint] : SDPointAt(Start, s) <- I_SDPointAt(s).
79      ! t [Time] s [SDPoint] : SDPointAt(Next(t), s) <- C_SDPoint(Next(t), s).
80      ! t [Time] s [SDPoint] : SDPointAt(Next(t), s) <- SDPointAt(t, s) & ~(? s1
             [SDPoint] : C_SDPoint(Next(t), s1)).
81
82      ! st [StackLevel] : CurrentStackLevel(Start) = st <- I_CurrentStackLevel =
             st.
83      ! t [Time] st [StackLevel] : CurrentStackLevel(t) = st <-
            C_CurrentStackLevel(t, st).
84      ! t [Time] st [StackLevel] : CurrentStackLevel(Next(t)) = st <-
            CurrentStackLevel(t) = st & ~(? st1 [StackLevel] : C_CurrentStackLevel
            (Next(t), st1)).
85
86      ! st [StackLevel] sd [SDPoint] : ReturnPoint(Start, st, sd) <-
            I_ReturnPoint(st, sd).
87      ! t [Time] st [StackLevel] sd [SDPoint] : ReturnPoint(t, st, sd) <-
            C_ReturnPoint(t, st, sd).
88      ! t [Time] st [StackLevel] sd [SDPoint] : ReturnPoint(Next(t), st, sd) <-
            ReturnPoint(t, st, sd) & ~Cn_ReturnPoint(Next(t), st, sd).
89
90      ! st [StackLevel] obj [A] : ObjT(Start, st, obj) <- I_ObjT(st, obj).
91      ! t [Time] st [StackLevel] obj [A] : ObjT(t, st, obj) <- C_ObjT(t, st, obj
            ).
92      ! t [Time] st [StackLevel] obj [A] : ObjT(Next(t), st, obj) <- ObjT(t, st,
             obj) & ~(? obj1 [A] : C_ObjT(Next(t), st, obj1) & ~(obj = obj1)).
93
94      ! st [StackLevel] obj [A] : Obj2T(Start, st, obj) <- I_Obj2T(st, obj).
95      ! t [Time] st [StackLevel] obj [A] : Obj2T(t, st, obj) <- C_Obj2T(t, st,
            obj).
96      ! t [Time] st [StackLevel] obj [A] : Obj2T(Next(t), st, obj) <- Obj2T(t,
            st, obj) & ~(? obj1 [A] : C_Obj2T(Next(t), st, obj1) & ~(obj = obj1)).
97
98      ! st [StackLevel] b [bool] : FinishedT(Start, st, b) <- I_FinishedT(st, b)
            .
99      ! t [Time] st [StackLevel] b [bool] : FinishedT(t, st, b) <- C_FinishedT(t
            , st, b).
100     ! t [Time] st [StackLevel] b [bool] : FinishedT(Next(t), st, b) <-
            FinishedT(t, st, b) & ~(? b1 [bool] : C_FinishedT(Next(t), st, b1) &
            ~(b = b1)).
101
102     ! st [StackLevel] n [LimitedInt] : MTwoArgT(Start, st, n) <- I_MTwoArgT(st
            , n).
103     ! t [Time] st [StackLevel] n [LimitedInt] : MTwoArgT(t, st, n) <-
            C_MTwoArgT(t, st, n).
104     ! t [Time] st [StackLevel] n [LimitedInt] : MTwoArgT(Next(t), st, n) <-
            MTwoArgT(t, st, n) & ~(? n1 [LimitedInt] : C_MTwoArgT(Next(t), st, n1)
             & ~(n = n1)).
105     }
106 }
107
108 structure S:V {
109     Time = { 0..20 }
110     Start = 0
111 //    Next = { 0->1;1->2;2->3;3->4 }
112     Next = {
            0->1;1->2;2->3;3->4;4->5;5->6;6->7;7->8;8->9;9->10;10->11;11->12;12->13;13->14;14->15;15->16;1
             }
```

```
113     NextSD = { methodOne_1->methodOne_2; methodOne_2->methodOne_3; methodOne_3
            ->methodOne_3post; methodOne_3post->methodOne_4; methodOne_4->
            methodOne_5; methodTwo_1->methodTwo_2; methodTwo_2->methodTwo_3;
            methodTwo_3->methodTwo_3post; methodTwo_3post->methodTwo_4;
            methodTwo_4->methodTwo_5 }
114     A = { objA }
115     I_SDPointAt = {methodOne_1}
116     I_CurrentStackLevel = 1
117     I_ReturnPoint = {1, finished}
118     I_ObjT = {1, objA}
119     I_Obj2T = {}
120     I_FinishedT = {}
121     I_MTwoArgT = {}
122     C_ObjT = {}
123 }
124
125 procedure main() {
126     print(modelexpand(T,S)[1])
127 }
```

Modellering van het gedrag van de sequentiediagrammen in figuur 5.3

Bibliografie

[1] Bart Bogaerts. Simulating dynamic systems using linear time calculus theories. *Theory and Practice of Logic Programming*, 14(4-5):477–492, 2014.

Fiche masterproef

Student: Thomas Vochten

Titel: Automatische verificatie en kwaliteitscontrole van UML-diagrammen met FO(.)

Engelse titel: TBD

UDC: TBD

Korte inhoud:
Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdoptie Gedistribueerde systemen

Promotor: Prof. dr. Marc Denecker

Assessor: TBD

Begeleider: Matthias van der Hallen