

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

© Copyright KU Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotor als de auteur is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wend u tot het Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 of via e-mail info@cs.kuleuven.be.

Voorafgaande schriftelijke toestemming van de promotor is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Voorwoord

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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

Voorwoord	i
Samenvatting	iii
1 Inleiding	1
2 Controleren van consistentie	3
3 Controleren op kwaliteitsgebreken	11
4 De rol van IDP	13
5 Simuleren van gedrag op basis van een sequentiediagram	27
A IDP-bestand voor sequentiediagram van het spelvoorbeeld	39
Bibliografie	51

Samenvatting

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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

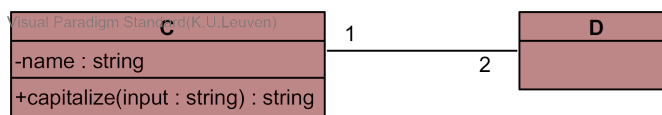
Inleiding

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 *klassen* genoemd) er bestaan binnen de software. Elk van die klassen kan attributen en operaties hebben. Verder geeft een klassediagram ook weer welke klassen 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 klassen 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 klassen. 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 onduidelikheden in het ontwerp introduceren of het onderhoud van de software éénmaal ingezet in productie bemoeilijken.

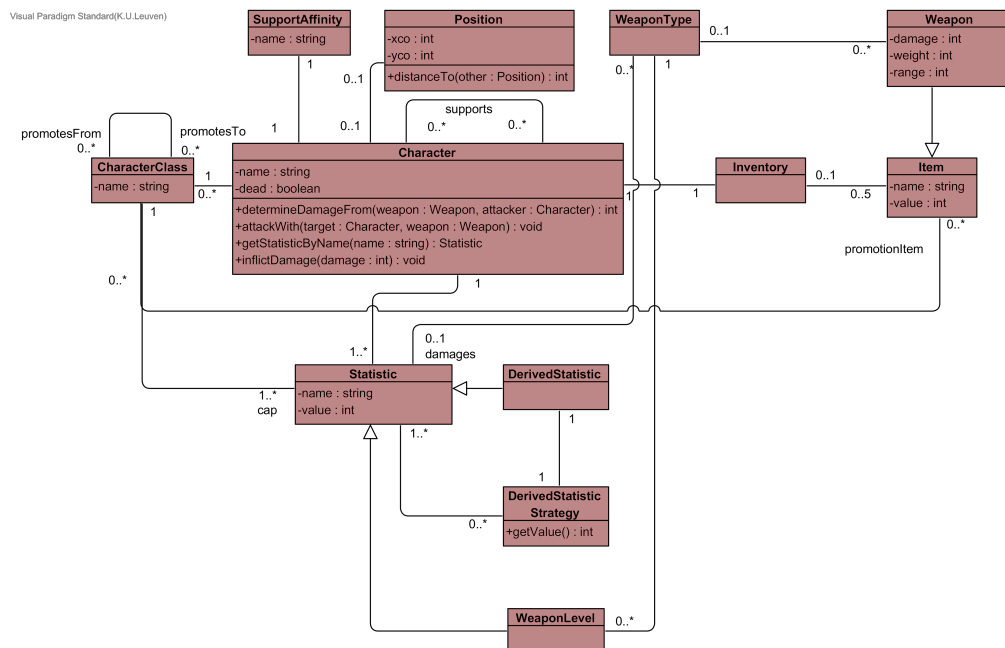
Deze masterproef heeft tot doel om automatisch op een gestructureerde manier uit te drukken welke informatie een UML-klassediagram juist bevat. Die informatie willen we op zijn beurt terug gebruiken om inconsistenties en kwaliteitsgebreken te detecteren. Concreter willen we predikatenlogica gebruiken 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 **klassen** 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 klassen 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 klassen die worden weergegeven in het diagram — niet meer en niet minder. *Character*, *Inventory*, *Item* enz. zijn dus *ClassObjects*. 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 expliciet 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: *Klassenaamoperatiennaam*\(*m* + 2), waarbij *m* het aantal argumenten dat als invoer wordt meegegeven aan de operatie. De signatuur ziet eruit als *Klasseoperatiennaam*(*o*, *p*₁, ..., *p*_{*m*}, *r*), waarbij *o* het object van logisch type *Object* waarop de operatie wordt opgeroepen, *p*₁ ... *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 *Klassenaamoperatiennaam*(*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:

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

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 $CharacterdetermineDamageFrom(o, p_1, r)$ 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:

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

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

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

2.0.5 Voorstellen van associaties

Voor elke associatie voegen we een predicaat toe dat beantwoordt aan volgend patroon: $ClassOneand\dots andClassM \setminus m$, 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:

$$\forall o_1[Object] \dots \forall o_m[Object](ClassOne \dots andClassM(o_1, \dots, o_m) \Rightarrow StaticClass(o_1ClassObj, o_1) \wedge \dots \wedge StaticClass(o_mClassObj, o_m))$$

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

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

- 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:

$$\forall c_1[Object] \dots \forall c_m[Object](StaticClass(c_1ClassObj, c_1) \wedge \dots \wedge StaticClass(c_mClassObj, c_m) \Rightarrow lowerBound_l \leq \#o_l : ClassOneand \dots ClassM(c_1, \dots, o_l, \dots, c_m) \leq upperBound_l)$$

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

$$\forall c_1[Object] \dots \forall c_m[Object](StaticClass(c_1ClassObj, c_1) \wedge \dots \wedge StaticClass(c_mClassObj, c_m) \Rightarrow \exists_{=upperbound_l} o_l(ClassOneand \dots andClassM(c_1, \dots, o_l, \dots, c_m)))$$

Voor *InventoryandItem\m* worden de volgende regels afgeleid:

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

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

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 *Objects* centraal stonden, doen we daar hier afstand van: we abstraheren *Objects* weg en concentreren ons in de plaats op *ClassObjects*. We gebruiken het diagram in figuur 2 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

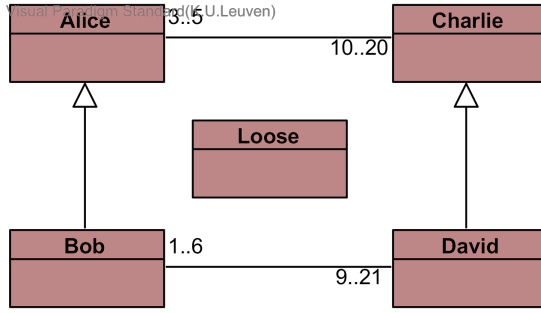
We bewaren het logisch type *ClassObject* en het predikaat *IsSupertypeOf(ClassObject, ClassObject)* exact zoals ze zijn in hoofdstuk 2 (en berekenen de transitieve sluiting horende bij *IsSupertypeOf* op dezelfde manier) en gebruiken daar bijkomend volgende predikaten:

- ***BiAssoc(ClassObject, ClassObject)***: drukt uit dat er een binaire associatie bestaat tussen de twee klassen.
- ***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*, 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 3.0.2. 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 gebruikt 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 $FO(\cdot)$, een uitbreiding van predikatenlogica. De basisblokken van een specificatie in IDP zijn als volgt:

- **Vocabulary:** Hier specificeert de ontwerper de logische types die bestaan in het beschouwd domein, predikaten en functies
- **Theorie:** Hier schrijft de ontwerper zinnen in $FO(\cdot)$ die bepalen welke structuren over het beschouwde vocabulary 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 vocabulary 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 vocabulary 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 vocabulary 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 codebestand 4.0.3 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 codebestand 4.0.3 staat de logische theorie die werd gegenereerd uit een combinatie van de diagrammen uit figuren 2 en 3.0.2 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.

kuis codebe-
standen flink
op

```

1 vocabulary V {
2   type LimitedInt = { 1..18 } isa int
3   type LimitedFloat = { 0.0; 0.5; 1.0; 1.5; 2.0; 2.5; 3.0; 3.5; 4.0; 4.5;
4     5.0; 5.5; 6.0; 6.5; 7.0; 7.5; 8.0; 8.5 } isa float
5   type LimitedString = { "SV0bSVp0wNsLeM1TCYkx"; "s0mJOUkvFu0Yxoypf0e2"; "
6     ucPRTrrfWBdnx8IbebrH"; "IIsx8mkhNB6tFKXhIh01"; "c8FoPQm8gzGloJi352R6";
7     "Q0wcTPcuxqohdJ00oYI5"; "nNhaMFI1sNq4FM9g9PFFK"; "THmoxPvd1k7axZ9Rx3Vo
8     "; "ps0JIEnr6CUFa2S1shdP"; "2ykrKTZkDAopHEMGzBgp"; "
9     YrU0vIjCy20ZLs39PE5t"; "LXDIF70705qElFzEF3WJ"; "NW3yPaSUa5NJERB5bpd0";
10    "NzD42F9XGUvbUNaZHU0q"; "s9Iudo9RU7iwdNeSJi8t"; "iN15Hkr0r9krS0lg2KER
11    "; "bk15G0Ix90UrFJfV1H7P"; "kC4BfZXQ4VDHxUmJ105G" } isa string
12   type bool constructed from { true, false }
13   type void constructed from { void }
14   type Object
15   type ClassObject constructed from { DerivedStatisticStrategy,
16     DerivedStatistic, WeaponType, CharacterClass, Inventory, Weapon,
17     Character, SupportAffinity, WeaponLevel, Statistic, Item }
18   RuntimeClass(ClassObject, Object)
19   StaticClass(ClassObject, Object)
20   IsDirectSupertypeOf(ClassObject, ClassObject)
21   IsSupertypeOf(ClassObject, ClassObject)
22
23   CharacterClassname(Object, LimitedString)
24   Weapondamage(Object, LimitedInt)
25   Weaponweight(Object, LimitedInt)
26   Weaponrange(Object, LimitedInt)
27   Charactername(Object, LimitedString)
28   SupportAffinityname(Object, LimitedString)
29   Statisticname(Object, LimitedString)
30   Itemvalue(Object, LimitedInt)
31   Itemname(Object, LimitedString)
32
33   DerivedStatisticStrategygetValue(Object, LimitedInt)
34   CharacterdetermineDamageFrom(Object, Object, LimitedInt)
35   StatisticgetValue(Object, LimitedInt)
36
37   StatisticandCharacterClass(Object, Object)
38   DerivedStatisticStrategyandStatistic(Object, Object)
39   WeaponTypeandWeaponLevel(Object, Object)
40   CharacterandStatistic(Object, Object)
41   CharacterandInventory(Object, Object)
42   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

```

4. DE ROL VAN IDP

```
94      ! o p1 r : CharacterdetermineDamageFrom(o, p1, r) => (StaticClass(
95          Character, o) & (StaticClass(Weapon, p1))).
96
97      ! o p1 : (StaticClass(Character, o) & (StaticClass(Weapon, p1))) => (?1 r :
98          CharacterdetermineDamageFrom(o, p1, r)).
99
100
101      ! o1 o2 : StatisticandCharacterClass(o1,o2) => ((StaticClass(Statistic, o1
102          )) & (StaticClass(CharacterClass, o2))).
103      ! o2 : ((StaticClass(CharacterClass, o2))) => (1 =< #{o1 :
104          StatisticandCharacterClass(o1,o2)}).
105      ! o1 : ((StaticClass(Statistic, o1))) => ?1 o2 :
106          StatisticandCharacterClass(o1,o2).
107
108      ! o1 o2 : DerivedStatisticStrategyandStatistic(o1,o2) => ((StaticClass(
109          DerivedStatisticStrategy, o1)) & (StaticClass(Statistic, o2))).
110      ! o1 : ((StaticClass(DerivedStatisticStrategy, o1))) => (1 =< #{o2 :
111          DerivedStatisticStrategyandStatistic(o1,o2)}).
112
113      ! o1 o2 : WeaponTypeandWeaponLevel(o1,o2) => ((StaticClass(WeaponType, o1)
114          ) & (StaticClass(WeaponLevel, o2))).
115      ! o2 : ((StaticClass(WeaponLevel, o2))) => ?1 o1 :
116          WeaponTypeandWeaponLevel(o1,o2).
117
118      ! o1 o2 : CharacterandStatistic(o1,o2) => ((StaticClass(Character, o1)) &
119          (StaticClass(Statistic, o2))).
120      ! o2 : ((StaticClass(Statistic, o2))) => ?1 o1 : CharacterandStatistic(o1,
121          o2).
122      ! o1 : ((StaticClass(Character, o1))) => (1 =< #{o2 :
123          CharacterandStatistic(o1,o2)}).
124
125      ! o1 o2 : CharacterandInventory(o1,o2) => ((StaticClass(Character, o1)) &
126          (StaticClass(Inventory, o2))).
127      ! o2 : ((StaticClass(Inventory, o2))) => ?1 o1 : CharacterandInventory(o1,
128          o2).
129      ! o1 : ((StaticClass(Character, o1))) => ?1 o2 : CharacterandInventory(o1,
130          o2).
131
132      ! o1 o2 : CharacterandCharacter(o1,o2) => ((StaticClass(Character, o1)) &
133          (StaticClass(Character, o2))).
134
135      ! o1 o2 : WeaponTypeandWeapon(o1,o2) => ((StaticClass(WeaponType, o1)) & (
136          StaticClass(Weapon, o2))).
137      ! o2 : ((StaticClass(Weapon, o2))) => ({o1 : WeaponTypeandWeapon(o1,o2)}
138          =< 1).
139
140      ! o1 o2 : DerivedStatisticStrategyandDerivedStatistic(o1,o2) => ((
141          StaticClass(DerivedStatisticStrategy, o1)) & (StaticClass(
142          DerivedStatistic, o2))).
143      ! o2 : ((StaticClass(DerivedStatistic, o2))) => ?1 o1 :
144          DerivedStatisticStrategyandDerivedStatistic(o1,o2).
145      ! o1 : ((StaticClass(DerivedStatisticStrategy, o1))) => ?1 o2 :
146          DerivedStatisticStrategyandDerivedStatistic(o1,o2).
147
148      ! o1 o2 : StatisticandWeaponType(o1,o2) => ((StaticClass(Statistic, o1)) &
149          (StaticClass(WeaponType, o2))).
150      ! o2 : ((StaticClass(WeaponType, o2))) => ({o1 : StatisticandWeaponType(
151          o1,o2)} =< 1).
152
153      ! o1 o2 : ItemandInventory(o1,o2) => ((StaticClass(Item, o1)) & (
154          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

```

1 vocabulary V {
2     type ClassObject constructed from { DerivedStatisticStrategy,
        DerivedStatistic, WeaponType, CharacterClass, Inventory, Weapon,
        Character, SupportAffinity, WeaponLevel, Statistic, Item, Loose, A, B,
        C, D}
3     type PrimitiveType constructed from { boolean, byte, character, double,
        floating, integer, long, short, astring, void}
4     IsSupertypeOf(ClassObject, ClassObject)
5
6     type LimitedInt = {1 .. 21} isa int
7
8     BiAssoc(ClassObject, ClassObject)
9     BiAssocLow(ClassObject, ClassObject, ClassObject, LimitedInt)
10    BiAssocHigh(ClassObject, ClassObject, ClassObject, LimitedInt)
11
12    ManyToMany(ClassObject, ClassObject)
13    LooseClass(ClassObject)
14    SubclassMorePermissiveMult(ClassObject, ClassObject, ClassObject)
15 }
16
17 theory T:V {
18
19     // ----- BAD DESIGN THAT MAY OCCUR IN UML -----
20
21     // many-to-many associations
22     ! x [ClassObject] y [ClassObject] : ManyToMany(x, y) <=> (BiAssoc(x, y) &
        ~ (? z [LimitedInt] : BiAssocHigh(x, y, x, z)) & ~ (? z [LimitedInt] :
        BiAssocHigh(x, y, y, z))).
23

```

4. DE ROL VAN IDP

```

24 // classes that are not associated with any other class
25 ! x [ClassObject] : LooseClass(x) <=> ~ ((? y [ClassObject] : BiAssoc(x, y
    )) | (? s [ClassObject] y [ClassObject] : IsSupertypeOf(s, x) & (
    BiAssoc(s, y)))).
26
27 ! x [ClassObject] y [ClassObject] : SubclassMorePermissiveMult(x, y, x)
28 <=> ((? sx [ClassObject] : IsSupertypeOf(sx, x)

```

&

```

((
BiAssoc
(
sx
,
y
)
&
((?
z1
[
Limite
]
z2
[
Limite
]
:
BiAssoc
(
x
,
y
,
x
,
z1
)
&
BiAssoc
(
sx
,
y
,
sx
,

```


z2
)
&
z1
<
z2
)

30

31

32

I

(?

sy

[
Cl
]

:

Is
(
sy
,

y
)

&

```
y
,
x
,
z1
)
&
BiAssoc
(
sx
,
sy
,
sx
,
z2
)
&
z1
<
z2
)
```

34

35

```
|
    (?
    sy
    [
    ClassObject
    ]
    :
    IsSupertyp
    (
    sy
    ,
    y
    )
    &
    Bi
    (
    x
    ,
    sy
    )
```

```
&
((?
z1
[
  LimitedInt
]
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,
94         DerivedStatistic, 1) <- .
94     BiAssocHigh(DerivedStatistic, DerivedStatisticStrategy,
95         DerivedStatistic, 1) <- .
95     BiAssocLow(DerivedStatistic, DerivedStatisticStrategy,
96         DerivedStatisticStrategy, 1) <- .
96     BiAssocHigh(DerivedStatistic, DerivedStatisticStrategy,
97         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) <- .

```

4. DE ROL VAN IDP

```
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) <-
121 |         BiAssocHigh(x, y, x, z).
122 |     ! x [ClassObject] y [ClassObject] z [nat] : BiAssocHigh(y, x, y, z) <-
123 |         BiAssocHigh(x, y, y, z).
124 |     ! x [ClassObject] y [ClassObject] z [nat] : BiAssocLow(y, x, x, z) <-
125 |         BiAssocLow(x, y, x, z).
126 |     ! x [ClassObject] y [ClassObject] z [nat] : BiAssocLow(y, x, y, z) <-
127 |         BiAssocLow(x, y, y, z).
128 | }
129 |
130 | //! x [ClassObject] y [ClassObject] : BiAssocDef(x, y) <=> BiAssocDef(
131 |     y, x).
132 |
133 | theory U:V {
134 |     ~ (? z [ClassObject] : BiAssoc(Loose, z)).
135 | }
136 |
137 | structure thestruct:V {
138 | }
139 |
140 | procedure main() {
141 |     print(modelexpand(T, thestruct)[1])
142 | }
```

chap-rol-idp/defs.idp

Hoofdstuk 5

Simuleren van gedrag op basis van een sequentiediagram

Een ander populair type van UML-diagram is het sequentiediagram. ~~In zulk een diagram worden er instanties van klassen gedefinieerd in een klassediagram gebruikt.~~ Waar klassediagrammen de informatie bevat in klassen en de verbanden tussen klassen benoemen, beschrijven sequentiediagrammen het gedrag van instanties van de klassen. Deze instanties ~~kunnen zowel berichten sturen als ontvangen~~ 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 5 geeft een voorbeeld van een sequentiediagram gebaseerd op het klassediagram voorgesteld in figuur 2.

~~Sequentiediagram gebaseerd op het klassediagram van figuur 2~~

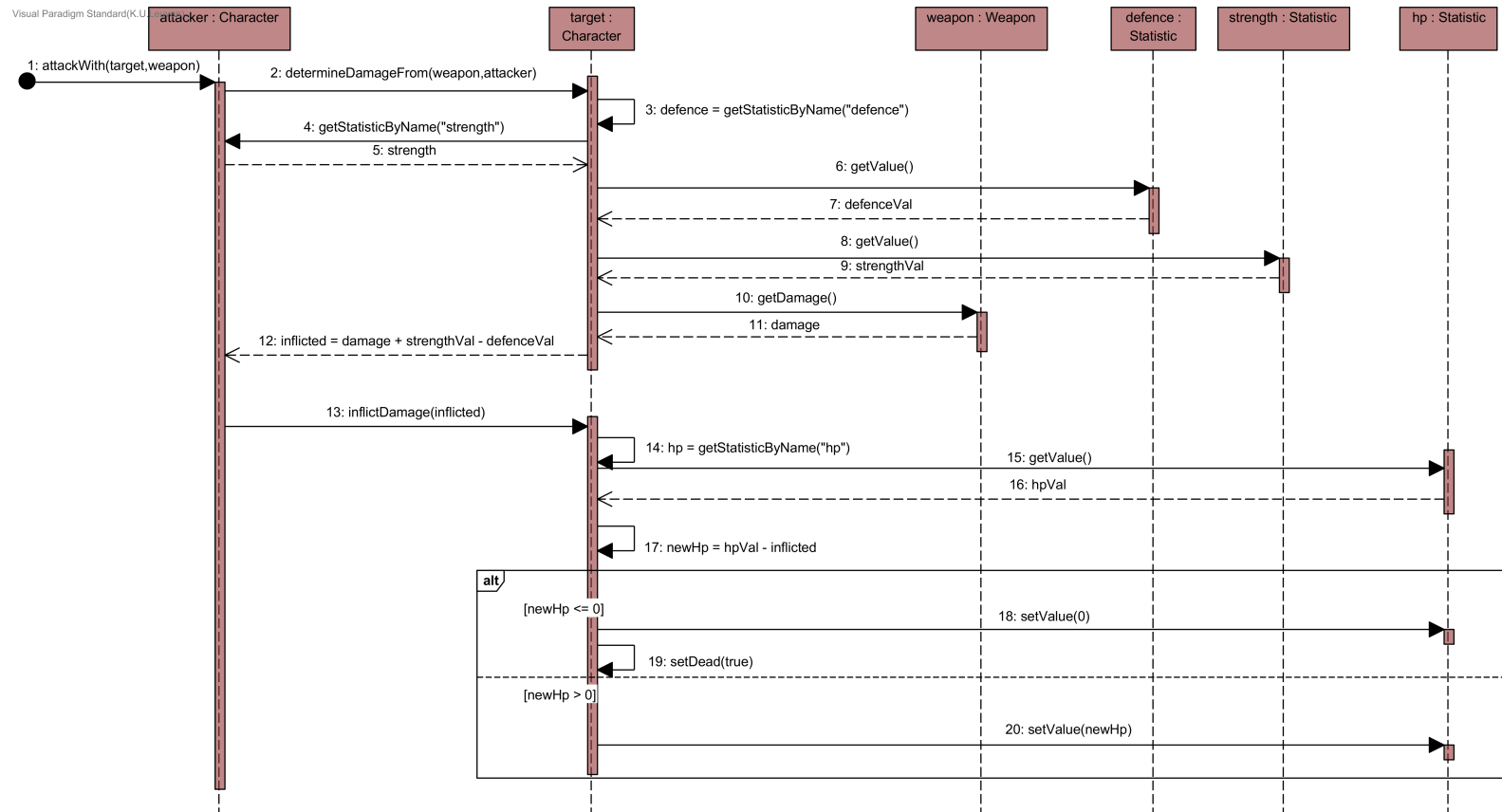
Een instantie wordt voorgesteld door een kader met daarin tekst volgens het patroon *instantienaam : klassenaam*. Dit wil zeggen dat bijvoorbeeld *attacker* een instantie is van de klasse *Character*. Vanuit elk kader vertrekt ook een streepjeslijn: de **levenslijn**. Deze levenslijn kan ingevuld worden 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 5 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 mogelijk is 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 ~~eapteren~~modelleren.



FIGUUR 5.1: [Sequentiediagram gebaseerd op het klassediagram van figuur 2](#)

5.0.1 De keuze voor lineaire tijdscalculus

UML-diagrammen schrijven mogelijke toestanden van ~~en acties~~, en acties, op softwaresystemen voor, ~~en die~~. 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 (~~in deze tekst verder afgekort tot LTC~~) ~~kan exact deze wensen bewerkstelligen~~ [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 ~~beschouwen we het sequentiediagram van werken we deze uitbreiding uit voor het sequentiediagram in~~ figuur 5.

5.0.2 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 inertiael en introduceren we deze ~~predicaten~~ 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 inertiael. 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 inertiael. Voor ~~*defenceVal* krijgen we~~: alle zulke variabelen definiëren we ook predicaten zoals hierboven voor *attacker*.

- ~~$DefenceValT(Time, LimitedInt)$~~
- ~~$I_DefenceValTLimitedInt$~~
- ~~$C_DefenceValT(Time, LimitedInt)$~~

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 inertiael. 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 ~~*uncauses-predicaat*~~ oncausatiepredicaat toe omdat het mogelijk is dat een attribuut meer dan één waarde heeft op een bepaald tijdstip. ~~Langs deze weg kunnen we bepaalde waarden van zulk een attribuut ongedaan maken~~ Met dit predicaat geven we aan dat bepaalde waarden die voor een bepaalde tijdstap gelden ongedaan moeten worden gemaakt in de volgende tijdstap.

5.0.3 Uitbreiden van de theorie

Voor elke inertiale eigenschap van het systeem moeten er twee dingen gebeuren: Toestandzinnen opstellen en voorwaarden voor ~~*causes-zinnen en uncauses-zinnen*~~ *causatiezinnen en oncausatiezinnen* specificeren. Het resultaat is een inductieve definitie die de inertiale predicaten definieert en een inductieve definitie die de ~~*causes-predicaten en uncauses-predicaten*~~ *causatiepredicaten en oncausatiepredicaten* definieert.

Toestandszinnen opstellen

Toestandszinnen worden geschreven in termen van ~~initialize-predicaten, causes-predicaten en uneauses-predicaten~~ begintoestandspredicaten, causatiepredicaten en oncausatiepredicaten. Ze garanderen dat inertiaële 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:

$$\begin{aligned} & \underline{\forall s[SDPoint](SDPointAt(Start, s) \leftarrow I_SDPointAt(s))} \\ & \underline{\forall s[SDPoint](SDPointAt(Start, s) \leftarrow I_SDPointAt(s))}. \end{aligned} \quad (5.1)$$

De volgende definities gebruiken ~~causes~~ het causatiepredicaat:

$$\begin{aligned} & \underline{\forall t[Time]\forall s[SDPoint](SDPointAt(Next(t), s) \leftarrow C_SDPointAt(Next(t), s))} \\ & \underline{\forall t[Time]\forall s[SDPoint](SDPointAt(Next(t), s) \leftarrow SDPointAt(t, s)} \\ & \quad \wedge \neg(\exists s1[SDPoint](C_SDPointAt(Next(t), s1)))) \\ & \underline{\forall t[Time]\forall s[SDPoint](SDPointAt(Next(t), s) \leftarrow C_SDPointAt(Next(t), s))}. \end{aligned} \quad (5.2)$$

$$\begin{aligned} & \underline{\forall t[Time]\forall s[SDPoint](SDPointAt(Next(t), s) \leftarrow SDPointAt(t, s)} \\ & \quad \wedge \neg(\exists s1[SDPoint](C_SDPointAt(Next(t), s1))))}. \end{aligned} \quad (5.3)$$

~~De tweede zin~~ 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~~ Tattacker).

Voor klasseattributen verloopt dit ook gelijkaardig, maar ~~gebruiken we voor het derde deel het uneauses-predicaat, bijvoorbeeld voor value in Statistie~~ we wijken af van het formaat van zin 5.3 door als volgt het oncausatiepredicaat te gebruiken:

$$\begin{aligned} & \forall t[Time]\forall s[Statistic]\forall i[LimitedInt](Statisticvalue(Next(t), s, i) \\ & \quad \leftarrow Statisticvalue(t, s, i) \wedge \neg Cn_Statisticvalue(Next(t), s, i)). \end{aligned}$$

Voorwaardes voor ~~causes~~ causatie en ~~uncauses~~ 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 ~~causes-zin en uncauses-zin~~ causatiezin en oncausatiezin die volgen uit instructie 18:

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

$$\begin{aligned} & \forall t[Time] \forall s[Statistic] \forall i[LimitedInt] (Cn_Statisticvalue(Next(t), s, v) \\ & \leftarrow SDPointAt(Next(t), 18) \wedge HpT(t, s) \wedge Statisticvalue(t, s, i) \wedge \neg(i = 0)). \end{aligned}$$

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 ~~uncauses-zin~~ oncausatiezin om te verzekeren dat de vorige waarde wordt gewist.

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

$$\begin{aligned} & \forall t[Time] \forall s[Statistic] \forall i[LimitedInt] (C_Statisticvalue(t, s, i) \\ & \leftarrow SDPointAt(t, 20) \wedge HpT(t, s) \wedge NewHpT(t, i)). \end{aligned}$$

$$\begin{aligned} & \forall t[Time] \forall s[Statistic] \forall i[LimitedInt] (Cn_Statisticvalue(Next(t), s, i) \\ & \leftarrow SDPointAt(Next(t), 20) \wedge HpT(t, s) \wedge Statisticvalue(t, s, i) \wedge \neg NewHpT(Next(t), i)). \end{aligned}$$

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 ~~causes-zinnen~~ 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:

$$\begin{aligned}
& \frac{\forall t[Time] \forall s[SDPoint] (C_SDPointAt(Next(t), (s+1) \leftarrow SDPointAt(t, s) \\
& \quad \wedge \neg((s=17) \vee (s=19))) \\
& \quad \frac{\forall t[Time] (C_SDPointAt(Next(t), 18) \leftarrow SDPointAt(t, 17) \wedge \\
& \quad \quad (\exists i[LimitedInt] (NewHpT(t, i) \wedge i \leq 0))) \\
& \quad \frac{\forall t[Time] (C_SDPointAt(Next(t), 20) \leftarrow SDPointAt(t, 17) \wedge \\
& \quad \quad (\exists i[LimitedInt] (NewHpT(t, i) \wedge i > 0))) \\
& \quad \forall t[Time] (C_SDPointAt(Next(t), 21) \leftarrow SDPointAt(t, 19) \vee SDPointAt(t, 20))
\end{aligned}$$

$$\begin{aligned}
& \frac{\forall t[Time] \forall s[SDPoint] (C_SDPointAt(Next(t), (s+1) \leftarrow SDPointAt(t, s) \\
& \quad \wedge \neg((s=17) \vee (s=19)))}. \tag{5.4}
\end{aligned}$$

$$\begin{aligned}
& \frac{\forall t[Time] (C_SDPointAt(Next(t), 18) \leftarrow SDPointAt(t, 17) \wedge \\
& \quad (\exists i[LimitedInt] (NewHpT(t, i) \wedge i \leq 0)))}. \tag{5.5}
\end{aligned}$$

$$\begin{aligned}
& \frac{\forall t[Time] (C_SDPointAt(Next(t), 20) \leftarrow SDPointAt(t, 17) \wedge \\
& \quad (\exists i[LimitedInt] (NewHpT(t, i) \wedge i > 0)))}. \tag{5.6}
\end{aligned}$$

$$\begin{aligned}
& \frac{\forall t[Time] (C_SDPointAt(Next(t), 21) \leftarrow SDPointAt(t, 19) \vee SDPointAt(t, 20))}. \tag{5.7}
\end{aligned}$$

~~De eerste zin drukt uit dat de normale uitvoeringsvolgorde wordt onderbroken wanneer wordt geselecteerd welk deel~~ Zin 5.4 verzekert het juiste gedrag van 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 wordt uitgevoerd tegenkomt en wanneer het einde van het if-deel wordt bereikt. De volgende twee zinnen drukken uit dat het correcte deel van de is bereikt. Zinnen 5.5 en 5.6 controleren de voorwaarde voor de uitvoering van het if-else-constructie wordt geselecteerd na instructie 17. De laatste zin 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 ~~causes-zinnen~~ 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 ~~causes-zin~~ 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*.

5.0.4 Resultaat

~~Codebestand A geeft het resultaat weer van het volgen van de regels die zijn uitgelijnd in dit hoofdstuk.~~

Zie appendix A voor het volledig model voor het gedrag van het sequentiediagram in figuur 5.

5.0.4 Interactie tussen meerdere sequentiediagrammen

Bijlagen

Bijlage A

IDP-bestand voor sequentiediagram van het spelvoorbeeld

```
1 include<LTC>
2
3 LTCvocabulary V {
4   type Time isa nat
5   Start: Time
6   partial Next(Time) : Time
7
8   type SDPoint = { 1..21 } isa nat
9
10  SDPointAt(Time,SDPoint)
11  I_SDPointAt(SDPoint)
12  C_SDPointAt(Time,SDPoint)
13
14  type LimitedInt isa int
15  type LimitedFloat isa float
16  type LimitedString isa string
17  type boolean constructed from { T, F }
18  type void constructed from { null }
19
20  type Statistic
21  type Item
22  type Position
23  type DerivedStatisticStrategy
24  type DerivedStatistic isa Statistic
25  type WeaponType
26  type CharacterClass
27  type Inventory
28  type Weapon isa Item
29  type Character
30  type SupportAffinity
31  type WeaponLevel isa Statistic
32
33  DamageT(Time, LimitedInt)
34  I_DamageT(LimitedInt)
35  C_DamageT(Time, LimitedInt)
36
37  StrengthT(Time, Statistic)
38  I_StrengthT(Statistic)
```

A. IDP-BESTAND VOOR SEQUENTIEDIAGRAM VAN HET SPELVOORBEELD

```

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)

```

```

102     I_Weaponweight(Weapon, LimitedInt)
103     C_Weaponweight(Time, Weapon, LimitedInt)
104     Cn_Weaponweight(Time, Weapon, LimitedInt)
105
106     Weaponrange(Time, Weapon, LimitedInt)
107     I_Weaponrange(Weapon, LimitedInt)
108     C_Weaponrange(Time, Weapon, LimitedInt)
109     Cn_Weaponrange(Time, Weapon, LimitedInt)
110
111     Characterdead(Time, Character, boolean)
112     I_Characterdead(Character, boolean)
113     C_Characterdead(Time, Character, boolean)
114     Cn_Characterdead(Time, Character, boolean)
115
116     Charactername(Time, Character, LimitedString)
117     I_Charactername(Character, LimitedString)
118     C_Charactername(Time, Character, LimitedString)
119     Cn_Charactername(Time, Character, LimitedString)
120
121     SupportAffinityname(Time, SupportAffinity, LimitedString)
122     I_SupportAffinityname(SupportAffinity, LimitedString)
123     C_SupportAffinityname(Time, SupportAffinity, LimitedString)
124     Cn_SupportAffinityname(Time, SupportAffinity, LimitedString)
125
126     Statisticvalue(Time, Statistic, LimitedInt)
127     I_Statisticvalue(Statistic, LimitedInt)
128     C_Statisticvalue(Time, Statistic, LimitedInt)
129     Cn_Statisticvalue(Time, Statistic, LimitedInt)
130
131     Statisticname(Time, Statistic, LimitedString)
132     I_Statisticname(Statistic, LimitedString)
133     C_Statisticname(Time, Statistic, LimitedString)
134     Cn_Statisticname(Time, Statistic, LimitedString)
135
136     Itemvalue(Time, Item, LimitedInt)
137     I_Itemvalue(Item, LimitedInt)
138     C_Itemvalue(Time, Item, LimitedInt)
139     Cn_Itemvalue(Time, Item, LimitedInt)
140
141     Itemname(Time, Item, LimitedString)
142     I_Itemname(Item, LimitedString)
143     C_Itemname(Time, Item, LimitedString)
144     Cn_Itemname(Time, Item, LimitedString)
145
146     CharacterandPosition(Character, Position)
147     WeaponTypeandWeaponLevel(WeaponType, WeaponLevel)
148     CharacterClassandCharacterClass(CharacterClass, CharacterClass)
149     CharacterandSupportAffinity(Character, SupportAffinity)
150     DerivedStatisticStrategyandDerivedStatistic(DerivedStatisticStrategy,
        DerivedStatistic)
151     CharacterandCharacter(Character, Character)
152     StatisticandCharacterClass(Statistic, CharacterClass)
153     DerivedStatisticStrategyandStatistic(DerivedStatisticStrategy, Statistic)
154     CharacterandStatistic(Character, Statistic)
155     CharacterandCharacterClass(Character, CharacterClass)
156     ItemandCharacterClass(Item, CharacterClass)
157     StatisticandWeaponType(Statistic, WeaponType)
158     CharacterandInventory(Character, Inventory)
159     WeaponTypeandWeapon(WeaponType, Weapon)
160     ItemandInventory(Item, Inventory)
161 }
162 theory T:V {
163     {

```

A. IDP-BESTAND VOOR SEQUENTIEDIAGRAM VAN HET SPELVOORBEELD

```

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,

```



```

201         x) & Characterdead(t, x, v) & ~(v = T).
202
203     ! t [Time] x [Statistic] v [LimitedString] : C_Statisticname(t, x,
204         v) <- false.
205
206     ! t [Time] x [Statistic] v [LimitedString] : Cn_Statisticname(t, x
207         , v) <- false.
208
209     ! t [Time] x [SupportAffinity] v [LimitedString] :
210         C_SupportAffinityname(t, x, v) <- false.
211     ! t [Time] x [SupportAffinity] v [LimitedString] :
212         Cn_SupportAffinityname(t, x, v) <- false.
213
214     ! t [Time] s [SDPoint] : C_SDPointAt(Next(t), (s+1)) <- SDPointAt(t, s
215         ) & ~(s = 17) | (s = 19) | (s = 21)).
216
217     ! t [Time] : C_SDPointAt(Next(t), 18) <- SDPointAt(t, 17) & ( ? newHp
218         [LimitedInt] : NewHpT(t, newHp) & newHp =< 0).
219
220     ! t [Time] : C_SDPointAt(Next(t), 20) <- SDPointAt(t, 17) & ( ? newHp
221         [LimitedInt] : NewHpT(t, newHp) & newHp > 0).
222
223     ! t [Time] : C_SDPointAt(Next(t), 21) <- SDPointAt(t, 19) | SDPointAt(
224         t, 20).
225
226     ! t [Time] : C_SDPointAt(Next(t), 1) <- SDPointAt(t, 21).
227
228     ! t [Time] defence [Statistic] : C_DefenceT(t, defence) <- SDPointAt(
229         t, 3) & (? target [Character] : TargetT(t, target) &
230         CharacterandStatistic(target, defence) & Statisticname(t, defence
231         , "defence")).
232
233     ! t [Time] strength [Statistic] : C_StrengthT(Next(t), strength) <-
234         SDPointAt(t, 4) & (? attacker [Character] : AttackerT(t, attacker
235         ) & CharacterandStatistic(attacker, strength) & Statisticname(t,
236         strength, "strength")).
237
238     ! t [Time] x [LimitedInt] : C_DefenceValT(Next(t), x) <- SDPointAt(t,
239         6) & (? o [Statistic] : DefenceT(t, o) & Statisticvalue(t, o, x)).
240
241     ! t [Time] x [LimitedInt] : C_StrengthValT(Next(t), x) <- SDPointAt(t,
242         8) & (? o [Statistic] : StrengthT(t, o) & Statisticvalue(t, o, x)
243         ).
244
245     ! t [Time] x [LimitedInt] : C_DamageT(Next(t), x) <- SDPointAt(t, 10)
246         & (? o [Weapon] : WeaponT(t, o) & Weapondamage(t, o, x)).
247
248     ! t [Time] inflicted [LimitedInt] : C_InflictedT(t, inflicted) <-
249         SDPointAt(t, 12) & (? damage [LimitedInt] strengthVal [LimitedInt]
250         defenceVal [LimitedInt] : DamageT(t, damage) & StrengthValT(t,
251         strengthVal) & DefenceValT(t, defenceVal) & inflicted=damage+
252         strengthVal-defenceVal).
253
254     ! t [Time] hp [Statistic] : C_HpT(t, hp) <- SDPointAt(t, 14) & (?
255         target [Character] : TargetT(t, target) & CharacterandStatistic(
256         target, hp) & Statisticname(t, hp, "hp")).
257
258     ! t [Time] x [LimitedInt] : C_HpValT(Next(t), x) <- SDPointAt(t, 15) &
259         (? o [Statistic] : HpT(t, o) & Statisticvalue(t, o, x)).
260
261     ! t [Time] newHp [LimitedInt] : C_NewHpT(t, newHp) <- SDPointAt(t, 17)
262         & (? hpVal [LimitedInt] inflicted [LimitedInt] : HpValT(t, hpVal)
263         & InflictedT(t, inflicted) & newHp=hpVal-inflicted).
264
265 }
266 {
267
268     ! s [SDPoint] : SDPointAt(Start, s) <- I_SDPointAt(s).
269
270     ! t [Time] s [SDPoint] : SDPointAt(Next(t), s) <- C_SDPointAt(Next
271         (t), s).
272
273     ! t [Time] s [SDPoint] : SDPointAt(Next(t), s) <- SDPointAt(t, s)
274         & ~(? s1 [SDPoint] : C_SDPointAt(Next(t), s1)).
275
276
277     ! x [LimitedInt] : DamageT(Start, x) <- I_DamageT(x).
278
279     ! t [Time] x [LimitedInt] : DamageT(t, x) <- C_DamageT(t, x).
280
281     ! t [Time] x [LimitedInt] : DamageT(Next(t), x) <- DamageT(t, x) &
282         ~( ? x1 [LimitedInt] : C_DamageT(Next(t), x1) & ~(x = x1)).
283
284

```

A. IDP-BESTAND VOOR SEQUENTIEDIAGRAM VAN HET SPELVOORBEELD

```

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)

```

A. IDP-BESTAND VOOR SEQUENTIEDIAGRAM VAN HET SPELVOORBEELD

```

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
356     , v).
357
358     ! t [Time] x [SupportAffinity] : ?1 v [LimitedString] :
359     SupportAffinityname(t, x, v).
360
361     ! x [Statistic] : ?1 v [LimitedInt] : I_Statisticvalue(x, v).
362     ! t [Time] x [Statistic] : ?1 v [LimitedInt] : Statisticvalue(t, x, v)
363     .
364
365     ! x [Statistic] : ?1 v [LimitedString] : I_Statisticname(x, v).
366     ! t [Time] x [Statistic] : ?1 v [LimitedString] : Statisticname(t, x,
367     v).
368
369     ! x [Item] : ?1 v [LimitedInt] : I_Itemvalue(x, v).
370     ! t [Time] x [Item] : ?1 v [LimitedInt] : Itemvalue(t, x, v).
371
372     ! x [Item] : ?1 v [LimitedString] : I_Itemname(x, v).
373     ! t [Time] x [Item] : ?1 v [LimitedString] : Itemname(t, x, v).
374
375     ! x2 [Position] : #{ x1 [Character] : CharacterandPosition(x1, x2)} =<
376     1.
377     ! x1 [Character] : #{ x2 [Position] : CharacterandPosition(x1, x2)} =<
378     1.
379
380     ! x2 [WeaponLevel] : ?1 x1 [WeaponType] : WeaponTypeandWeaponLevel(x1,
381     x2).
382
383     ! x2 [SupportAffinity] : ?1 x1 [Character] :
384     CharacterandSupportAffinity(x1, x2).
385     ! x1 [Character] : ?1 x2 [SupportAffinity] :
386     CharacterandSupportAffinity(x1, x2).
387
388     ! x2 [DerivedStatistic] : ?1 x1 [DerivedStatisticStrategy] :
389     DerivedStatisticStrategyandDerivedStatistic(x1, x2).
390     ! x1 [DerivedStatisticStrategy] : ?1 x2 [DerivedStatistic] :
391     DerivedStatisticStrategyandDerivedStatistic(x1, x2).
392
393     ! x2 [CharacterClass] : #{ x1 [Statistic] : StatisticandCharacterClass
394     (x1, x2)} >= 1.
395     ! x1 [Statistic] : ?1 x2 [CharacterClass] : StatisticandCharacterClass
396     (x1, x2).
397
398     ! x1 [DerivedStatisticStrategy] : #{ x2 [Statistic] :
399     DerivedStatisticStrategyandStatistic(x1, x2)} >= 1.
400
401     ! x2 [Statistic] : ?1 x1 [Character] : CharacterandStatistic(x1, x2).
402     ! x1 [Character] : #{ x2 [Statistic] : CharacterandStatistic(x1, x2)}
403     >= 1.
404
405     ! x1 [Character] : ?1 x2 [CharacterClass] : CharacterandCharacterClass
406     (x1, x2).
407
408     ! x2 [WeaponType] : #{ x1 [Statistic] : StatisticandWeaponType(x1, x2)
409     } =< 1.
410
411     ! x2 [Inventory] : ?1 x1 [Character] : CharacterandInventory(x1, x2).
412     ! x1 [Character] : ?1 x2 [Inventory] : CharacterandInventory(x1, x2).

```

A. IDP-BESTAND VOOR SEQUENTIEDIAGRAM VAN HET SPELVOORBEELD

```

399      ! x2 [Weapon] : #{ x1 [WeaponType] : WeaponTypeandWeapon(x1, x2)} =<
400      1.
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_SDPPointAt = { 1 }
411
412      LimitedInt = { -3..3 }
413      LimitedFloat = { 0.0; 0.5; -0.5; 0.5; -0.5}
414      LimitedString = { "FcPBe4HTw3ZpeLBKRbR6"; "UNsSbSrxsg21BWTZuV41"; "
              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 5

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