TPUPT –Gebruikershandleiding

René Ladan, r.c.ladan@gmail.com

3 oktober 2006

1 Introductie

TPUPT staat voor Two Phase UML Phunction Transformer, het afstudeerproject van de auteur. Het biedt de mogelijkheid om UML-modellen te transformeren. Hierbij staat UML voor Unified Modeling Language. Het transformeren van UML-modellen gebeurt in twee fasen. In de eerste fase worden de reeds gedefinieerde functies omgezet in een programma, in de tweede fase wordt dit programma toegepast op een UML-model.

Bij TPUPT worden drie modelbestanden met daarin verschillende functies geleverd:

- reservation.xml, hierin staan functies voor de UML Component Method. Ook staan hierin enkele voorbeelden van het toepassen van deze functies. Het model is gebaseerd op het hotelreserveringssysteem zoals beschreven in [2].
- student.xml, dit model is analoog aan reservation.xml, maar is gebaseerd op een voorbeeldsysteem voor het toekennen van cijfers aan studenten.
- aosd.xml, hierin staan functies en enkele toepassingsvoorbeelden voor het aspectgeöriënteerd ontwikkelen van software. Momenteel is deze methode nog niet in de UML-standaard opgenomen. Een goede introductie over deze methode is te vinden in [3].

2 Installatie

Voor het gebruiken van TPUPT zijn drie dingen nodig:

- Een programma dat XSLT 2.0 en XPath 2.0 kan verwerken. Voorbeelden hiervan zijn Saxon 7.9 of nieuwer [6] en XML Spy 2006 Professional of nieuwer [7].
- Een programma waarmee UML-modellen bewerkt kunnen worden. Hierbij is het van belang dat de modelbestanden in het formaat van de XMI 1.2 standaard [5] worden opgeslagen, anders kan TPUPT er niet mee werken. Een programma dat hieraan voldoet is ArcStyler 5.1 [1]. De modellen zelf dienen aan de UML 1.4 standaard [4] te voldoen.
- De bestanden tpupt.xsl, base.xsl, en libxmi.xsl dienen in dezelfde map staan. tpupt.xsl is het hoofdprogramma, de andere twee bestanden zijn hulpbestanden en dienen niet direct te worden uitgevoerd.

Wanneer dit voltooid is, kan een functie zoals beschreven in sectie 3 worden uitgevoerd. Indien dit goed gaat, kunnen de functies op zelfgedefinieerde invoer worden toegepast. Dit staat beschreven in sectie 4.

3 De transformatiestappen

TPUPT wordt met drie modellen (reservation.xml, student.xml, aosd.xml) geleverd, die allemaal kunnen worden getransformeerd. De procedure is voor al deze modellen analoog. Alle voorbeelden zullen worden uitgevoerd met Saxon en ArcStyler, eventueel kunnen ook andere programma's gebruikt worden.

Zoals is beschreven in de introductie gebeurt het omzetten in twee fasen:

1. De functiedefinities dienen in XSLT te worden omgezet:

   ```
   saxon N tpupt.xml
   ```

   Hierbij is *N* de naam van het om te zetten modelbestand. Deze stap genereert een XSLT-bestand uit de functiedefinities in *N*. De naam van dit bestand is out-*M*.xsl, waarbij *M* de naam van het model in *N* is. Tijdens deze stap zullen alle functiedefinities met hun formele parameters op de console worden weergegeven, zodat kan worden gecontroleerd of alle functiedefinities goed herkend worden.

2. De functies kunnen nu op het model worden toegepast:

   ```
   saxon -o U N X
   ```

   Hierbij is *U* de naam van het resulterende modelbestand, *N* de naam van het om te zetten model, en *X* de naam van het gegeneerde XSLT-bestand (standaard out-*M*.xsl). Tijdens deze stap zullen alle functietoepassingen met hun actuele parameters op de console worden weergegeven, zodat kan worden gecontroleerd of alle functietoepassingen goed herkend worden. Nu het transformeren voltooid is, kan *U* in ArcStyler worden bekeken en eventueel worden aangepast.

4 Zelf functies toepassen

In het pakket example van student.xml en aosd.xml en het pakket auto-hotel van reservation.xml staan enkele voorbeelden van functietoepassingen. Deze kunnen bekeken worden om een eerste indruk van het toepassen van de functies te krijgen.

De volgende procedure gaat ervan uit dat er een nieuwe functietoepassing in een van de drie bijgeleverde modellen wordt gespecificeerd. Als voorbeeld is hier de functietoepassing *do_csa* uit student.xml genomen. Deze functietoepassing is te zien in figuren 1 en 2.

1. Ergens in het model dient een nieuw (naamloos) pakket met het stereotype <<apply-function>> aangemaakt te worden. Dit stereotype heeft een tag 'name', dat de naam van de toe te passen functie bevat.

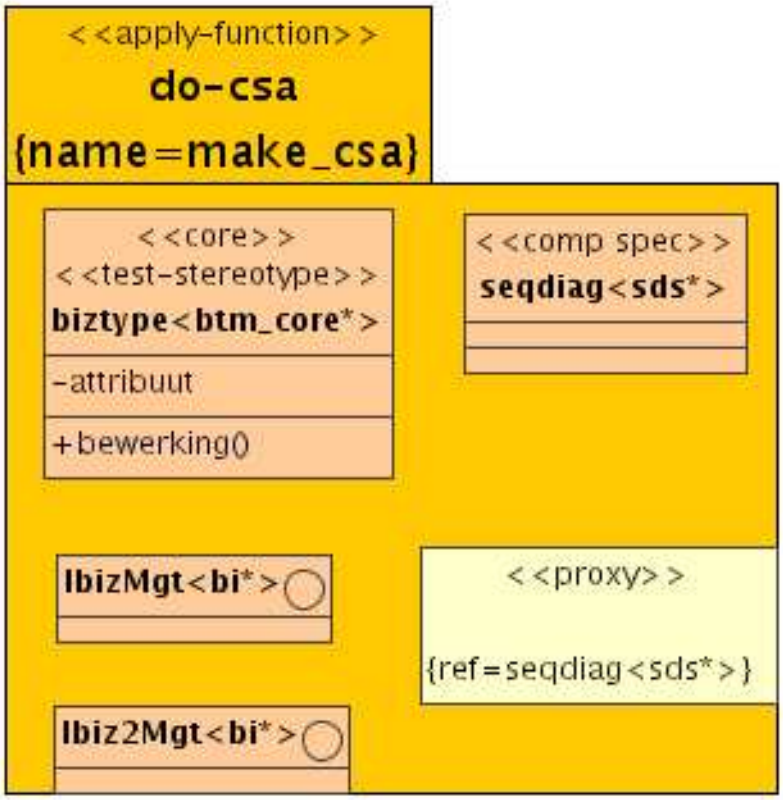
2. De actuele parameters dienen te worden opgegeven. Deze dienen overeen te komen met de formele parameters van de toe te passen functie. Functies zijn te herkennen als pakketten met het stereotype <<function>>. Voor elke formele parameter dient gewoonlijk 1 actuele parameter te worden ingevuld. Het soort (interface, klasse, enzovoorts) en de stereotypen van elke actuele parameter dienen overeen te komen met de formele tegenhanger. Formele parameters zijn te herkennen aan het feit dat ze geen uitgaande dependencies hebben en dat de naam ervan tussen '<' en '>' staat. Het formaat van de naam van de actuele parameter dient $actueel < formeel >$ te zijn. Hierbij is *actueel* de naam van de actuele parameter en $formeel$ de naam van de formele tegenhanger van de actuele parameter. De '<' en '>' dienen als zodanig in de naam te staan.

   - Indien de naam van de formele parameter een ?, *, of + bevat kunnen er respectievelijk ten hoogste 1, tenminste 0, of tenminste 1 actuele tegenhangers zijn. Het formaat van de actuele parameter dient ook hier $actueel < formeel >$ te zijn, dus met de ?, *, of +.

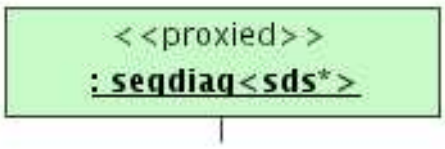
     In figuur 1 dienen de interfaces IbizMgt en Ibiz2Mgt als actuele parameters voor de formele parameter <bi*>.

   - De 'inhoud' van de actuele parameter wordt niet gebruikt. Een uitzondering hierop is indien er voor attributen en/of operaties dependencies zijn tussen de attributen en/of operaties van de formele parameters en de attributen en/of operaties van de formele uitvoer: in dit geval worden de attributen en/of operaties van de actuele parameter gekopieerd naar het resulterende element.

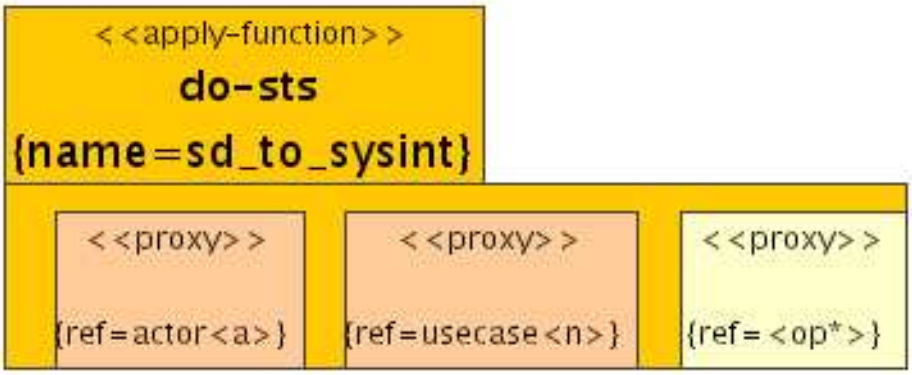
     Deze uitzondering komt voor in de functie *make_csa* (zie figuur 7), dus het attribuut 'attribuut' en de operatie 'bewerking' van de actuele parameter biztype<btm_core*> worden gekopieerd naar het resultaat van de functietoepassing.



Figuur 1: Een toepassing van de functie *make_csa*



Figuur 2: Aanvullende elementen voor de toepassing van de functie *make_csa*



Figuur 3: Een toepassing van de functie *sd_to_sysint*

- Indien een formele parameter het stereotype <<proxy>> bevat dient er ergens in de functie een element met het stereotype <<proxied>> te zijn. De naam van dit laatste element dient overeen te komen met de tag 'ref' van het eerste element. Elementen met het stereotype <<proxy>> hoeven geen naam te hebben.

  Voor de bijhorende actuele parameter geldt dat ook deze uit twee delen bestaat:

  - Het <<proxy>>-gedeelte dient in het pakket te staan. Dit element dient een tag 'ref' te bevatten. De inhoud van deze tag dient $actueel < formeel >$ te zijn.
  - Het <<proxied>>-gedeelte dient ergens in hetzelfde <<apply-function>>-pakket, maar in een ander diagram, te staan. De naam van dit element dient overeen te komen met de inhoud van de tag 'ref' van bovenstaand <<proxy>>-gedeelte. Eventueel kan voor dit element een nieuw plaatje worden aangemaakt.

  Een voorbeeld hiervan is te zien in de figuren 1 en 2: het object met het stereotype <<proxy>> verwijst door middel van de tag 'ref' naar de classifier-rol seqdiag<sds*>.

- Berichten in sequentiediagrammen kunnen in operaties worden omgezet door een object als proxy naar deze berichten te definiëren. Hierbij is het niet nodig om in de proxy de parameters en het resulterende type van de te definiëren operatie op te nemen. Alle om te zetten berichten dienen een call-bericht te zijn.

  De naam van het bericht is als volgt opgebouwd:

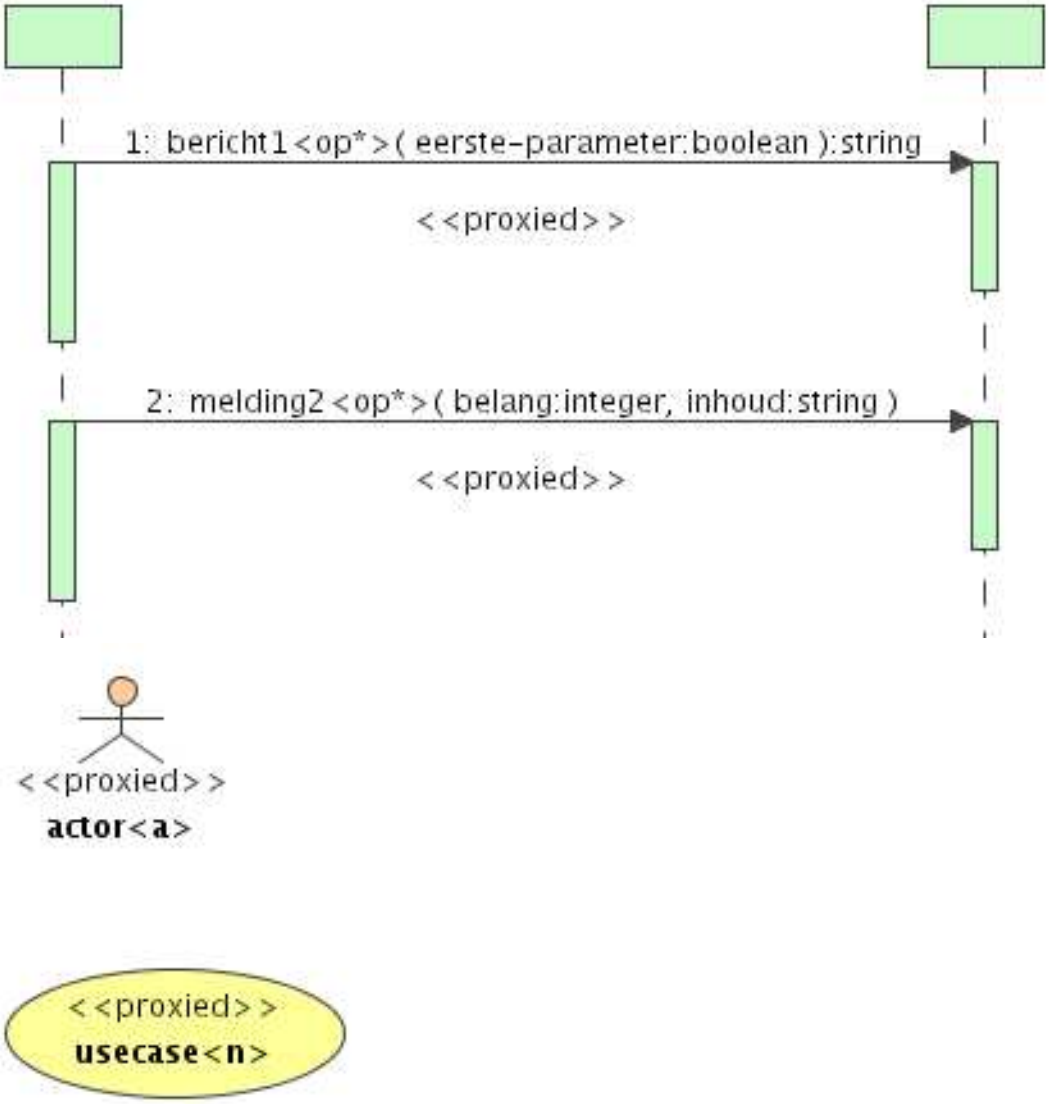
  ```
  actuele-naam<formele-naam>(P):T
  ```

  Dit formaat is strikt, er mogen geen spaties en dergelijke tussen de ')' en ':' zitten. *P* bevat de eventuele parameters van de te genereren operatie. Deze kunnen zelf ook types hebben, waarbij een ':' de namen van de types scheidt. De parameters zelf worden door komma's gescheiden. *T* is het eventuele type van de te genereren operatie. Alle types dienen de naam van een in het model bestaand DataType te zijn. *P* en *T* dienen geen formeel gedeelte te bevatten, dus vrij te zijn van formele namen tussen '<' en '>'.

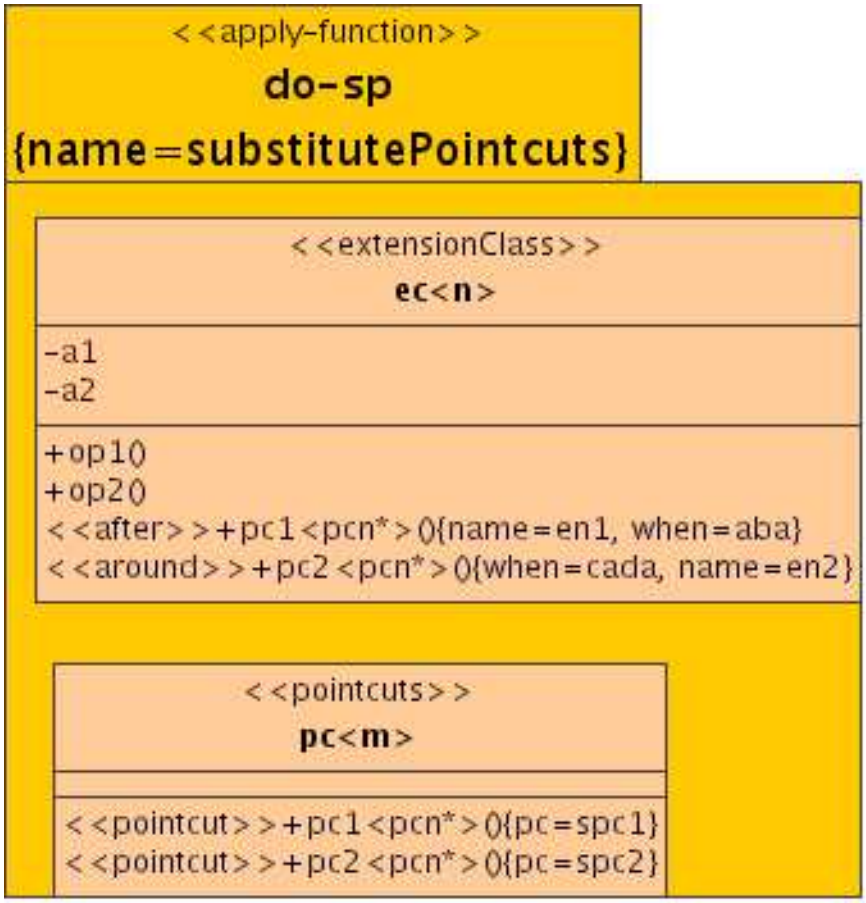
  In figuur 3 staat een voorbeeld van een functie die sequentieberichten in operaties omzet: de functie *sd_to_sysint*. In dit figuur verwijst een object door middel van de tag 'ref' die voor dat object gedefinieerd is naar een element <op*>. Dit element <op*> is een verzameling van sequentieberichten. Deze berichten zijn te zien in figuur 4. Zoals kan worden afgeleid uit figuur 10 worden er twee operaties gegenereerd:

  - `bericht1(eerste-parameter:boolean):string`
  - `melding2(belang:integer, inhoud:string)`

- Voor AOSD kunnen structurele pointcuts in extensieklassen worden geïntegreerd. Hiervoor dient er een klasse met het stereotype <<pointcuts>> te worden gedefinieerd. Deze klasse be-



Figuur 4: Aanvullende elementen voor de toepassing van de functie *sd_to_sysint*



Figuur 5: Een toepassing van structurele pointcuts en extensieklassen

vat de te integreren pointcuts als operaties. Deze operaties, met het stereotype <<pointcut>>, hebben een tag 'pc' dat het structurele pointcut aangeeft.

Een voorbeeld hiervan is de functietoepassing $do - sp$ uit aosd.xml welke te zien is in figuur 5. De klasse pc<m> heeft het stereotype <<pointcuts>> en twee operaties pc1 en pc2. Beide operaties hebben het stereotype <<pointcut>>. De operatie pc1 heeft een tag pc met de waarde spc1, de operatie pc2 heeft een tag pc met de waarde spc2.

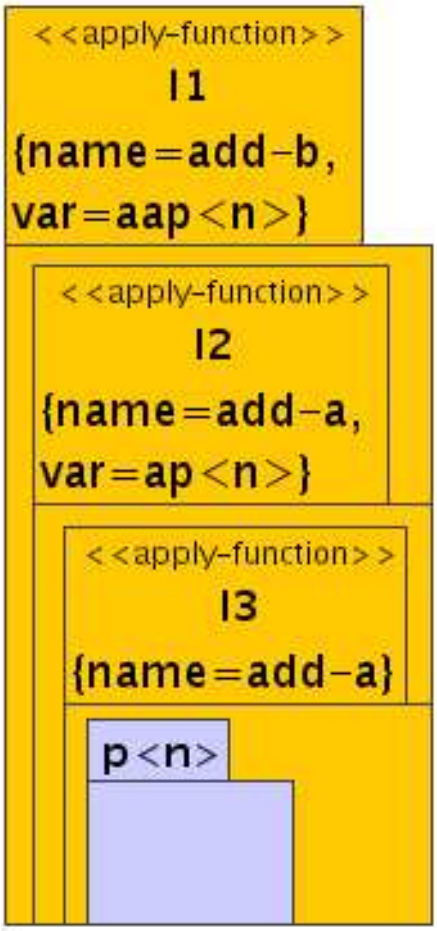
In de extensieklasse worden de te bewerken operaties voorzien van het stereotype *B*, waarbij *B* één van de stereotypen <<after>>, <<afterReturning>>, <<around>>, of <<before>> is. *B* heeft twee tags:

- 'name', dat de naam van de resulterende operatie aangeeft. In het voorbeeld zijn dit en1 van pc1 en en2 van pc2.
- 'when', dat het dynamische pointcut aangeeft. In het voorbeeld zijn dit aba van pc1 en cada van pc2.

Het resultaat is een operatie met als naam de inhoud van tag 'name'. Verder heeft deze operatie een stereotype *B* met de tags 'when' en 'struct', welke de inhoud van respectievelijk bovenstaande tags 'when' en 'pc' bevatten.

Voor het voorbeeld in figuur 5 worden de volgende operaties gegenereerd:

- `<<after>> d3e437__en1(){struct=spc1, when=aba}`
- `<<after>> d3e447__en2(){struct=spc2, when=cada}`



Figuur 6: Geneste functietoepassingen

Hierbij zijn de voorvoegsels d4e437_ en d4e447_ onwillekeurig gegenereerd door TPUPT. Dit is het resultaat van het teken '^' dat voor de operatienaam <extname> van de klasse <n>[1..-1] van de functie *substitutePointcuts* staat. Deze functie is afgebeeld in figuur 9.

3. Het is mogelijk om een functie binnen andere functietoepassingen toe te passen (i.e. nesten). Een voorbeeld hiervan is te zien in figuur 6. Voor geneste functietoepassingen geldt het volgende:

   - De geneste functietoepassing is een normale functietoepassing, dus een pakket met het stereotype <<apply-function>> en een tag 'name'. De functietoepassing $l3$ maar ook $l2$ in figuur 6 zijn hier een voorbeeld van.
   - De functietoepassing die de geneste functietoepassing bevat is een normale functietoepassing met een extra tag 'var'. Deze tag dient om de actuele parameters aan te duiden welke het resultaat zijn van de geneste functietoepassing. Het formaat van deze tag per actuele parameter is $actueel < formeel >$. Indien de tag meerdere actuele parameters bevat worden de verschillende delen gescheiden door spaties (dus '$actueel1 < formeel1 > \quad actueel2 < formeel2 > \ldots$').

     De functietoepassingen $l2$ en $l1$ in figuur 6 zijn hier een voorbeeld van. De functietoepassing $l3$ genereert een pakket ap en de functie $add - a$ heeft een formele parameter <n>, dus heeft de tag 'var' in de functietoepassing $l2$ de waarde ap<n>. De functietoepassing $l2$ genereert een pakket aap, dus heeft de tag 'var' in de functietoepassing $l1$ de waarde aap<n>.

5 Zelf functies definiëren

Het is mogelijk om nieuwe functies te definiëren. Als voorbeeld is hier de functie *make_csa* uit student.xml genomen. Deze functie is afgebeeld in de figuren 7 en 8. Hiervoor dient de volgende procedure te worden uitgevoerd:

1. Ergens in het model dient een nieuw pakket met het stereotype <<function>> aangemaakt te worden. De functie dient een naam te hebben, omdat deze anders nergens kan worden toegepast.

2. De inhoud van het pakket beschrijft de functie. Deze inhoud bestaat uit drie soorten elementen:

   - Formele parameters. Dit zijn elementen die geen uitgaande dependencies hebben en waarvan de naam tussen '<' en '>' staat. Formele parameters zelf komen nooit in het actuele resultaat (dat van de tweede fase) terecht. Deze elementen hebben slechts als doel om een aanknopingspunt voor de actuele parameters te geven.

     In figuur 7 zijn dit de interfaces <bi*> en <si*>, de klasse <btm_core*>, en de proxy naar <sds*>. Verder is de classifier-rol <sds*> in figuur 8 een formele parameter.

   - Resulterende elementen. Dit zijn de elementen die het resultaat van de functietoepassing vormen. Alle soorten UML-elementen kunnen als resulterende elementen dienen.

     Indien de naam van de actuele parameter in de naam van het actuele resulterende element zichtbaar dient te zijn, kan dit worden aangeven door de formele naam (inclusief '<' en '>') in de naam van dit element op te nemen. De naam van de actuele parameter kan gedeeltelijk worden opgenomen door er $[a..b]$ achter te zetten. Hierbij geeft a de beginpositie en b de eindpositie binnen de naam aan, waarbij een negatief getal aangeeft dat er van het einde dient te worden teruggeteld.

     In figuur 7 zijn de interfaces <bi> en <si>, en de klassen <btm_core>, <sds>, en <bi>[2..-4]Mgr resulterende elementen. De laatstgenoemde klasse is tevens een voorbeeld van een resulterend element dat een gedeelte van de naam van de actuele parameter opneemt: het eerste en de laatste 3 tekens van die naam worden genegeerd. Ook zijn de associaties tussen de klassen <bi>, <bi>[2..-4]Mgr, en <btm_core>, de associatie tussen de klasse <sds> en het interface <si>, en de <<use>> dependency tussen de klasse <bi> en <sds> voorbeelden van resulterende elementen.

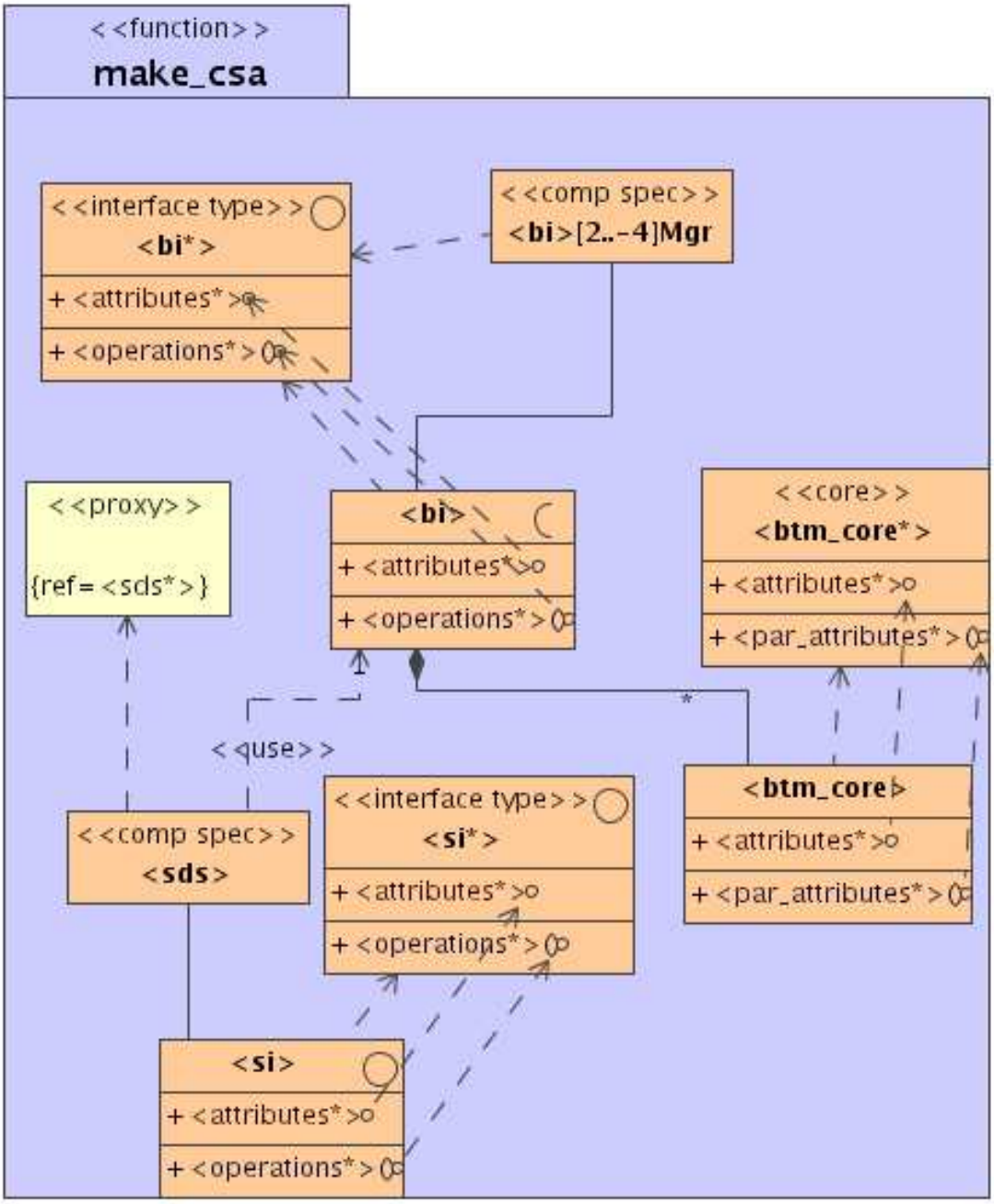
   - Overgangselementen. Overgangselementen geven aan welke formele parameters in welke formeel resulterende elementen worden omgezet. Alleen UML-dependencies kunnen als overgangselement dienen. De dependencies wijzen van de formeel resulterende elementen naar de bijhorende formele parameters. Ze hebben geen speciale stereotypen.

   Relationele elementen zijn elementen die een relatie tussen twee andere UML-elementen aangeven. Voorbeelden van dit soort elementen zijn associaties, dependencies, en aggregaties. Deze elementen kunnen zowel als formele parameters als resulterende elementen als overgangselementen dienen:

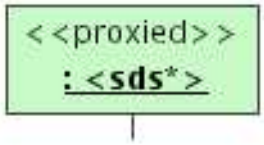
   - Het zijn formele parameters indien de twee elementen waarmee ze verbonden zijn beide een formele parameter zijn. Hier wordt in de tweede fase echter niets mee gedaan.
   - Het zijn formeel resulterende elementen indien de twee elementen waarmee ze verbonden zijn beide resulterende elementen zijn.
   - In alle overige gevallen zijn het overgangselementen.

3. Voor de overige functionaliteit dienen de beschrijvingen in de deelstappen van stap 2 van sectie 4 in acht te worden genomen, waarbij het volgende opgemerkt dient te worden:

   - Voor formeel resulterende AOSD-operaties geldt dat indien de naam met het teken '^' begint, dat dit teken wordt vervangen door een unieke string welke eindigt op '_'. Voor de definitie van AOSD-operaties geldt verder dat het stereotype <<behavioral>> wordt gebruikt als formele tegenhanger van de actuele stereotypen <<after>>, <<afterReturning>>, <<around>>, en



Figuur 7: De functie *make_csa*



Figuur 8: Aanvullende elementen voor de functie *make_csa*

<<before>>. Dit stereotype kan zowel in de formele parameters als in de formeel resulterende elementen gebruikt worden.

Voorbeelden van toepassingen van het stereotype <<behavioral>> zijn te zien in figuur 9. Dit figuur laat de functie *substitutePointcuts* uit aosd.xml zien met de operatie <pcn*> in de klasse <n> en de operatie ^<extname> in de klasse <n>[1..-1]. De [1..-1] maakt deel uit van de resulterende naam om te voorkomen dat er twee klassen <n> in de functie staan.

- Voor de definitie van berichten in sequentiediagrammen geldt dat de naam van het formele sequentiebericht als volgt is opgebouwd:

  `<formele-naam*>(<P?>):<T?>`

  Hierbij is P een parameter die de formele parameterverzameling voorstelt, en T het formele datatype. Beide dienen in het formele sequentiebericht te zitten, maar beide zijn optioneel in de formeel resulterende operatie. Het formaat van de formeel resulterende operatie bepaalt het formaat van de actueel resulterende operatie.

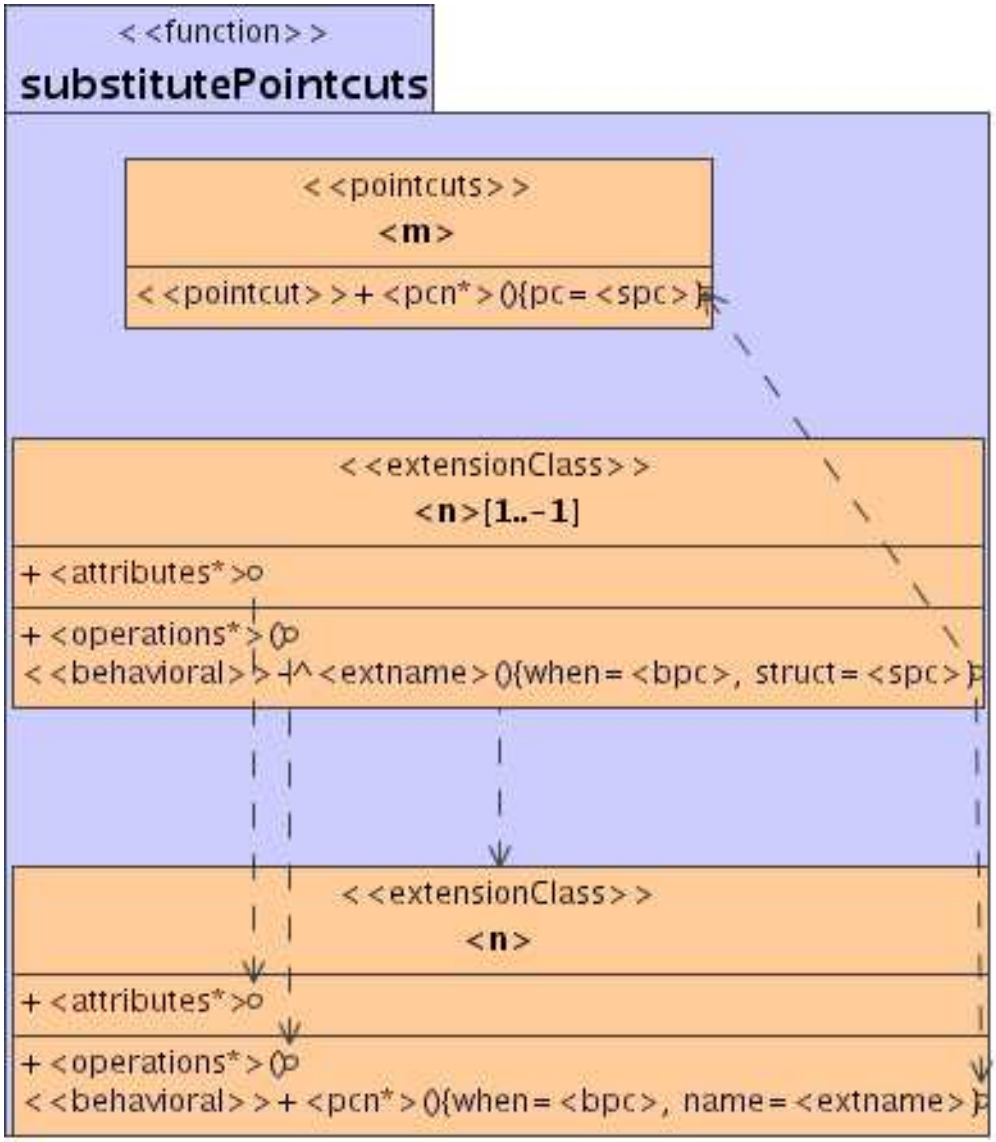
  In het voorbeeld in de figuren 10 en 11 wordt P voorgesteld door 'parameter-set' en T door 't'. De formeel resulterende operatie bevat zowel P als T, dus bevatten de actueel resulterende operaties zowel alle opgegeven parameters als alle opgegeven types.

6 Overzicht van de gebruikte stereotypen

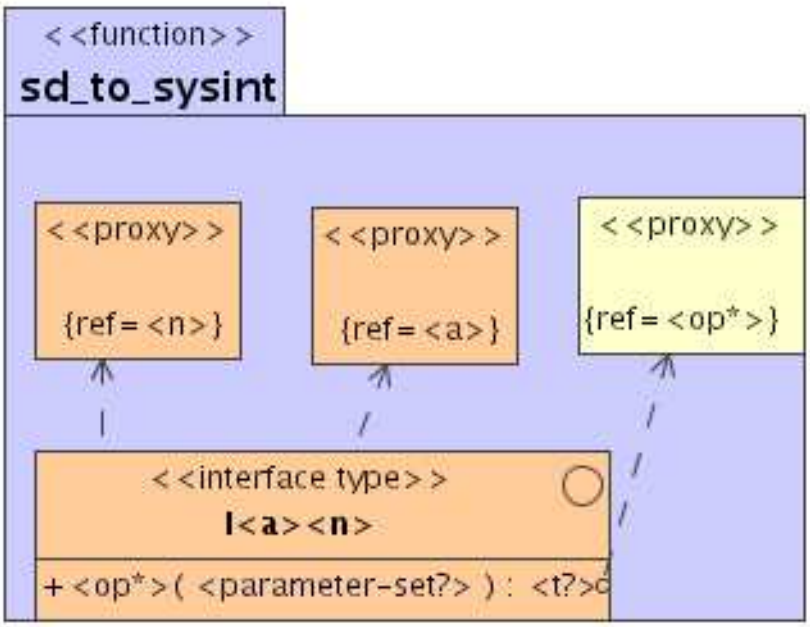
Deze sectie geeft een overzicht van de gebruikte stereotypen en de tags die ze definiëren. Voor elk stereotype wordt verder opgegeven wat het basiselement is en of het een formeel (stap 1) of actueel (stap 2) stereotype is. Voor het stereotype <<behavioral>> geldt dus dat het basiselement een operatie is en dat het in stap 1 gebruikt dient te worden. Voor elke tag wordt het type en de verplichtheid gegeven. Er kunnen uiteraard ook zelfgedefinieerde stereotypen worden gebruikt in de functies.

De stereotypen zijn:

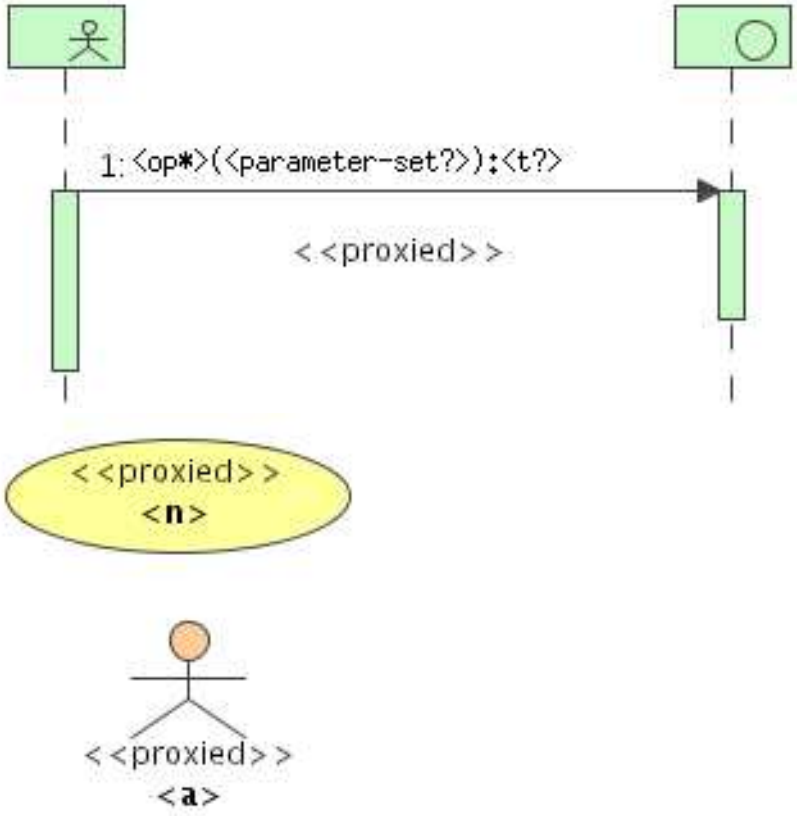
- <<behavioral>> (operatie, formeel)
  Dit stereotype geeft aan dat de actuele tegenhanger van deze operatie een van de stereotypen <<after>>, <<afterReturning>>, <<around>>, of <<before>> dient te bevatten.
  - name : string[1] (verplicht)
    de naam van de resulterende operatie.
  - struct : string[1] (resulterend, verplicht)
    de naam van het resulterende structurele pointcut.
  - when : string[1] (verplicht)
    de naam van het resulterende gedrags-pointcut.
- <<after>>, <<afterReturning>>, <<around>>, <<before>> (operatie, actueel)
  Deze stereotypen specificeren wanneer de operatie tijdens het draaien van het programma dat uit het model volgt dient te worden uitgevoerd.
  - name : string[1] (verplicht)
    de naam van de resulterende operatie.
  - struct : string[1] (resulterend)
    de naam van het resulterende structurele pointcut. Deze tag dient niet bij de invoer te worden opgegeven.
  - when : string[1] (verplicht)
    de naam van het resulterende gedrags-pointcut.
- <<apply-function>> (pakket, actueel)
  Dit stereotype geeft aan dat er een functiedefinitie wordt toegepast op de inhoud.



Figuur 9: De functie *substitutePointcuts*



Figuur 10: De functie *sd_to_sysint*



Figuur 11: Aanvullende elementen voor de functie *sd_to_sysint*

  - name : string[1] (verplicht)
    de naam van de toe te passen functie.
  - var : string[1] (optioneel)
    de door spaties gescheiden namen van de resulterende elementen van de geneste functietoepassing. Het formaat van elke naam is $actueel < formeel >$.

- <<function>> (pakket, formeel)
  Dit stereotype geeft aan dat dit pakket een functiedefinitie is.

- <<extensionClass>> (klasse, formeel/actueel)
  Dit stereotype geeft aan dat deze klasse een extensieklasse is en waarschijnlijk gedragsoperaties bevat.

- <<pointcut>> (operatie, formeel/actueel)
  Dit stereotype geeft aan dat deze operatie een pointcut is.
  - pc : string[1] (verplicht)
    de naam van het structurele pointcut.

- <<pointcuts>> (klasse, formeel/actueel)
  Dit stereotype geeft aan dat deze klasse pointcut-operaties bevat.

- <<proxy>> (elk element, formeel/actueel)
  Dit stereotype geeft aan dat dit element een tussenelement is. Het echte element wordt met het stereotype <<proxied>> aangeduidt.
  - ref : string[1] (verplicht)
    de naam van het <<proxied>> element.

- <<proxied>> (elk element, formeel/actueel)
  Dit stereotype geeft het element aan waar een tussenelement naar verwijst.

7 Mogelijke uitbreidingen

TPUPT bevat momenteel de volgende tekortkomingen, waarvan sommigen ook als uitbreiding gezien kunnen worden:

- Indien er functies op een UML-model worden toegepast, dienen ze in hetzelfde bestand als waarin het model staat te zijn gedefinieerd. TPUPT zal niet werken wanneer dit niet het geval is, omdat UML-bewerkprogramma's identificatienummers aan modelelementen toekennen die per modelbestand verschillen.

- Indien er zich in het model een element bevindt dat meerdere malen geïnstantieerd kan worden en dit element meerdere formeel resulterende elementen heeft die onderling gerelateerd zijn, dan is de manier waarop de actuele resulterende elementen gerelateerd zijn ongedefinieerd.

- Modellen die met TPUPT zijn getransformeerd bevatten corrupte informatie over de plaatjes, wat in ArcStyler leidt tot een reeks negeerbare waarschuwingen. De corruptie wordt veroorzaakt doordat de plaatjes in gedeelten worden opgeslagen die specifiek zijn voor elk UML-programma. TPUPT houdt zich niet met deze gedeelten bezig.

- Het toepassen en definiëren van functies dient met enige zorgvuldigheid te gebeuren, aangezien TPUPT weinig controles met betrekking tot de correctheid van de invoer uitvoert.

Referenties

[1] Welcome to Interactive Objects –Interactive Objects.
http:///www.interactive-objects.com/.

[2] John Cheesman and John Daniels. *UML Components –A Simple Process for Specifying Component-Based Software*. Addison-Wesley, 2001. ISBN 0-201-70851-5.

[3] Ivar Jacobson and Pan-Wei Ng. *Aspect-Oriented Software Development with Use Cases*. Pearson Education Inc., 2005. ISBN 0-321-26888-1.

[4] OMG. *Unified Modeling Language Specification 1.4*, September 2001.
http://www.omg.org/docs/formal/01-09-67.pdf.

[5] OMG. *XML Metadata Interchange (XMI) Specification 1.2*, January 2002.
http://www.omg.org/cgi-bin/apps/doc?formal/02-01-01.pdf.

[6] The SAXON XSLT and XQuery Processor.
http://saxon.sourceforge.net/.

[7] Altova XML Spy –XML Editor, XSLT / XQuery Debugger, XML Schema / WSDL Designer, SOAP Debugger.
http://www.altova.com/products/xmlspy/xml_editor.html.