UML-kielen formalisointi Object-Z:lla

Miika Nurminen (minurmin@cc.jyu.fi)

2. toukokuuta 2003

Tiivistelmä

Seminaarityössä tutkitaan UML-kielen formalisointimahdollisuuksia käyttäen Object-Z -kuvauskieltä. UML-kielestä käytään läpi määrittelyperiaatteet ja puoliformaaliuden aiheuttamia ongelmia, Object-Z:n notaatio esitellään lyhyesti. UML-kielestä keskitytään luokkakaavioiden formalisointiin, mutta myös muita muunnoksia UML-kielen osien ja Object-Z -kielen välillä käydään läpi.

1 Johdanto

UML-kielestä (*Unified Modelling Language*) on tullut *de facto*-standardi ohjelmistojärjestelmien kuvaamiseen. UML-notaatiolla piirretyt kaaviot ovat havainnollisia, mutta niiden analysointia ja koodin generointia hankaloittaa kielen puoliformaalius. Notaation syntaksi on määritelty täsmällisesti, mutta semantiikka on kuvattu vain luonnollisella kielellä.

Seminaarityössä käsitellään mahdollisuuksia UML:n formalisointiin Object-Z -kuvauskielen avulla. Object-Z on Z-notaation pohjalta kehitetty oliopohjainen formaali kuvauskieli. Aiheen laajuuden takia UML-notaatiosta keskitytään lähinnä luokkakaavioihin.

Seminaarityö jakautuu seuraavasti: luvussa 2 kuvataan UML-kieltä, sen määrittelyä ja puoliformaaliudesta seuraavia epäselvyyksiä. Luvussa 2 esitellään Object-Z -kieli Z-notaation pohjalta. Luvussa 4 pohditaan UML:n formalisointimahdollisuuksia.

Oletan, että lukija tuntee UML-kielen alkeet ja Z-notaation perusteet. Myös predikaattilogiikan perusteiden osaamisesta on hyötyä.

2 UML-kielestä

UML on OMG[1]-organisaation standardoima kieli ohjelmistojärjestelmien, liiketoimintamallien ja muiden järjestelmien spesifiointiin, visualisointiin, muodostamiseen ja dokumentointiin.

[1] http://www.omg.org/

UML-kieli määrittää 8 kaaviota, jotka ovat malleja kehitettävästä järjestelmästä. Eri kaaviot kuvaavat järjestelmää eri näkökulmista ja eri abstraktiotasoilla. UML-kielen määrittelevä malli pyrkii yhdistämään nämä näkökulmat. Kaaviotyypit ovat seuraavat:

- Yleiset: käyttötapauskaavio ja luokkakaavio.
- Käyttäytymiseen liittyvät: tilakaavio, aktiviteettikaavio, sekvenssikaavio ja yhteistoimintakaavio.
- Toteutukseen liittyvät: komponenttikaavio ja sijoituskaavio.

Näistä käyttötapauskaaviot ja luokkakaaviot lienevät tunnetuimpia. Käyttäytymistä kuvaavat tila- ja aktiviteettikaaviot perustuvat David Harelin tilakaavionotaatioon, joka puolestaan on koostetuilla tiloilla ja rinnakkaisuuden tuella varustettu yleistys äärellisille automaateille. Sekvenssi- ja yhteistoimintakaavioilla kuvataan järjestelmän olioiden välistä vuorovaikutusta. Komponentti- ja sijoituskaavioilla kuvataan järjestelmän toteutukseen liittyviä fyysisiä ja loogisia rakenteita. Jatkossa keskitytään ensisijaisesti luokkakaavioihin. [8]

2.1 UML-kielen määrittely

UML:n kunnianhimoiset tavoitteet ovat seuraavat [8]:

- Tarjota käyttökelpoinen ja ilmaisuvoimainen kieli merkityksellisten mallien kehittämiseen ja vaihtamiseen.
- Sisällyttää kieleen laajennus- ja erikoistamismekanismit.
- Tukea spesifikaatioita, jotka ovat riippumattomia tietystä ohjelmointikielestä tai kehitysprosessista.
- Tarjota muodollinen perusta mallinnuskielen ymmärtämiseen. Kielen pitää olla sekä täsmällinen että helposti lähestyttävä.
- Kasvattaa oliotekniikkaa hyödyntävien työkalujen markkinoita.
- Tukea korkeamman abstraktiotason käsitteitä ohjelmistokehityksessä (esim. komponentit, sovelluskehykset ja suunnittelumallit).
- Integroida alan parhaat käytännöt.

UML:n tavoitteiden onnistumista voi kritisoida monella (tämän kirjoitelman aihealueen ulkopuolelle menevällä) tavalla. Formaalien menetelmien käytön kannalta UML-kielen ongelma on semantiikan epätäsmällisyys.

UML-kieli määritellään nelikerroksisen metamallin avulla. Sen kerrokset on lueteltu taulukossa 1. Meta-metamalli muodostaa perustan arkkitehtuurin metamallinnukselle. Sen pääasiallinen tehtävä on määrittää kieli, jota metamalli käyttää. UML:n meta-metamalli on yhteinen toisen OMG:n kehittämän määrityksen, MOF:n (Meta Object Facility) ohella.

Metamallia käytetään varsinaisen UML-kielen määrittelyyn. UML-kielen abstrakti syntaksi määritellään luokkakaavioiden avulla, joissa (meta)luokkina ovat varsinaisen mallikerroksen

Kerros	Kuvaus	Esim.
Meta-metamalli	Infrastruktuuri metamallinnusarkkitehtuurille. Määrittelee kielen metamalleille.	Metaluokka, Meta-attribuutti, Metaoperaatio
Metamalli	Meta-metamallin ilmentymä. Määrittelee kielen mallin spesifiointiin.	Luokka, Attribuutti, Operaatio, Komponentti
Malli	Metamallin ilmentymä. Määrittelee kielen kohdealueen kuvaukseen.	Jäsen, haeNimellä(), Web-palvelin.
Käyttäjän oliot	Mallin ilmentymä. Määrittelee sovellusalueen.	`<'Aku Ankka','Ankkalinna'>`, `NULL`, Apache

Taulukko 1: UML-metamallikerrokset.

oliot. Luokkakaavioiden metamallinnuskäytön takia niiden formalisointi on välttämätöntä UML-kielen laajempaa formalisointia ajatellen.

Luokkakaavioiden ohella UML-kielen staattinen semantiikka (ns. hyvinmuodostetut kaavat) määritellään OCL (*Object Constraint Language*) -kielen avulla. OCL-kieli tyypitetty lausekekieli, jota käytetään invarianttien rajoitteiden määrittämiseen. Invariantteja voidaan soveltaa mm. luokkamallissa, stereotyyppien tyyppirajoitteissa, tilakaavion tilasiirtymävahtien määrityksessä ja annettaessa esi- ja jälkiehtoja operaatioille [8]. OCL lisää UML:n ilmaisukykyä, mutta sen semantiikassa on UML:n perustan tapaan epätäsmällisyyksiä [4]. UML:n dynaaminen semantiikka (mallinnuselementtien merkitys) määritellään luonnollisella kielellä.

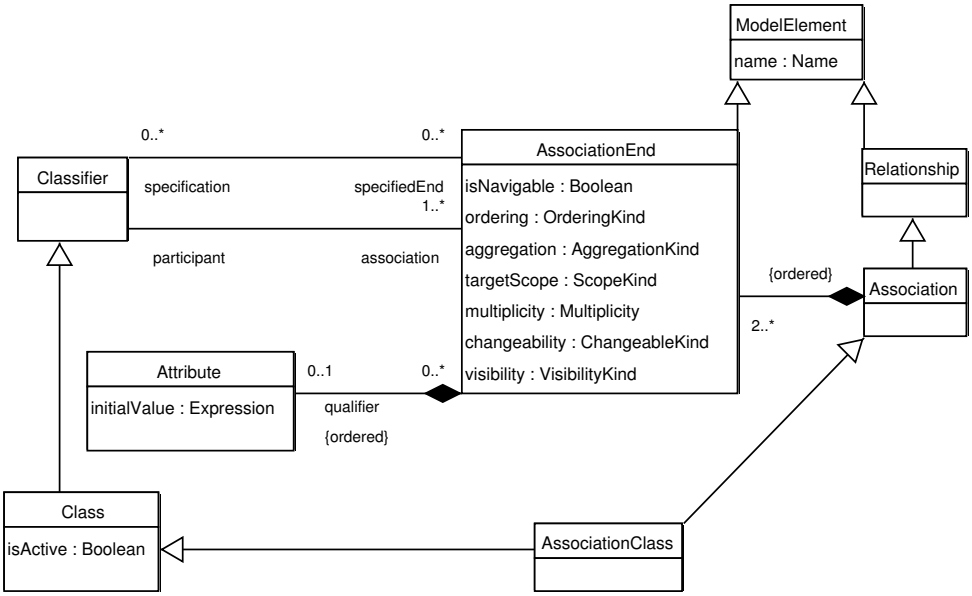
UML-kieleen on sisällytetty standardielementtien lisäksi laajennusmekanismi, jolla voidaan määritellä uusia elementtejä. UML-kieltä voi laajentaa stereotyypeillä, rajoitteilla tai kiinnitetyillä arvoilla (*tagged values*). Stereotyypit ovat loogisesti metaluokkia, jotka on peritty jostain metamallin luokasta. Näin olemassaolevia mallielementteja voidaan erikoistaa ja määritellä esimerkiksi tiettyyn ohjelmointikieleen suoraan soveltuva elementti tai mukauttaa jonkin elementin ulkoasua. Rajoitteet ovat OCL-kielellä ilmaistavia määrityksiä mallielementille (esim. järjestetty attribuuttijoukko) ja kiinnitetyt arvot ovat mallielementtiin liitettäviä vakioarvoja.

2.2 Puoliformaaliuden ongelmia

UML:n puoliformaalius aiheuttaa monia ongelmia työkalujen toiminnassa liittyen koodin generointiin tai mallin muodostamiseen olemassaolevan koodin pohjalta. Edelleen järjestelmän mallin analysointi tai päätelmien teko eivät ole vakaalla pohjalla. Epätäsmällisesti määritellyt mallit voivat helpottaa järjestelmän hahmottelua, mutta voivat toteutusvaiheessa aiheuttaa hankalasti havaittavia virheitä. Asiakkaat, projektin johto ja kehittäjät saattavat ymmärtävät moniselitteistä mallia eri tavoin.

Stevens [6] ottaa esimerkiksi UML:n moniselitteisyyksistä assosiaation käsitteen. Assosiaatio määritellään luokkien väliseksi yhteydeksi. Assosiaatio voi olla yksi- tai kaksisuuntainen ja siihen osallistuville olioille voi määritellä lukumäärärajoitteet eli kardinaliteetit. Assosiaatiolle

voi määritellä myös erityisen assosiaatioluokan. Assosiaatio osana UML:n metamallia on esitetty kuvassa 1.



Kuva 1: Assosiaatio UML:n metamallin osana.

Luokkien välille määritellyn assosiaation merkitystä voi tarkastella eri näkökulmista. Ne ovat samalla vaihtoehtoisia tulkintoja, joihin UML-määritys ei suoraan ota kantaa. Assosiaatio voi olla *staattinen* tai *dynaaminen*. Staattinen assosiaatio on rakenteellinen yhteys, joka voidaan toteuttaa esim. luokan attribuuttina. Dynaaminen assosiaatio on käyttäytymisyhteys, joka voidaan tulkita esim. metodin parametrina. Assosiaatioita voi tarkastella myös oliomallin kannalta, jolloin täytyy ottaa huomioon olioiden mahdollisesti ajonaikana muuttuvat viittaukset muihin olioihin.

Toinen assosiaation käsitteeseen liittyvä epäselvyys on, tulkitaanko assosiaatio monikkona, joka sisältää viitteet assosioitaviin olioihin, vai onko assosiaatio mallinnuselementti, jolla on oma identiteetti. Ainakin assosiaatio*luokka* on selvästi mallinnuselementti.

Kim ja Carrington [4] huomauttavat, ettei myöskään UML:n rajoitekieli OCL ole täsmällinen. Sillä ei ole formaalisti määriteltyä semantiikkaa eikä yksikäsitteistä kuvausta abstraktia syntaksia kuvaaviin luokkakaavioihin. Lisäksi samojen asioiden määrittely kolmesta eri näkökulmasta (metaluokat, OCL-määritykset ja sanallinen semantiikka) aiheuttaa määrittelyyn päällekkäisyyksiä ja moniselitteisyyksiä.

3 Johdatus Object-Z -kuvauskieleen

Object-Z on Queenslandin yliopistossa [2] kehitetty oliopohjainen laajennus Z-kuvauskieleen. Kieli hyödyntää Z:n syntaksia ja lisää notaation ja semantiikan oliopohjaisille käsitteille. Spesifikaatiot tehdään edelleen skeemoilla ja invarianteilla, mutta ne on kapseloitu luokan sisälle.

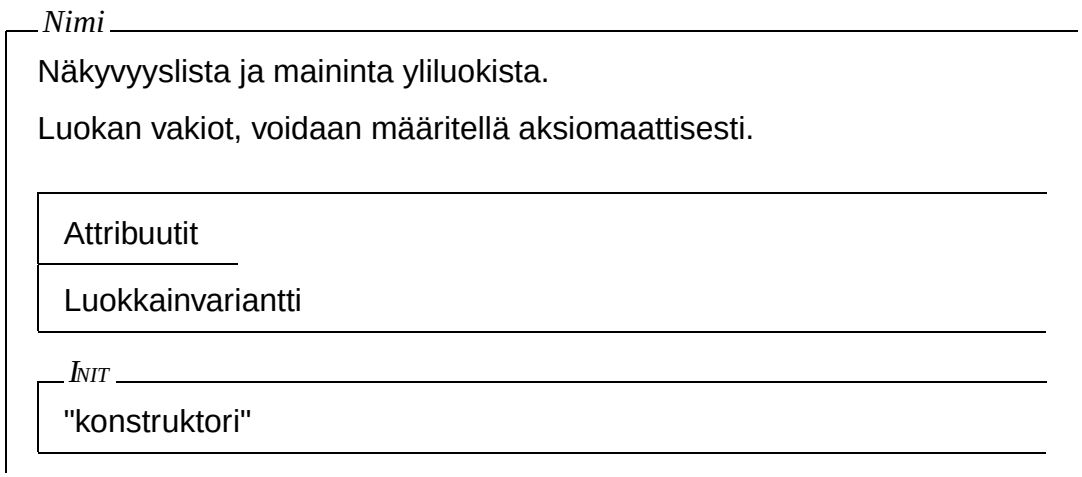
Object-Z:n sisältämistä symboleista mainittakoon sisällytettyä attribuuttia (*containment*) ilmaiseva ©-symboli, skeemojen näkyvyyksiä perinnässä määrittävä projektiosymboli ↾, Lotos-metodista [2] tuttu valintaoperaattori [] (*angelic choice*) ja B-metodissa käytty rinnakkaisoperaattori ||. Lisäksi temporaalilogiikan symbolit □ ja ◇ ovat käytössä ns. historiainvarianttien määrityksessä.

Duke & al. [1] pitävät Object-Z:n merkittävänä sovellusalueena standardien kuvaamista. Nykyisten standardien monimutkaisuudesta johtuen rakenteiset kuvaukset ovat olennaisia. Object-Z:n olio-ominaisuudet ja kielen formaalius helpottavat standardin käsitteiden määrittelyä ja päättelyä. Esimerkkinä he kuvaavat osan hajautetuissa järjestelmissä käytettävästä ODP Trader -standardista.

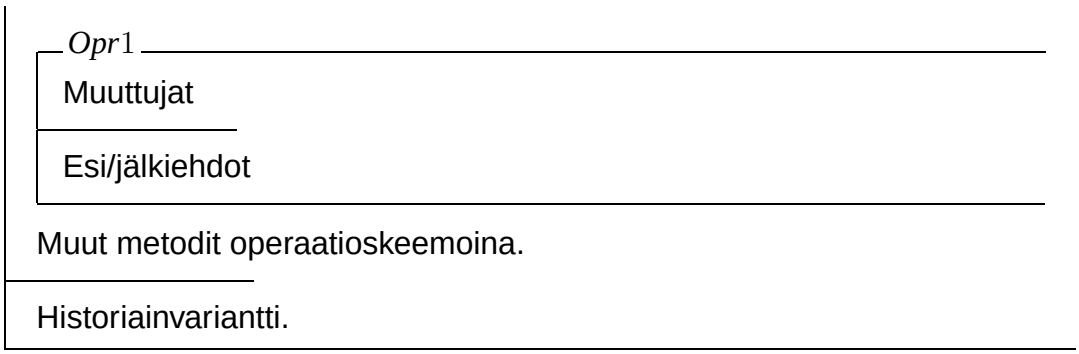
3.1 Z-skeemoista Object-Z -luokkiin

Z-notaatiossa järjestelmä määritellään joukkona tila- ja operaatioskeemoja. Jos halutaan päätellä, mitkä operaatiot vaikuttavat tiettyyn tilaskeemaan täytyy kaikkien operaatioiden kuvaukset käydä läpi. Toisaalta, Object-Z:ssa yksittäiset operaatiot vaikuttavat vain yhteen tilaskeemaan. Tämä tilaskeema ja siihen liittyvät operaatiot muodostavat luokan. Luokka toimii myös tyyppinä, jolloin tietyn luokan oliot voivat viitata muiden luokkien olioihin.

Object-Z:n luokan määritys on yleiseltä rakenteeltaan seuraavanlainen:

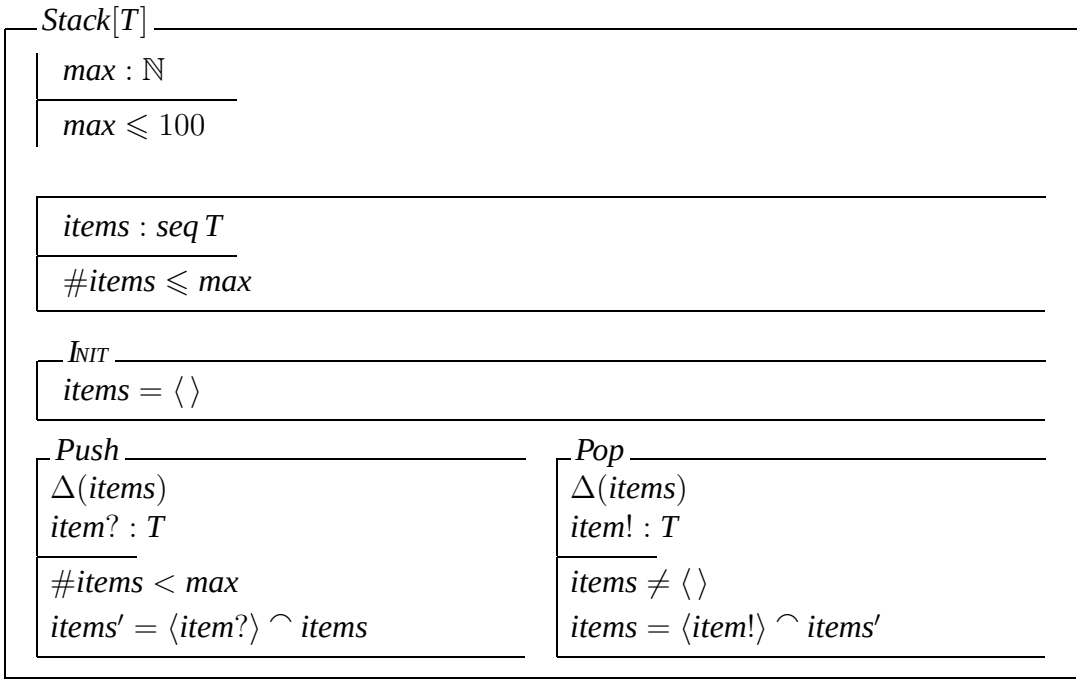


[2] http://www.uq.edu.au/



Object-Z tukee oliotermein ilmaistuna tiedon kapselointia ja polymorfismia. Operaatioiden kuormitusta vastaavaa mekanismia ei ole käytössä, esimerkiksi luokan oliot luodaan Init-merkityn skeeman ehdoilla. Polymorfista käyttäytymistä varten luokan attribuutti voidaan määritellä viitearvoiseksi $\downarrow$-symbolilla. Olioiden kompositiota varten on määritetty sisältymismekanismi (*containment*). Objektiin sisällytettyä attribuuttia merkitään $\copyright$-symbolilla. Sisällytetty olio ei voi olla missään muussa oliossa.

Yksinkertaisena esimerkkinä Object-Z:n luokkien käytöstä määritellään pinoa kuvaava geneerinen luokka Stack:



Pinoon on määritelty vakio max, joka kuvastaa pinon maksimikokoa. Eri pino-olioilla voi olla eri max-arvo, joka on kullakin ilmentymällä vakio (mutta aina enintään 100). Tilaskeemassa määritellään attribuutti items, joka sisältää pinon alkiot. Uudella pino-oliolla ei ole alkioita.

Operaatioiden määrittelyssä käytetään Z:sta tuttua konventiota merkitä syötemuuttujia ?:llä ja paluumuuttujia !:lla. Muutettavaa attribuuttia merkitään Δ-etumerkillä.

Esimerkkinä perinnästä määritellään vielä jonoa kuvaava geneerinen luokka Queue:

Queue[*T*]

$\upharpoonright(max, Init, Join, Leave)$

$Stack[T][Leave/Pop]$

Join

$\Delta(items)$

$item? : T$

$\#items < max$

$items' = items \frown \langle item? \rangle$

Ensimmäisen rivin projektiosymbolilla $\upharpoonright$ määritellään jonon julkiset ominaisuudet. Jos symbolia ei ole määrityksessä, oletetaan, että kaikki ominaisuudet ovat julkisia. Toisella rivillä määritetään jonon yliluokka Stack[T] sekä nimetään pinon Pop-operaatio Leave-nimiseksi. Jonon käsittelyä varten määritellään uusi operaatio Join, joka lisää uuden alkion jonon loppuun.

Huomattavaa on, että pinon Push-operaatio periytyy myös jonoon, mutta ei ole julkinen, koska sitä ei ole näkyvyyslistassa. Metodien uudelleennimeämisellä voidaan määritellä oliokielten virtuaalifunktioiden tavoin käyttäytyviä metodeja nimeämällä aliluokan metodi samalle nimelle kuin yliluokassa. Aliluokan metodi syrjäyttää yliluokan metodin. Jos yli- ja aliluokassa on samannimisiä operaatioskeemoja ilman uudelleennimeänistä, aliluokan skeema laajentaa yliluokan skeeman toimintaa.

Historiainvariantin avulla voidaan kuvata mahdollisia olion mahdollisia tapahtumasekvenssejä. Historiainvariantti on valinnainen, ja se voidaan ilmaista temporaalilogiikalla. Temporaalilogiikka on modaalinen logiikka, jossa mahdolliset maailmat kuvataan käsitteillä *aina* ($\Box$) ja *joskus* ($\Diamond$). Esimerkkinä historiainvariantin käytöstä määritellään vielä luokka FairStack:

FairStack[*T*]

$Stack[T]$

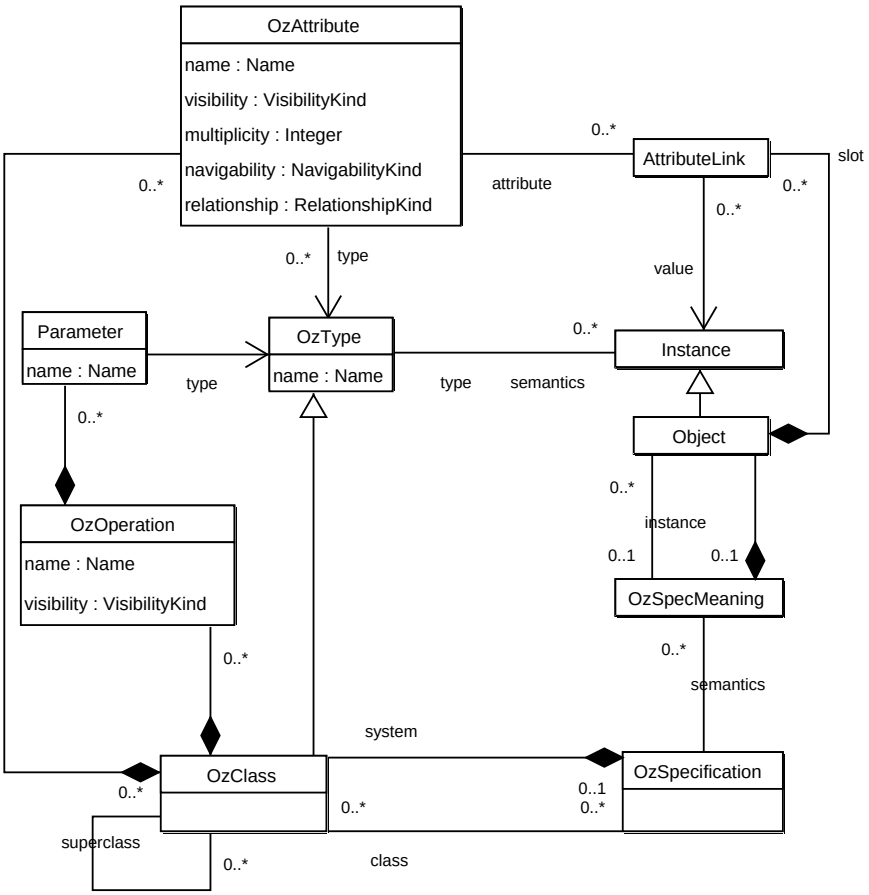
$\Box(items \neq \langle\rangle \Rightarrow \Diamond((\bigcirc \#items) < \#items))$

Historiainvariantin mukaan epätyhjälle pinolle pätee aina Pop-operaation suoritus jossain vaiheessa (eli pinon koko pienenee). $\bigcirc$-symboli merkitsee ajallista seuraamista. [1]

3.2 Object-Z:n semantiikasta

Object-Z:n semantiikka perustuu Z-kuvauskieleen, mutta oliomaisuus tuo mukanaan lisäpiirteitä, jotka monimutkaistavat sitä merkittävästi. Näitä olioihin liittyviä käsitteitä ovat esim. olion iden-

titeetti, luokan ja olion käsitteet sekä koostetut attribuutit. Object-Z:n ymmärrettävyyden helpottamiseksi Kim ja Carrington [3] esittävät Object-Z:n käsitteiden kuvaamista UML:n tapaan metamallilla, joka koostuu Object-Z:n luokista. Puoliformaali kuvaus Object-Z:n metamallista on esitetty UML-kaaviona kuvassa 2.



Kuva 2: Object-Z:n ydinkäsitteiden metamalli UML-kaaviona.

Formaali kuvaus Object-Z:n metamallista esitetään Object-Z -luokkina. Tarkastellaan esimerkkinä metaluokkaa, joka kuvaa Object-Z:n luokkakäsitteen. Määritellään aluksi yliluokka Object-Z:n tyypeille:

OzType

name : *Name*

Luokan lisäksi Object-Z:n tyyppeihin kuuluvat Z:ssa käytyt tyypit. Näitä ovat mm. luetellut tyypit, potenssijoukon tai karteesisen tulon avulla muodostetut tyypit, skeemat tai oletuksena annetut tyypit, kuten luonnollisten lukujen joukko $\mathbb{N}$. Luokan määrittelyä varten määritellään aksiomaattisesti operaatiot *directSuperclass* ja *allSuperclass*:

$$
\begin{array}{|l}
directSuperclass : OzClass \rightarrow \mathbb{P}\, OzClass \\
allSuperclass : OzClass \rightarrow \mathbb{P}\, OzClass \\
\hline
\forall\, oc : OzClass \bullet directSuperclass(oc) = oc.superclass \\
\qquad\qquad\qquad\qquad allSuperclass(oc) = directSuperclass(oc) \cup \\
\qquad\qquad\qquad\qquad\quad (\bigcup \{sco : directSuperclass(oc) \bullet allSuperclass(sco)\}) \\
\end{array}
$$

directSuperclass palauttaa tietyn Object-Z:n luokan välittömät yliluokat. *allSuperclass* palauttaa joukon, joka sisältää rekursiivisen sulkeuman kaikista luokan yliluokista. Nyt voidaan määritellä luokkaa kuvaava metaluokka OzClass (oletetaan metaluokat OzAttribute ja OzOperation määritellyiksi):

OzClass

OzType

$$
\begin{array}{|l}
superclass : \mathbb{F}\, OzClass \\
attributes : \mathbb{F}\, OzAttribute\textcircled{c} \\
operations : \mathbb{F}\, OzOperation\textcircled{c} \\
\hline
self \notin allSuperclass(self) \\
\forall\, a1, a2 : attributes \bullet a1.name = a2.name \Rightarrow a1 = a2 \\
\forall\, op1, op2 : operations \bullet op1.name = op2.name \Rightarrow op1 = op2 \\
\end{array}
$$

Attribuuteissa käytettävä $\mathbb{F}$-symboli tarkoittaa äärellistä joukkoa. Esimerkiksi luokka sisältää äärellisen määrän OzAttribute-metaluokan olioita. Luokkainvariantti kieltää syklit tyyppihierarkiassa ja varmistaa, että attribuutit ja operaatiot määräytyvät yksikäsitteisesti nimensä perusteella.

4 UML:n formalisoinnista

Formaali määrittely poistaisi UML-kielestä luvussa 2.2 mainitut ongelmat. Lisäksi UML:n käsitteiden täsmällinen tutkiminen, päättely ja todistusten tekeminen mallin pohjalta helpottuisi. Lisäksi formaali semantiikka on perusta muunnoksille muihin mallinnuskieliin. [3]

Välttämätön edellytys UML:n formalisoinnille on sen metamallin formalisointi. UML:n metamallin abstrakti syntaksi ilmaistaan UML-luokkakaavioina, joten luonnollinen perusta formalisoinnille on luokkakaavion määrittely Object-Z:lla. Tämän jälkeen muunnos luokkakaavioista

Object-Z:n luokkaskeemoiksi on suoraviivainen. Metamallin semantiikka ja tarkemmat määritykset voidaan ilmaista Object-Z:lla. [4]

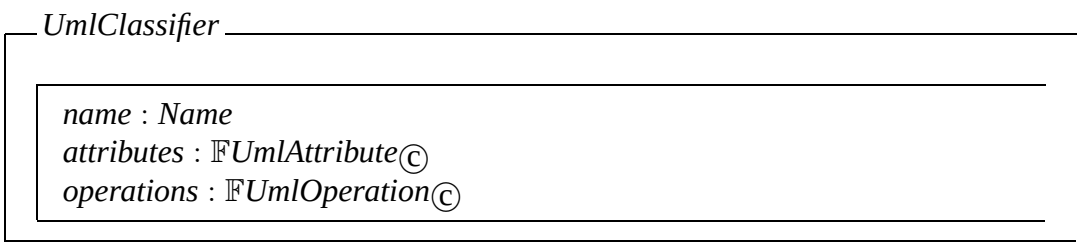
UML:n ja Object-Z:n välisistä muunnoksista on esitetty erilaisia malleja. Moreira ja Araújo [5] ovat kehittäneet menetelmän käyttötapaus- ja yhteistoimintakaavioiden formalisointiin. Sun & al. [7] ovat esittäneet XSLT-pohjaisen tekniikan Object-Z -spesifikaatioiden muuntamiseksi UML-luokkakaavioiksi. Luokkakaavioilla voidaan kuvata vain staattiset piirteet Object-Z -kuvauksesta, mutta se helpottaa formaalin mallin ymmärtämistä.

4.1 Luokkakaavion määrittely Object-Z:lla

Kimin ja Carringtonin [3] menetelmä luokkakaavion formalisointiin perustuu Object-Z:llä määritellyille muunnosoperaatioille formalisoidusta UML:n metamallista Object-Z:n metamalliin. Oman metamallin määrittely Object-Z:lle yksinkertaistaa muunnoksen määrittelyä ja tekee kielen semantiikasta ymmärrettävämmän. Esimerkki Object-Z:n metamallista luokan osalta on annettu luvussa 3.2.

UML-metamalli voidaan formalisoida määrittelemällä Object-Z -luokkaskeemat UML-määrityksessä [8] annettujen luokkakaavioiden pohjalta. UML-kielen laajuudesta johtuen formalisointi täytyy osittaa. Yksi tapa osittamiseen on edetä UML-määrityksen pakettirakenteen mukaan: luokkakaaviot kuuluvat ydinkäsitteisiin, jonka päälle määritellään käyttäytymiseen liittyvät elementit, kuten tila- ja yhteistoimintakaaviot.

Esimerkkinä luokkakaavioiden formalisoinnista määritellään Object-Z -kuvaukset luokkakaavion luokkaa kuvaavalle metaluokalle. Kimin ja Carringtonin kuvauksessa käytetty UML-metaluokkarakenne on yksinkertaistus UML-määrityksessä annetusta mallista (ks. esim. kuva 1), mutta formalisoinnin periaate on sama. Määritellään aluksi yliluokka UML-luokille ja tietotyypeille (oletetaan metaluokat UmlAttribute ja UmlOperation tunnetuiksi):



UML-luokka peritään UmlClassifier-luokasta. Luokkainvariantti määrää, että attribuutit ja operaatiot on nimetty yksikäsitteisesti.

UmlClass

UmlClassifier

$$\forall a1, a2 : attributes \bullet a1.name = a2.name \Rightarrow a1 = a2$$
$$\forall op1, op2 : operations \bullet$$
$$(op1.name = op2.name \land \#op1.parameters = \#op2.parameters \land$$
$$\forall i : 1..\#op1.parameters \bullet$$
$$op1.parameters(i).name = op2.parameters(i).name \land$$
$$op2.parameters(i).type = op2.parameters(i).type) \Rightarrow op1 = op2$$

Vastaavasti voidaan määritellä Object-Z -esitykset assosiaatioille, yleistyksille ja koko UML-luokkakaaviolle. Mallinnuselementtien formalisoinnin jälkeen voidaan määritellä muunnossäännöt UML-mallinnuselementeistä Object-Z:n luokiksi. Esimerkkinä muunnossäännöstä määritellään muunnos yksittäisten luokkien välillä. Määritellään aluksi funktio convType:

$$convType : \downarrow UmlClassifier \rightarrowtail\!\!\!\!\!\!\rightarrow \downarrow OzType$$

convType on abstrakti funktio, joka kuvaa UML-tyypin Object-Z -tyypiksi. Funktio täytyy määritellä eri tyypeille erikseen eikä sitä käsitellä tässä tarkemmin. Nyt voidaan määritellä luokkien välinen muunnosoperaatio:

$$mapUmlClassToOz : UmlClass \rightarrow \mathbb{P}\, OzClass$$

$$\forall uc : UmlClass \bullet$$
$$mapUmlClassToOz(uc) = \{oc : OzClass \mid uc.name = oc.name \land$$
$$\forall ua : uc.attributes \bullet$$
$$\exists oa : oc.attributes \bullet$$
$$oa.name = ua.name \land oa.type = convType(ua.type) \land$$
$$oa.visibility = ua.visibility \land oa.multiplicity = ua.multiplicity \land$$
$$oa.relationship = relNone \land oa.navigability = navNone$$
$$\forall uo : uc.operations \bullet$$
$$\exists oo : oc.operations \bullet$$
$$oo.name = uo.name \land oo.visibility = uo.visibility$$
$$\forall up : ran\, uo.parameters \bullet$$
$$\exists op : ran\, oo.parameters \bullet$$
$$op.name = up.name \land op.type = convType(up.type)\}$$

Funktiossa määritellään, että kaikille UML-luokan attribuuteille ja operaatioille on oltava vastaava Object-Z:n luokan attribuutti ja operaatio tyypinmuunnoksen jälkeen. Yksi UML-luokka voi kuvautua samanaikaisesti useammaksi Object-Z -luokaksi.

Vastaavat muunnosoperaatiot voidaan määritellä assosiaatioille ja UML-luokkakaaviolle. Assosiaation kuvautuminen riippuu sen ominaisuuksista. Yleisesti assosiaatio kuvataan Object-Z -luokan viitearvoiseksi attribuutiksi, kooste tavalliseksi attribuutiksi ja kompositio sisällytetyksi attribuutiksi. Assosiaatioluokat kuvataan Object-Z -luokiksi, joista on viite assosiaation kohteena oleviin luokkiin. UML-luokkakaavio kuvataan Object-Z -spesifikaatioksi.

4.2 UML:n ja Object-Z:n välisistä muunnoksista

Kim ja Carrington ovat formalisoineet luokkakaavion lisäksi myös osan UML:n metamallin muista paketeista, kuten tilakaaviot [4]. Hieman erilainen lähestymistapa on Moreiralla ja Araújolla [5], jotka ovat kehittäneet erityisesti vaatimusmäärittelyvaihetta tukevan mallin. Aluksi etsitään käyttötapaukset epämuodollisten vaatimusten perusteella. Käyttötapausten perusteella määritellään yhteistoimintakaaviot. Prosessia sovelletaan iteratiivisesti, kunnes vaatimukset tarkentuvat. Yhteistoimintakaaviokokoelman ja mallissa annettujen sääntöjen perusteella voidaan määritellä järjestelmän oliot Object-Z -luokkaskeemoina. Eri käyttötapausten perusteella voidaan määritellä luokan metodien suoritusehdot ja järjestys. Nämä ilmaistaan temporaalilogiikalla Object-Z -luokkien historiainvariantteihin.

Johtuen Object-Z:n UML-kieltä suuremmasta ilmaisuvoimasta spesifikaatiota voidaan kehittää edelleen muunnoksen jälkeen esim. uusia invariantteja lisäämällä. UML:n formalisointi tukee siis ohjelmistoprosessia, jossa karkea suunnittelu tehdään UML:llä ja tarkennettu malli, päättely ja oikeellisuuden tutkiminen voidaan tehdä Object-Z:aa hyödyntäen. UML:n formalisointi tukee myös formaalien menetelmien yleistymistä, koska UML on jo nyt — puoliformaalina — laajassa käytössä järjestelmien suunnitelussa.

Sun & al. ovat tutkineet Object-Z -kuvausten käyttöä WWW:ssä ja CASE-välineissä. He ovat määritelleet XML-esityksen Object-Z -skeemoista ja XSLT-kieliset muunnosmääritykset Object-Z:sta HTML:ään ja XMI-formaattiin. Object-Z:n XML-esitys on suunniteltu samankaltaiseksi LaTeX-järjestelmän OZ-paketin kanssa. Esimerkkinä XML-esityksestä on osa luvussa 3.1 määritellystä Stack-luokan skeemasta.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="http://nt-appn.comp.nus.edu.sg/fm/zml/objectzed.xsl"?>
<!DOCTYPE unicode SYSTEM "http://nt-appn.comp.nus.edu.sg/fm/zml/unicode.dtd">
<objectZnotation>
<classdef>
  <name>Stack</name>
  <axdef>
    <decl>
      <name>max</name>
      <dtype><type>N</type></dtype>
    </decl>
    <st/>
    <predicate>max &leq; 100</predicate>
    </axdef>
    <state>
    <decl>
      <name>items</name>
      <dtype>&seq; <type>T</type></dtype>
```

```
      </decl>
      <st/>
      <predicate>#items &leq; max</predicate>
    </state>
    <init>
      <predicate>items = &emptyseq;</predicate>
    </init>
    <op>
      <name>Push</name>
      <delta>items</delta>
      <decl>
        <name>item?</name>
        <dtype><type>T</type></dtype>
      </decl>
      <st/>
      <predicate>#items? &lt; max</predicate>
      <predicate>items' = &lseq; item? &rseq; &cat; items</predicate>
    </op>
    <!-- ... -->
  </classdef>
  </objectZnotation>
```

HTML-muunnos mahdollistaa Object-Z -skeemojen katsomisen XSLT-tuella varustetulla selaimella (matemaattisten symbolien katselu vaatii tosin Unicode-fontin käytön). XMI (*XML Metadata Interchange*) on useissa CASE-välineissä käytössä oleva XML-formaatti, joka on tarkoitettu UML-mallien vaihtoon eri ohjelmien välillä. Kuten UML, XMI ei ole yhtä ilmaisuvoimainen kuin Object-Z, mutta staattisten luokkarakenteiden siirto onnistuu. Laajempi esimerkki Object-Z:n XML-esityksestä on saatavilla artikkelin [7] kirjoittajien WWW-sivuilta [3].

5 Yhteenveto

OMG-organisaation standardoima UML-kieli on laajassa käytössä ohjelmistojärjestelmien suunnittelussa. Sen puoliformaalius estää kuitenkin UML-mallien täsmällisen analysoinnin ja päättelyn. Seminaarityössä tutkittiin UML:n formalisointimahdollisuuksia Object-Z -kuvauskielellä. Lupaava lähestymistapa on formalisoida UML:n metamalli Object-Z -luokkarakenteiksi ja täydentää Object-Z -kuvaus UML-määrityksen puutteiden osalta. UML-kielen laajuudesta johtuen formalisointi vaatii kuitenkin paljon työtä.

UML:n formalisointi edistäisi formaalien menetelmien käyttöä ohjelmisotuotannossa. XSLT-muunnos Object-Z:n XML-esityksen ja XMI-formaatin välillä helpottaa Object-Z -kuvausten käyttöä CASE-ohjelmissa. Lisäksi ohjelmistokehitysprosessin formaalius voidaan määrätä tarpeen mukaan. Vaatimusmäärittelyn alkuvaihe voitaneen edelleen hoitaa epämuodollisesti (asiakkaan ymmärtämällä kielellä), mutta suunnittelun edetessä voitaisiin siirtyä Object-Z -kuvauksiin ja tarvittaessa todistaa mallin johdonmukaisuus. Formaalin kuvauksen pohjalta myös ohjelmiston toiminnan simulointi (ennen toteutusta) ja verifiointi (toteutuksen jälkeen) onnistuu luotettavammin [2].

[3] http://nt-appn.comp.nus.edu.sg/fm/zml/

Lähteet

[1] R. Duke., G. Rose, ja G. Smith. Object-z: a specification language advocated for the description of standards. Tekninen raportti 94-45, Department of Computer Science, The University of Queensland, 1994. Saatavilla WWW:ssä, URL http://svrc.it.uq.edu.au/pub/techreports/tr94-45.ps.

[2] Antti-Juhani Kaijanaho ja Tommi Kärkkäinen. Luentomoniste: Formaalit menetelmät. Jyväskylän yliopisto, Tietotekniikan laitos, 2003.

[3] Soon-Kyeong Kim ja David Carrington. A formal mapping between uml models and object-z specifications. Kirjassa J. P. Bowen, S. Dunne, A. Galloway, ja S. King, toim., *ZB 2000: Formal Specification and Development in Z and B, First International Conference of B and Z Users York, UK*. Springer, 2000. Saatavilla WWW:ssä, URL http://www.informatik.fernuni-hagen.de/pi5/lehre/Seminare/1920_SS2002/FormalMappingBetweenUML.pdf.

[4] Soon-Kyeong Kim ja David Carrington. A formal model of the uml metamodel: The uml state machine and its integrity constraints. Kirjassa Space D. Bert, J.P. Bowen, M.C. Henson, ja K. Robinson, toim., *ZB 2002: Formal Specification and Development in Z and B, 2nd International Conference of B and Z Users, Grenoble, France*. Springer, 2002. Saatavilla WWW:ssä, URL http://link.springer.de/link/service/series/0558/papers/2272/22720497.pdf.

[5] Ana Moreira ja João Araújo. Specifying the behaviour of uml collaborations using object-z. Kirjassa *Association for Information Systems, Americas Conference on Informatin Systems*, 2000. Saatavilla WWW:ssä, URL http://www-ctp.di.fct.unl.pt/~amm/papers/collaborations2000.pdf.

[6] Perdita Stevens. On associations in the unified modelling language. Kirjassa Martin Gogolla ja Cris Kobryn, toim., *UML 2001 - The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada*. Springer, 2001. Saatavilla WWW:ssä, <URL http://dblp.uni-trier.de/>.

[7] Jing Sun, Jin Song Dong, Jing Liu, ja Hai Wang. Object-z web environment and projections to uml. Kirjassa *Tenth International World Wide Web Conference, Hong Kong*. ACM Press, 2001. Saatavilla WWW:ssä, URL http://www10.org/cdrom/papers/182/.

[8] Unified modeling language specification v1.5. Tekninen raportti 03-03-01, Object Management Group, 2003. Saatavilla WWW:ssä, URL http://www.omg.org/technology/uml/.