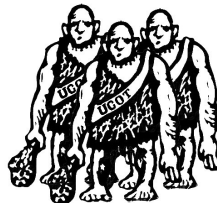


UCOT-Sovellusprojekti

Sovellusraportti

Ilari Liukko
Tuomo Pieniluoma
Vesa Pikki
Panu Suominen



Versio: 0.03
Julkinen
13. joulukuuta 2006

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Hyväksyjä	Päivämäärä	Allekirjoitus	Nimenselvennys
Projektipäällikkö	__.__.2006		
Tilaja	__.__.2006		
Ohjaaja	__.__.2006		

Tietoa dokumentista

Tekijät:

- | | | |
|-------------------------|----------------------|-------------|
| • Ilari Liukko (IL) | ilanliuk@cc.jyu.fi | 050-4367494 |
| • Tuomo Pieniluoma (TP) | tujupien@cc.jyu.fi | 040-7202054 |
| • Vesa Pikki (VP) | vevijopi@cc.jyu.fi | 044-5288031 |
| • Panu Suominen (PS) | panu.suominen@iki.fi | 050-3458484 |

Dokumentin nimi: UCOT-projekti, Sovellusraportti

Sivumäärä: 30

Tiedosto: UCOT-sovellusraportti-0.03.tex

Tiivistelmä: Tämä dokumentti on sovellusraportti Jyväskylän yliopiston tietotekniikan laitoksen UCOT-sovellusprojektille. Dokumentissa kuvataan UCOT-sovellusprojektin kehittämää ohjelmistoa.

Avainsanat: UCOT, heuristiikka, käyttötapaus, olioluokka.

Versiohistoria

Versio	Päivämäärä	Muutokset	Tekijät
0.01	21.11.2006	Luonnoksen tekeminen aloitettu.	VP
0.02	4.12.2006	Johdanto kirjoitettu ja arkkitehtuurilukua aloitettu.	IL
0.03	13.12.2006	Ensimmäinen katselmoitava versio.	IL, PS

Tietoa projektista

UCOT-projekti suunnittelee ja toteuttaa Jyväskylän yliopiston tietotekniikan laitokselle ohjelmiston. Tällä ohjelmistolla voidaan heuristiikkoja käyttäen etsiä käyttötapauksista ohjelmistokehityksen analyysivaiheen olioluokkia.

Tekijät:

- | | | |
|-------------------------|----------------------|-------------|
| • Ilari Liukko (IL) | ilanliuk@cc.jyu.fi | 050-4367494 |
| • Tuomo Pieniluoma (TP) | tujupien@cc.jyu.fi | 040-7202054 |
| • Vesa Pikki (VP) | vevijopi@cc.jyu.fi | 044-5288031 |
| • Panu Suominen (PS) | panu.suominen@iki.fi | 050-3458484 |

Tilaaaja:

- | | | |
|--------------------|--------------------|-------------|
| • Tommi Kärkkäinen | tka@mit.jyu.fi | 040-5677854 |
| • Antti Hakala | anthakal@cc.jyu.fi | 040-7096224 |

Ohjaajat:

- | | | |
|---------------------|--------------------|-------------|
| • Ville Isomöttönen | vilisom@cc.jyu.fi | 014-2604976 |
| • Miika Nurminen | minurmin@cc.jyu.fi | 014-2602530 |

Tarkkailija:

- | | | |
|-------------------|--------------------|-------------|
| • Antti Hallamäki | antahall@cc.jyu.fi | 044-3555356 |
|-------------------|--------------------|-------------|

Yhteystiedot:

- | | |
|---------------------|---|
| • Sähköpostilistat: | ucot06@korppi.jyu.fi,
ucot_opetus@korppi.jyu.fi |
| • Projektiarkisto: | https://korppi.jyu.fi/list-archive/ucot06/ind.html |
| • Opetusarkisto: | https://korppi.jyu.fi/list-archive/ucot_opetus/ind.html |
| • Työhuone: | Ag C222.2 / 014-2604963 |

Sisältö

1	Johdanto	1
2	Käyttöliittymä	2
2.1	Käyttöliittymän rakenne	2
2.1.1	Käsittemallipuu	2
2.1.2	Dot-paneeli	3
2.1.3	Käyttötapauskuvauspuu	3
2.1.4	Käyttötapauskuvauspaneeli	3
2.1.5	Tilarivi	3
2.1.6	Entiteetin ominaisuuksien muokkaus-dialogi	3
2.1.7	Asetus-dalogi	4
2.2	Dot	4
3	Arkkitehtuuri ja toiminnallisuus	7
3.1	Toimintaperiaate	7
3.2	Yleinen arkkitehtuuri	7
3.2.1	Core	8
3.2.2	UserInterface	8
3.2.3	InputAdapter	8
3.2.4	ParserAdapter	9
3.2.5	HeuristicModule	9
3.2.6	OutputAdapter	9
4	Tiedostot ja tietorakenteet	10
4.1	Luettu käyttötapaus	10
4.2	SimpleInput-muoto	11
4.3	ProcessML	11
4.4	Jäsennetty teksti	12
4.5	Käsittemalli	13
4.6	GXL	14
5	Rajapinnat	17
5.1	ControllInterface	17
5.2	InputInterface	17
5.3	ParserInterface	17

5.3.1	SimpleParser	18
5.3.2	Stanfordin jäsenin	18
5.4	HeuristicInterface	19
5.4.1	Abbottin heuristiikka	20
5.5	OutputInterface	20
5.6	UIInterface	21
6	Ohjelmointikäytännöt ja toteutusympäristö	22
6.1	Lähdekoodin ulkoasu	22
6.2	Toteutusympäristö	23
7	Jatkokehitysideat	24
7.1	Käyttöliittymä	24
7.2	Käsitemalli	24
7.3	Jäsenin	25
8	Lähteet	26
Liitteet		
A	Termit	27
B	Javadoc	30

1 Johdanto

UCOT-projekti toteutti Jyväskylän yliopiston tietotekniikan laitokselle, Agora Centerille ja Tekes-hankkeen rahoittamalle Tuotanto 2010-tutkimusprojektille sovelluksen, jolla analyysivaiheen olioluokkia voidaan muodostaa käyttötapauskuvauksista. Sovelluksen tarkoituksena on tukea, sekä osittain automatisoida olioanalyysin tekemistä jäsentämällä käyttötapauskuvausta ja erottelemalla siitä analyysin kannalta oleelliset asiat. Tätä sovelluksen erottamaa, analyysin tukena käytettävää tietoa, kutsutaan käsitteeksi.

Tämä dokumentti on UCOT-projektin sovellusraportti. Raportissa käydään läpi sovelluksen rakennetta ja osien toimintaa, sekä kuvataan ja perustellaan sovelluksen kehittämisessä käytettyjä toteutusratkaisuja. Lopuksi esitellään projektin aikana esille tulleet kehitysideat, jotka ajanpuutteen vuoksi on jätetty jatkokehitykseen.

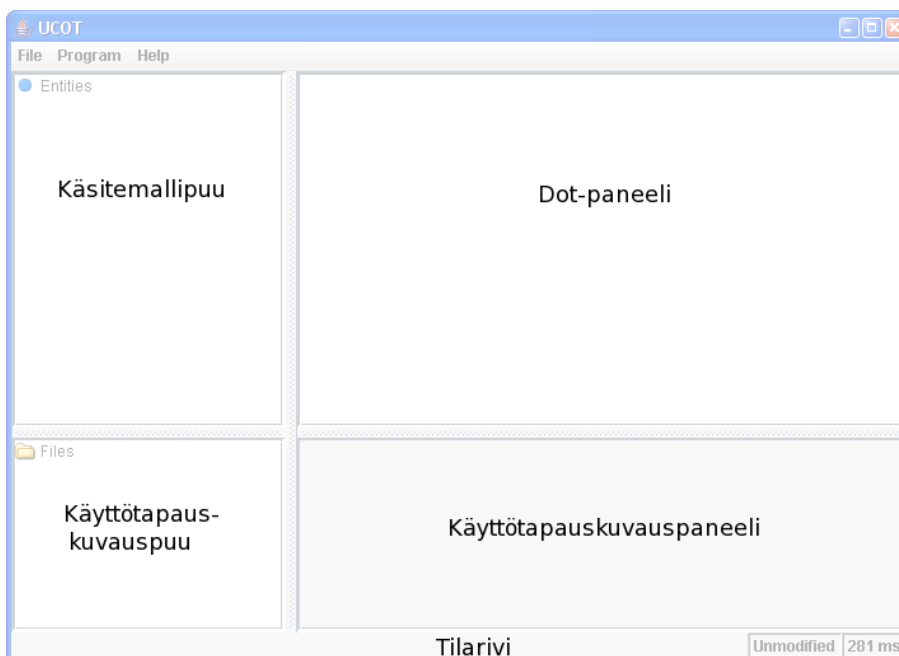
Luvussa 2 esitellään sovelluksen käyttöliittymän rakennetta ja toimintoja. Luvussa 3 esitellään sovelluksen arkkitehtuuria ja toiminnallisuutta. Luku 4 keskittyy sovelluksen sisäisten, sekä muiden sen tukemien tietorakenteiden esittelyyn. Luvussa 5 esitellään sovelluksen modulien rajapinnat. Luku 6 esittelee sovelluksen lähdekoodin muotoilu- ja kommentointikäytänteet, toteutusympäristön sekä projektissa käytetyt työkalut. Luvusta 7 löytyvät jatkokehitysideat sekä toteutusratkaisujen analyysi. Liitteessä A esitellään sovelluksessa ja projektissa esiintyvät termit. Liite B koostuu sovelluksen luokkien Javadoc-kommenteista.

2 Käyttöliittymä

Tässä luvussa esitellään UCOT-sovelluksen käyttöliittymän rakennetta ja toimintoja, sekä esitellään käyttöliittymän hyödyntämät ulkoiset ohjelmistot. Käyttöliittymän tarkempi toiminta kerrotaan Käyttöohje-dokumentissa [1].

2.1 Käyttöliittymän rakenne

Käyttöliittymän alueet on esitetty kuvassa 2.1.



Kuva 2.1: UCOT-sovelluksen käyttöliittymä.

2.1.1 Käsittemallipuu

Käsittemallipuu esittää ohjelmaan muodostetun käsittemallin sisältämät entiteetit, sekä niiden sisältämät attribuutit, metodit ja lapsientiteetit. Puussa voidaan ponnahdusvalikon kautta tehdä muutoksia käsittemalliin. Tietyn entiteetin ominaisuuksien suurempaa muokkausta varten ponnahdusvalikosta voi myös avata entiteetin ominaisuuksien muokkausdialogin (tarkemmin luvussa 2.1.6).

2.1.2 Dot-paneeli

Dot-paneeli esittää Dot-sovelluksen (tarkemmin luvussa 2.2) avulla piirretyn kuvan sovellukseen muodostetusta käsitemallista. Kuvasta korostetaan joko käsitemallissa aktiivisena oleva entiteetti ja mahdollisesti sen attribuutteja tai metodeja, tai käyttötapauskuvauspuussa aktiivisena olevasta käyttötapauskuvauksesta käsitemalliin johdetut entiteetit.

2.1.3 Käyttötapauskuvauspuu

Käyttötapauskuvauspuu esittää sovellukseen ladatut käyttötapauskuvauksia sisältävät lähteet, sekä lähteiden sisältämät käyttötapauskuvaukset. Puusta voidaan ponnahdusvalikon kautta lisätä käyttötapauskuvauksia käsitemalliin.

2.1.4 Käyttötapauskuvauspaneeli

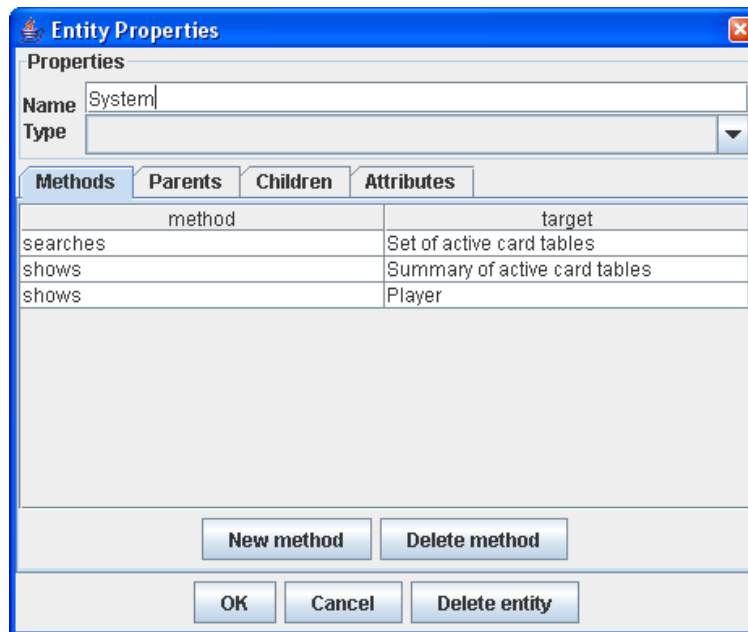
Käyttötapauskuvauspaneelissa näkyy käyttötapauskuvauspuussa aktiivisena olevan käyttötapauskuvauksen nimi ja suoritusaskeleet, sekä käytettyjen jäsenin- ja heuristiikka-adapterien nimet jos käyttötapauskuvaus on lisätty käsitemalliin.

2.1.5 Tilarivi

Tilarivillä esitetään tietoa sovelluksen tilasta. Vasemmanpuoleinen teksti kertoo, onko sovelluksen käsitemallissa tallentamattomia muutoksia. Oikeanpuoleinen teksti kertoo Dot-paneelin (luku 2.1.2) viimeiseen päivitykseen kuluneen ajan millisekunteina.

2.1.6 Entiteetin ominaisuuksien muokkaus-dialogi

UCOT-sovelluksen käsitemallin entiteettien ominaisuuksia voidaan muokata kuvassa 2.2 näkyvän dialogin avulla. Ylimpänä olevassa tekstikentässä näkyy muokattavana olevan entiteetin nimi, ja siihen voi myös muokata entiteetille uuden nimen. Nimikentän alapuolella on entiteetin tyyppivalintalista, josta entiteetille voi valita tyyppin asetusdialogissa (luku 2.1.7) asettetuista tyypeistä.



Kuva 2.2: Entiteetin ominaisuuksien muokkaus-dialogi.

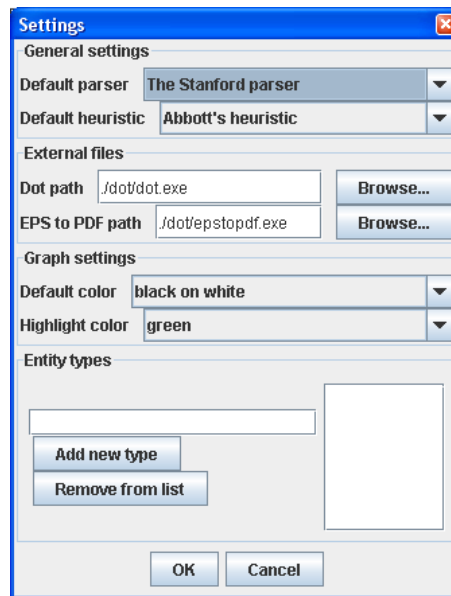
Edellä mainittujen kenttien alapuolella sijaitsevat välilehdet entiteetin metodisuhteiden, lapsi- ja vanhempientiteettien, sekä attribuuttien ja niiden kaardinaalisuuksien muokkaamiseen. Dialogissa on myös painike entiteetin poistamiseen käsitelmällistä.

2.1.7 Asetus-dalogi

UCOT-sovelluksen asetustialogi näkyy kuvassa 2.3. Dialogin avulla voidaan muuttaa sovelluksen oletusjäsenintä ja -heuristiikkaa, sovelluksen käyttämien ulkoisten komponenttien ja ohjelmien sijaintia ja Dot-sovelluksen piirtämän kuvan värejä. Asetustialogissa on myös mahdollisuus lisätä ja poistaa entiteeteille mahdollisesti lisättäviä tyyppisiä.

2.2 Dot

Dot on suunnattujen graafien asetteluun ja piirtämiseen tarkoitettu ohjelmisto. Se lukee graafin tekstimuotoista kuvausta ja tuottaa siitä graafin kuvana tai tekstimuotoisen asettelun kanssa. UCOT-ohjelmassa sitä käytetään käsitelmällistä kuvaa-



Kuva 2.3: UCOT-sovelluksen asetusdialogi.

van graafin piirtämiseen käyttöliittymässä. Käsitelmä käännetään Dot-kielelle, joka Dot-sovelluksen avulla käännetään PNG-kuvaformaattiin. Kuva ladataan sovellukseen ja esitetään Dot-paneelissa. Myös muut UCOT-ohjelman tallentamat kuvat luodaan tällä sovelluksella.

```

1: digraph G { [rankdir="TB"]
2:   <Käyttäjä>;
3:   <Käyttäjä> -> <Lähde> [label="antaa"];
4:   <Käyttäjä> -> <Käyttötapaus> [label="valita"];
5:   <Käyttötapaus>;
6:   <Lista>;
7:   <Lähde>;
8:   <Ohjelma>;
9:   <Ohjelma> -> <Lista> [label="esittää"];
10: }

```

Kuva 2.4: Esimerkki Dot-kielen syntaksista.

Dot-sovelluksen lukema tekstimuotoinen graafinkuvaus tulee olla Dot-kielen mukainen. Kieli kuvaa kolmenlaisia elementtejä graafeja (graph), solmuja (node) ja kaaria (edge). Elementtien ulkoasua voidaan muokata halutunlaiseksi monipuolisilla optioilla. Kuvassa 2.2 on kuvaus yksinkertaisesta graafista Dot-kielellä. Ensimmäisellä rivillä kerrotaan graafin nimi ja tyyppi, tässä suunnattu graafi (digraph), sekä annetaan optio graafin piirtämiselle. Rivillä kaksi määritellään *Käyttäjä*-niminen solmu. Kolmannella rivillä määritellään kaari edellisen ja *Lähde*-nimisen solmun välille,

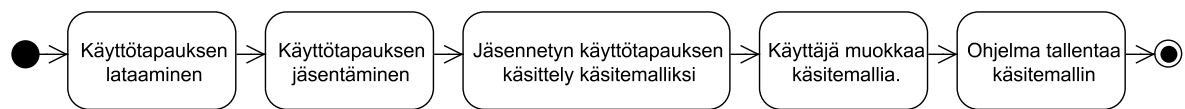
jolle annetaan nimeksi *antaa*.

Dotin tuottama tekstimuotoinen kuvaus sisältää graafin nimen, graafin sisältämät solmut ja niiden sijoittelun, sekä solmuja yhdistävät kaaret ja niiden asettelu b-spline-käyränä. Dot-sovelluksen tukemia kuvaformaatteja ovat muun muassa GIF, PNG, SVG ja PostScript (joka voidaan muuntaa PDF muotoon).

3 Arkkitehtuuri ja toiminnallisuus

Tässä luvussa käydään läpi sovelluksen arkkitehtuuria, sekä sovelluksen osien toimintaa ja tehtäviä. Lisäksi esitellään sovelluksen hyödyntämät ulkopuoliset komponentit ja sovellukset.

3.1 Toimintaperiaate



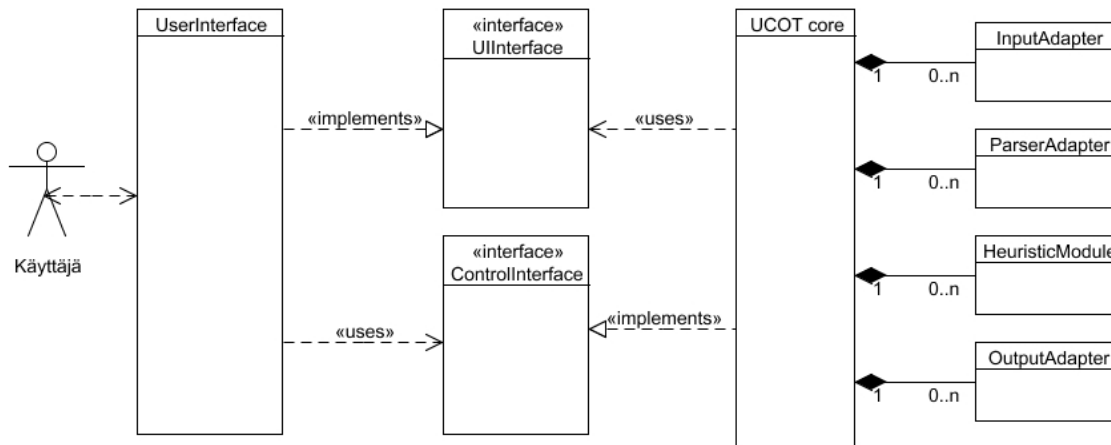
Kuva 3.1: Sovelluksen toimitavaiheet.

Ohjelman käytön vaiheet on esitetty kuvassa 3.1.

Käyttötapausten kulku ohjelman sisällä alkaa käyttötapausten lataamisesta. Tämän jälkeen se jäsennetään, jolloin luonnollisella kielellä kirjoitettu teksti muutetaan tietokoneen ymmärtämään rakenteisempaan esitysmuotoon. Varsinainen käsitteellinen muodostetaan jäsennetyin tiedon perusteella heuristiikkaa (luku 3.2.5) soveltaen. UCOT-ohjelmassa heuristiikka kuvaa jäsentimen löytämiä rakenteita käsitteellisen osiksi. Käyttötapausten kirjoittaminen ei ole välttämätöntä vaan käsitteellisiä voidaan rakentaa myös tyhjästä, jolloin se ei liity varsinaisesti mihinkään käyttötapaukseen. Tässä kuitenkin sivuutetaan iso osa ohjelman toiminnallisuudesta.

3.2 Yleinen arkkitehtuuri

Sovellus suunniteltiin modulaariseksi, jotta sovelluksen komponentteja voidaan myöhemmin korvata tai lisätä tarpeen mukaan. Myös testaaminen helpottui, kun jokaisen komponentin toiminta voitiin tarkistaa erikseen. Kuvassa 3.2 on kuvattu sovelluksen arkkitehtuurin yleiskuva. Seuraavaksi esitellään kuvan pääkohdat tarkemmin. Luvussa 5 käydään tarkemmin läpi, kuinka moduuleihin jako on rajapintojen kannalta toteutettu.



Kuva 3.2: Sovelluksen arkkitehtuuri.

3.2.1 Core

Sovelluksen toiminnan kannalta keskeisin komponentti on *core*. Sen vastuulla on sovelluksen muiden komponenttien hallinta, sekä niiden toiminnan ohjaaminen. Coren vastuulla on myös muiden komponenttien alustaminen sovelluksen käynnistyessä. Core hoitaa sovelluksen *InputAdapter*in lukeman datan ohjaamisen käytettävälle *ParserAdapter*ille ja sen ulostulon ohjaamisen *HeuristicAdapter*ille.

3.2.2 UserInterface

Ohjelman käyttöliittymäkomponentti, jonka kautta käyttäjän on mahdollista ohjata sovelluksen toimintaa. Ohjelmaan on toteutettu graafinen käyttöliittymä, jota esitellään luvussa 2 ja ohjelman käyttöohjeessa.

3.2.3 InputAdapter

InputAdapter:n vastuulla on käyttötapauskuvausten lataaminen sille osoitetusta lähteestä. Lähde voi olla kovalevyllä sijaitseva tiedosto, verkossa sijaitseva tiedosto tai jokin vastaava, jonka sijainnin voi osoittaa URL:n avulla.

InputAdapter lataa ja jäsentää coren sille toimittaman lähteen sisältämät käyttötapauskuvaukset sovelluksen sisäiseen esitysmuotoon, jota esitellään tarkemmin

luvussa 4. Lataamisen jälkeen `InputAdapter` toimittaa lähteestä saadut käyttöta-
pauskuvaukset `corelle`.

3.2.4 `ParserAdapter`

`ParsedAdapter` suorittaa `coren` sille sovelluksen sisäisessä esitysmuodossa anta-
malle jäsentämättömälle käyttötapauskuvaukselle morfologisen jäsennyksen.

`ParserAdapter` palauttaa `corelle` saamansa käyttötapauskuvauksen jäsennet-
tynä.

`ParserAdapter` voi itse suorittaa jäsennyksen sille annetulle käyttötapauskuvauk-
selle tai se voi toimia sovelluksen jäsenin-rajapinnan toteuttavana sovittimena ul-
kopuoliselle jäsentimelle.

3.2.5 `HeuristicModule`

`HeuristicModule:n` tehtävänä on muuntaa jäsennetty käyttötapauskuvaus käsi-
temalliksi.

3.2.6 `OutputAdapter`

Komponentin tehtävä on käsitemallin muuntaminen ja tallentaminen ulkoiseen muo-
toon. Ulkoinen muoto voi olla jokin graafien kuvauskieli tai jokin kuvaformaatti.
Ohjelmaan toteutettiin käsitemallin tallentamisen GXL-muotoon, jota esitellään tar-
kemmin luvussa 4.6.

4 Tiedostot ja tietorakenteet

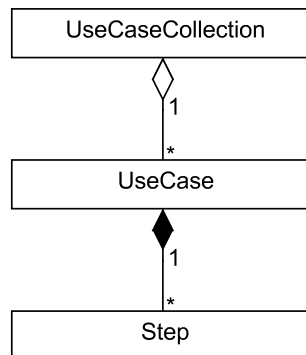
Tässä luvussa kuvataan ohjelman käyttämien tiedostojen rakennetta sekä kerrotaan käytetyistä tietorakenteista.

Tärkeimpänä ohjelman sisäisenä tietorakenteena on käsitelmä (luku 4.5). Tämän lisäksi `InputAdapter`:n lukemille käyttötapauksille sekä `ParserAdapter`:n jäsenneyille käyttötapauksille on oma muotonsa. Tiedon muoto vaihtuu käsittelyn edetessä (katso kuva 3.1).

Tiedostoja ohjelma käyttää käyttötapauksen lähteenä ja valmiin käsitelmän kirjoittamiseen mahdollista jatkokäsittelyä varten. Samoin asetuksia tallennetaan XML-tiedostoon.

4.1 Luettu käyttötapaus

Ohjelman lukiessa käyttötapauksia, ne tallennetaan ohjelman sisälle jatkokäsittelyä varten yleiskäyttöisessä muodossa, joka ei ole riippuvainen lähteestä, josta käyttötapaukset on luettu. Tällä tavoin jäsennin voidaan erottaa varsinaisesta tiedostojen tai muiden mahdollisten lähteiden lukemisesta.



Kuva 4.1: Luetun käyttötapauksen luokkarakenne.

Kuvassa 4.1 on esitetty käyttötapauksen tallennukseen käytetty tietorakenne. `UseCase` on luokka, joka pitää sisällään käyttötapauksen. Käytännössä luokka sisältää vain vektorin `UseCaseStep`-tyyppisistä olioista. `UseCaseStep` on nimensä mukaisesti käyttötapauks kuvauksen askel ja se pitää sisällään askeleen tekstin. Yhden lähteen käyttötapaukset sisällytetään kaikki `UseCaseCollection` olion sisälle, jota ohjelma käyttää jäsentimen syötteenä. Tietojen lukemisesta kyseiseen muotoon vastaa

jokin `InputInterfacen` (luku 5.2) toteuttava olio. Yksi esimerkki tällaisesta on `SimpleInputAdapter`, jonka lukema tiedostomuoto esitellään seuraavaksi.

4.2 SimpleInput-muoto

Ohjelmaa kehitettäessä tarvittiin yksinkertainen tiedostomuoto käyttötapauksien kirjoittamiseksi. Tämä muoto nimettiin `SimpleInput`-muodoksi sitä lukevan `SimpleInputAdapter`-luokan mukaisesti. Lähtökohtana oli, että käyttötapaukset voidaan kirjoittaa helposti tavallisella tekstieditorilla ja ohjelman tarvitsemista merkkauksista huolimatta esitys säilyy luettavana. Tiedosto sisältää käyttötapauksien nimet ja askeleet. Lisäksi askeleita voidaan tarkentaa alikäyttötapauksilla. Ohjelman tarvitsemat merkkaukset kirjoitetaan hakasulkujen sisälle. Seuraavat merkinnät tunnistetaan: `name`, `id`, `steps`, `end`.

```
1: [name]
2: Osta tuote
3: [id]
4: 1
5: [steps]
6: Hanki rahat. (2)
7: Anna rahat myyjälle.
8: Ota tuote.
9: [end]
```

Kuva 4.2: Esimerkki `SimpleInput`-muodosta.

Yksi tiedosto voi sisältää useampia kuvan 4.2 tapaan kirjoitettuja käyttötapauksia. Hakasulkujen sisällä oleva `name` aloittaa käyttötapauksen ja `end` lopettaa sen. Askeleet seuraavat `steps`-merkintää ja käyttötapauksen tunniste kirjoitetaan `id`:n jälkeen. Askeleen peräessä olevien sulkeiden sisällä viitataan askeleeseen liittyvän alikäyttötapauksen tunnisteeseen. Alikäyttötapauksia etsitään vain samasta tiedostosta. Yllä olevassa esimerkissä ensimmäiseen askeleeseen liitettäisiin käyttötapaus, jonka tunniste on määritelty 2:ksi kirjoittamalla se `[id]`-merkinnän alle.

4.3 ProcessML

`ProcessML` on Jyväskylän Yliopistossa kehitetty prosessien kuvaamiseen suunniteltu XML-kieli. UCOT-ohjelma osaa lukea myös kyseisellä formaatilla kirjoitettuja käyttötapauksia. `ProcessML`-tiedosto sisältää kuitenkin paljon enemmän tietoa

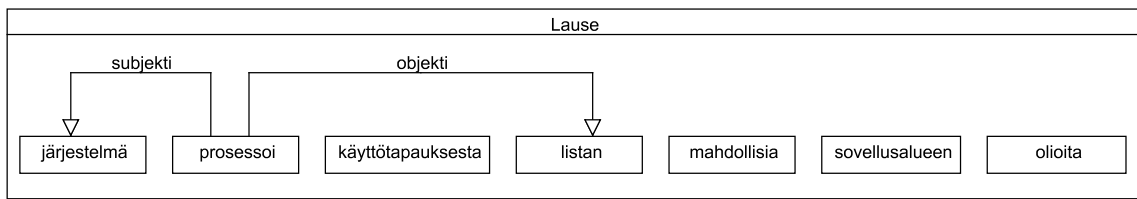
```
1: <?xml version="1.0"encoding="ISO-8859-1"?>
2: <modelDataBase>
3:   <processModel>
4:     <processInstance id="i1">
5:       Buy Something
6:       <abstraction level="0">
7:         <step id="s1">User accesses system with ID and password</step>
8:         <step id="s2">System validates user.</step>
9:         <step id="s3">User provides name.</step>
10:        <step id="s4">User provides address.</step>
11:        <step id="s5">User provides telephone number.</step>
12:        <step id="s6">User selects product.</step>
13:        <step id="s7">User identifies quantity.</step>
14:        <step id="s8">System validates that user is a known customer.</step>
15:        <step id="s9">System opens connection to warehouse system.</step>
16:        <step id="s10">System requests current stock levels from warehouse system.</step>
17:        <step id="s11">Warehouse storage system returns current stock levels</step>
18:        <step id="s12">System validates that requested quantity is in stock.</step>
19:      </abstraction>
20:    </processInstance>
21:  </processModel>
22:</modelDataBase>
```

Kuva 4.3: Esimerkki ProcessML-muodosta.

kuin mitä UCOT-ohjelma tarvitsee käsitelmän luomiseen ja tästä syystä osa tiedoston sisältämästä informaatiosta sivuutetaan. Kuvassa 4.3 on ProcessMLää mukaileva XML-tiedosto. Se ei ole validi ProcessML:n kannalta, mutta esittelee ohjelman käyttämän osan kyseisestä notaatiosta. Käyttötapausta vastaa `processInstance`, josta UCOT-ohjelma käyttää vain abstraktiotasoa 0.

4.4 Jäsennetty teksti

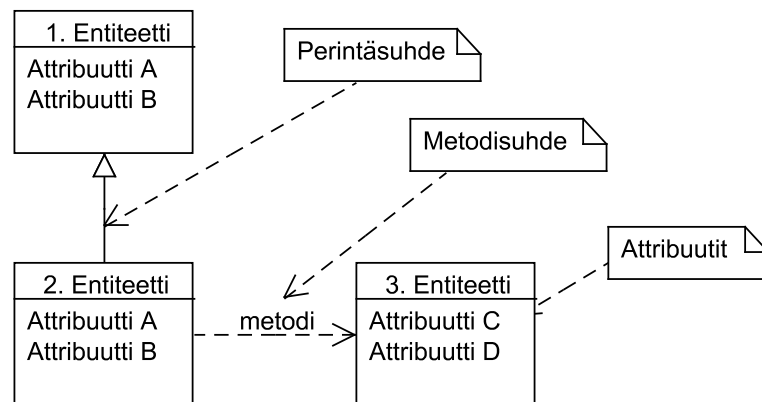
Käyttötapausten askeleet pitää jäsentää ennen kuin niitä voidaan alkaa tutkimaan heuristiikan avulla ja muodostaa niistä käsitelmä. Jäsennin tuottaa luetuista käyttötapauksista (`UseCaseCollection`) jäsennetyn tekstin (`ParsedData`). Jäsennetty teksti koostuu lauseista (`Sentence`), jotka ovat lista sanoista (`Word`). Sanat sisältävät tiedon kyseisen sanan sanaluokasta ja sen mahdollisista linkeistä muihin sanoihin. Jäsennetty esitysmuoto muistuttaa kuvan 4.4 esitystä.



Kuva 4.4: Kuva jäsenetystä teksistä.

4.5 Käsitelmä

Käsitelmä on UCOT-ohjelman monimutkaisin tietorakenne. Se tarkoituksena on pitää tallessa analyysivaiheen oliomallissa esiintyviä asioita. Tällaisia ovat mm. entiteetit, niiden metodit, attribuutit ja vanhemmat.

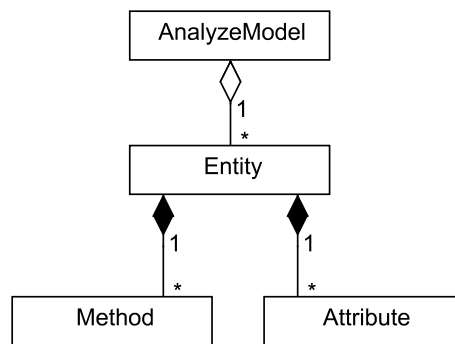


Kuva 4.5: Käsitelmän logiikka.

Kuvassa 4.5 on esitetty käsitelmän toteutuksesta riippumaton logiikka. Entiteetti on käsitelmän perusosanen, jonka voidaan ajatella edustavan oliota tai luokkaa. Entiteetti sisältää metodeja, jotka voivat tarkoittaa suoraan olioiden ja luokkien metodeja tai muuta toiminnallisuutta. Näillä toiminnoilla on kohteenaan jokin toinen entiteetti. Tämä kohde tarkoittaa metodin parametria, paluuarvoa, muutettavaa kohdetta tai muuta asiaa, johon metodilla on jokin vaikutus tai suhde. Lisäksi metodilla on nimi, joka kuvaa tätä vaikutussuhdetta.

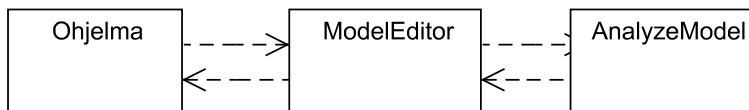
Metodien lisäksi entiteetti voi sisältää attribuutteja, jotka kuvaavat yleisimmin entiteetin sisältävän yhden tai useamman jotakin toista entiteettiä. Oliomallia mukaillen entiteeteillä on myös mahdollisuus olla jonkin toisen entiteetin alientiteetti, niin sanottu lapsi tai ylientiteetti eli vanhempi. Tämä kuvaa löyhästi luokkien peritymistä toisistaan. Entiteeteille voidaan määritellä myös tyyppi, joka on vapaasti valittava merkkijono.

Käsitemallin tarkoituksena on tarjota käyttäjälle mahdollisuus mahdollisimman paljon UMLää muistuttavien rakenteiden käsittely. Ohjelma ei kuitenkaan sido käsitemallissa olevien asioiden merkitystä mihinkään vaan merkityksen määrittelyille asioille antaa viimekädessä käyttäjä ja käsitemallia mahdollisesti myöhemmin käyttävät ohjelmat.



Kuva 4.6: UML-kaavio käsitemallin toteutuksesta.

Käsitemalli on ohjelmassa toteutettu kuvan 4.6 mukaisena. Kehitysvaiheessa oli käytössä myös toisenlainen toteutus, jossa mallin asiat oli tallennettu eräänlaisina suhteina. Kun vaatimukset tarkentuivat pystyttiin mallin toteutuskin rakentamaan paremmin vaadittuun toiminnallisuuteen sopivaksi.



Kuva 4.7: Käsitemallin muokkain.

Käsitemalli on erotettu ohjelmasta niin sanotun muokkaimen (ModelEditor) taakse (kuva 4.7). Tällöin varsinaisen mallin toteutustapa ei vaikuta ohjelman toimintaan. Toteutustapa mahdollisti käsitemallin toteutuksen vaihtamisen kesken ohjelman kehitystä. Itse muokkain on käytännössä katsoen kokoelma metodeja, jotka sisältävät kaikki mahdolliset toimenpiteet, mitä mallin käsittely vaatii.

4.6 GXL

Jotta käsitemallista olisi hyötyä, pitää se pystyä välittämään muille ohjelmille. Yksi tällainen yleiskäyttöinen muoto on GXL (Graph Exchange Language), joka on tarkoitettu graafien kuvaamiseen. Se on XML-muotoista dataa, joka käyttää sille suunnitettua

niteltua syntaksia. Tässä esitetään syntaksi UCOT-ohjelman tarvitsemilta osin. Tarkeempaa tietoa asiasta löytyy GXL-projektin kotisivuilta <http://www.gupro.de/GXL/>.

Koska käsitelmä pystytään hyvin kuvaamaan graafina sopii muoto ohjelman tarpeisiin hyvin. Käsitelmä kuvataan graafiksi siten, että entiteetit ovat solmuja ja niitä yhdistävät kaaret ovat metodisuhteita yms.

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <!DOCTYPE gxl SYSTEM "http://www.gupro.de/GXL/gxl-1.0.dtd" >
3: <gxl xmlns:xlink="http://www.w3.org/1999/xlink" >
4:   <graph edgemode="directed" hypergraph="false" id="UCOT-Model" >
5:     <node id="node1" >
6:       <attr name="name" >
7:         <string>1. Entiteetti</string>
8:       </attr>
9:     </node>
...

```

Kuva 4.8: Entiteetin määrittely GXL-tiedostosta.

Kuvassa 4.8 määritellään graafi nimeltä "UCOT-Model" (rivi 4), joka sisältää solmun nimeltä "1. Entiteetti". Entiteetin nimi määritellään solmun attribuutiksi nimeltä "name".

```
...
20: <edge from="node1" id="edge1" to="node2" >
21:   <attr name="type" >
22:     <string>method</string>
23:   </attr>
24:   <attr name="name" >
25:     <string>metodi</string>
26:   </attr>
27:   <attr name="isdirected" >
28:     <bool>true</bool>
29:   </attr>
30: </edge>
...

```

Kuva 4.9: Metodien määrittely GXL-tiedostossa.

Kuva 4.9 määrittelee kahden entiteetin välille metodin vaikutussuhteen. Kaaren tyyppi ("method") määritellään attribuutilla "type" (rivi 21-23). Metodille on annettu nimi "metodi" (rivi 25).

Attribuuttisuhteet määritellään vastaavasti, mutta kaarella ei ole nimeä vaan sille on annettu solmujen väliset kardinaalisuudet kuvan 4.10 rivien 39-44 tapaan. Ensimmäinen osa suhteesta (rivi 41) on suhde lähtösolmuun nähden ja toinen koh-

desolmun suhteen. Lisäksi kaaren tyyppi (rivit 32-34) on erilainen kuin metodilla ("attribute").

```
...
31: <edge from="node1" id="edge2" to="node2"»
32: <attr name="type"»
33: <string>attribute</string>
34: </attr>
35: <attr name="isdirected"»
36: <bool>true</bool>
37: </attr>
38: <attr name="limits"»
39: <tup>
41: <string>1</string>
42: <string>1..N</string>
43: </tup>
44: </attr>
45: </edge>
...
```

Kuva 4.10: Attribuutin määrittely GXL-tiedostossa.

Kuvassa 4.11 liitetään entiteettiin perintä (vanhempi) suhde. Tämä tapahtuu luomalla kaari entiteettien välille. Kaaren kohde on lähtösolmun vanhempi. Kaaren tyyppi määritellään taas sopivaksi (rivit 47-49) eli vanhemman tapauksessa "parent".

```
...
46: <edge from="node2" id="edge3" to="node3"»
47: <attr name="type"»
48: <string>parent</string>
49: </attr>
50: <attr name="isdirected"»
51: <bool>true</bool>
52: </attr>
54: </edge>
55: </graph>
...
```

Kuva 4.11: Vanhemman määrittely GXL-tiedostossa.

Vaikka GXL on yleiskäyttöinen kuvauskieli jää UCOT-ohjelmasta riippuvaisten merkintöjen semantiikka muilta ohjelmilta ymmärtämättä. Tämä ongelma voidaan ratkaista kirjoittamalla XML-tiedostolle muunnin esimerkiksi XSLT-kieltä käyttäen. Lisäksi ohjelman rakenne mahdollistaa muun tyyppisten ulostulojen lisäämisen helposti.

5 Rajapinnat

Ohjelman toteutuksessa varsinainen toiminnallisuus on pyritty piilottamaan mahdollisimman hyvin rajapintojen taakse, jotta ohjelman iteratiivinen kehittäminen onnistuisi helposti. Tässä luvussa esitellään arkitehtuurillisesti tärkeimmät rajapinnat. Tämän lisäksi ohjelman sisällä käytetään usealle pienemmälle komponentille yhteisiä liityntätapoja. Suurin osa näistä on pieniä käyttöliittymäkomponentteja, joiden esittely tässä sotkisi turhaan kokonaiskuva. Tarkempaa tarkastelua varten liitteessä B on yksityiskohtaisempi dokumentointi kaikista luokista.

5.1 ControlInterface

Vaikka ohjelman varsinaisen ytimen vaihtuminen ei sinällään ole todennäköistä, on rajapinnan määrittelemisen sille perusteltua jo muunmuassa testaamisen vuoksi. Järjestelmää ohjaavia osia (lähinnä käyttöliittymä) voidaan testata itsenäisesti, kun ytimen toteutus on irroitettavissa muista osista.

Rajapinta tarjoaa keinot käyttötapausten lukemiseksi, jäsentämiseksi ja tulostamiseksi. Lisäksi ohjelman käyttöä, kuten sammuttamista, kontrolloidaan `ControlInterface:n` kautta. Käytännössä rajapinta tarjoaa keinot luoda käsitte-malli muiden komponenttien avulla.

5.2 InputInterface

Nimensä mukaisesti `InputInterface` on rajapinta sisääntulolle. Sen tarjoaa tiedon lukemisen jostakin lähteestä. Tieto palautetaan ohjelman sisäisenä käyttötapauksen esitysmuotona (katso luku 4.5), jota voidaan käyttää myöhemmin parserin kanssa. Lisäksi rajapinta osaa vastata, osaako se lukea jossakin URL:ssa sijaitseva käyttötapauksukuvauksia sisältävän lähteen.

5.3 ParserInterface

Rajapinnan tehtävän on piilottaa muutosprosessi, jossa luettu käyttötapaus muutetaan heuristiikalle sopivaan muotoon. Moduuli saattaa käyttää ulkoista jäsenintä,

kuten esimerkiksi Stanfordin jäsentimen tapauksessa (luku 5.3.2), toteuttaa omansa tai lukea jäsenitys tiedot vaikka XML-merkinnöistä. Jäsentimen tehtävä on selvittää käyttötapauskuvauksen askeleiden sisältämien lauseiden sanojen sanaluokat ja niiden väliset suhteet.

5.3.1 SimpleParser

Suomenkielisen jäsentimen puuttumisen vuoksi ohjelmaan toteutettiin yksinkertainen jäsenin (`SimpleParser`), joka ei varsinaisesti suorita tekstin jäsenystä vaan lukee tiedon valmiiksi rakenteisessa muodossa. Kuvassa 5.3.1 on esitetty yksinkertainen "lause". Sanat erotetaan toisistaan pilkuilla. Ensimmäinen ja kolmas sana tulkitaan substantiiveiksi, jotka Abbotin heuristiikassa (kts. luku 5.4.1) kuvautuvat entiteeteiksi. Välissä oleva toinen sana tulkitaan verbiksi (Abbotin heuristiikassa metodi). Tämä formaatti ei ole kauhean käyttökelpoinen tuotantokäytössä, mutta tarjosi projektiryhmälle mahdollisuuden jatkaa kehitystyötä ilman varsinaista jäsenintäkin.

Käyttäjä, käyttää, ohjelma

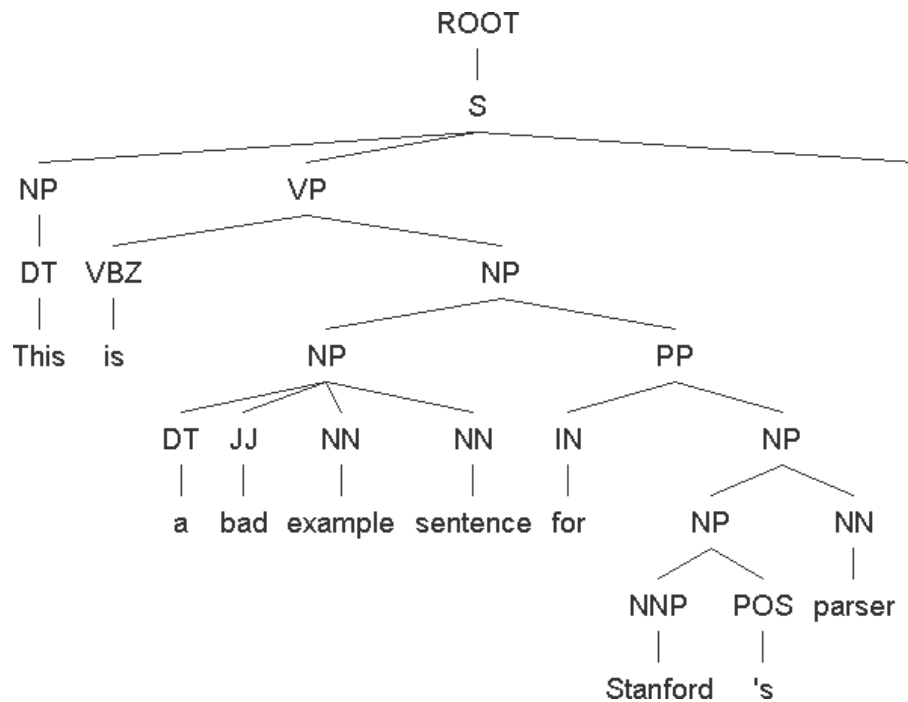
Kuva 5.1: "Lause" SimpleInput-mudossa.

5.3.2 Stanfordin jäsenin

Stanford natural language processing group (<http://nlp.stanford.edu/>) on kehittänyt tilastollisen luonnollisen kielen jäsentimen muunmuassa englannin kielelle. Tästä jäsentimestä puhutaan UCOT-projektin dokumenteissa Stanfordin jäsentimenä. Se osaa löytää jäsenityspuun engalnninkielisille lauseille, josta selviää sanan kieliopin mukainen asema lauseessa.

Stanfordin jäsenin ei ole osa UCOT-ohjelmaa, mutta sitä käytetään tässä esimerkkinä luonnollisen kielen jäsentimestä. Stanfordin jäsenintä voidaan haluttaessa käyttää liittämällä se ohjelmaan jälkikäteen niin sanottuna lisäosana. Stanfordin jäsenintä ei haluttu ottaa varsinaiseen UCOT-ohjelmaan mukaan sen tiukemman lisensoinnin takia.

Kuvassa 5.2 on jäsenityspuu lauseelle "This is a bad example sentence for Stanford's parser". Tärkeimmät solmut puusta ovat NP eli substantiivilause (noun phrase) ja VP eli verbilause (verb phrase). Jokin muu jäsenin voi nimetä solmut eri



Kuva 5.2: Esimerkki jäsennyspuusta.

tavoin, mutta niiden merkitys pysyy kuitenkin samanlaisena. Suurin osa UCOT-ohjelman vaatimasta tiedosta on näiden kahden tyyppin solmujen lapsisolmuissa. Sanojen sanaluokat laitetaan jäsennyspuun solmujen mukaisesti siten, että verbi-lauseessa määritelty verbi (VBZ) merkitään verbiksi ja osa substantiivilauseen juurisolmut merkitään substantiiveiksi. Käytännössä tällaisia solmuja ovat lähinnä NN:t. Lisäksi substantiivilauseita käsitellään ohjelmassa kokonaisuuksina siten, että esimerkkilauseesta syntyisi entiteetti nimeltä "a bad example sentence".

5.4 HeuristicInterface

HeuristicInterface:n tehtävänä on muuttaa sen saama tieto käsitemalliksi. Rajapinnalle annetaan käyttötapaus peräkkäisinä lauseina, joihin on merkitty sanojen sanaluokat ja niiden väliset suhteet. Heuristiikka tekee tästä tiedosta päätelmät minkälaisia entiteettejä, metodeja ja muita käsitemallin osia se luo.

5.4.1 Abbottin heuristiikka

Ohjelmaan toteutettiin Abbottin heuristiikkaan perustuva arviointimenetelmä (taulukko 5.3). Luonnollisen kielen sanat kuvautuvat oliomalliin osiksi sanaluokkien perusteella. Yksinkertaisimmillaan lauseen substantiivit kuvautuvat luokiksi, verbit metodeiksi ja adjektiivit attribuuteiksi. Tässä tulee kuitenkin ongelmia, koska saatavan oliomallin laatu riippuu täysin käyttötapauksen kirjoitustyylistä. Toinen ongelma tulee siitä, että substantiiveja on käyttötapauks kuvauksessa paljon enemmän kuin asiaankuuluvia luokkia. Abbottin heuristiikka soveltuu kuitenkin kärkeän oliomallin luomiseen, josta voidaan edelleen muokata paremmin käyttötapauks kuvausta vastaava malli.

Sanaluokka	Merkitys Abbotin mukaan	Merkitys käsitemallissa
Erisnimi	Olio	Entiteetti
Yleisnimi	Luokka	Entiteetti
Tekemisen verbi	Operaatio	Metodi
Olemisen verbi	Perintä	Metodi
Omistamisen verbi	Koostuminen	Metodi
Käsky	Rajoite	Metodi
Adjektiivi	Attribuutti	

Kuva 5.3: Abbottin heuristiikka.

Taulukossa 5.3 oikeenpuolimmanen sarake kuvaa, miltä osin Abbotin heuristiikka pystyttiin sovelluksessa käyttämään. Puutteet toteutuksessa johtuvat heuristiikan sopimattomuudesta automaattiseen päättelyyn ja ongelmista jäsenyspuun tarjoaman tiedon siirtämisestä heuristiikalle.

5.5 OutputInterface

Käsitemalli voidaan tallentaa `OutputInterface`:n kautta johonkin ulkoiseen muotoon. Rajapinnan avulla ohjelmaan voidaan helposti lisätä mahdollisuus tallentaa tiedot jatkokäsittelyyn mahdollistavaa muotoa käyttäen.

5.6 UIInterface

UIInterface tarjoaa UCOT-ohjelmalle mahdollisuuden viestiä käyttäjälle. Rajapinta ottaa vastaan viestejä ohjelman eri vaiheiden (käyttötapauksen lataaminen, jäsentäminen, heuristiikan käyttäminen, tallentaminen) valmistumisesta ja tarjoaa mahdollisuuden ilmoittaa virheistä käyttäjälle.

6 Ohjelmointikäytännöt ja toteutusympäristö

6.1 Lähdekoodin ulkoasu

```
package ucot;

import java.text.MessageFormat;

/**
 * <pre>public class <b>Messages</b></pre>
 * <p>This class contains interface to the localised message strings.</p>
 *
 * @author pajumasu
 */
public class Messages {

    private static final String BUNDLE_NAME = "ucot.messages";

    /**
     * <p>Returns localized string for the given key.
     * If no string is found then !key! is returned
     * (where key is the key-string that was given as parameter)</p>
     *
     * @param key Key for the string required.
     * @return localized string for the given key.
     */
    public static String getString(String key) {
        try {
            return RESOURCE_BUNDLE.getString(key);
        } catch (MissingResourceException e) {
            return '! ' + key + '!';
        }
    }
}
```

Kuva 6.1: Esimerkki lähdekoodin muotoilukäytännöstä.

Lähdekoodin muotoilussa ja kommentoinnissa seurattiin Java-kielen muotoilu periaatteita. Käytetty Eclipse-ohjelmointiympäristö mahdollisti koodin automaattisen muotoilun kuvassa 6.1 esitetyn muodon mukaisesti. Luokkien ja metodien kommentointi kirjoitettiin Javadoc-komentointikäytännön mukaisesti.

Lähdekoodin muuttujat, luokat ja kommentointi kirjoitettiin englannin kielellä, koska esimerkiksi ääkösten käyttäminen lähdekooditiedostoissa aiheuttaa merkistöongelmia.

6.2 Toteutusympäristö

Sovellus toteuttiin Java standard edition 5.0 -ympäristöön. Toteutuksessa käytettiin apuna Eclipse 3.1 -ohjelmointiympäristöä.

Sovellus hyödyntää myös ulkoisia komponentteja: Stanford Parser -jäsenointä käytetään käyttötapauskuvauksen jäsentämiseen (tarkemmin luvussa 5.3.2) ja Dot-sovellusta käytetään käsitelmällin piirtämiseen (tarkemmin luvussa 5.4.1).

7 Jatkokehitysideat

Luvussa käsitellään ohjelmiston kehittämisideoita, joita tämän projektin puitteissa ei pystytty toteuttamaan. Lisäksi pohditaan, mitä asioita olisi kannattanut jälkikäteen ajateltuna tehdä toisin.

7.1 Käyttöliittymä

Käyttöliittymä nojaa vahvasti Dot-ohjelmistoon, mikä hidastaa käyttöliittymän toimintaa ja lisää turhaa monimutkaisuutta UCOT-ohjelman koodiin. Lisäksi Dot:n piirtämä staattinen kuva tarjoaa hyvin vähän lisäarvoa käyttäjälle. Alusta asti oli selvää, että ohjelman joustavampi käyttäminen vaatisi tämä korvaamista dynaamisemmalla esitystavalla, jota käyttäjä pystyisi muokaamaan suoraan. Kuitenkin projektin päämääräksi asetettu "proof of concept" (toteutuskelpoisuuden todistaminen) toteutui karummalla käyttöliittymälläkin.

Koska tilaajalla on tarvetta myös ohjelman varsinaiselle käytölle niin sanotun konseptin todistamisen ("proof of concept") ajatuksen lisäksi, olisi luultavasti kannattanut aloittaa käyttöliittymän tekeminen suoraan jollakin graafien käsittelyä tukevalla Java kirjastolla (esim. TouchGraph tai JGraph). Tämä vaihtoehto sivuutettiin aluksi liian isotöisenä ja päädyttiin nykyiseen ratkaisuun. Dot:n käyttäminen ei luultavasti kuitenkaan ollut yhtään sen helpompaa kuin muidenkaan vaihtoehtojen.

Jatkokehitystä silmällä pitäen nykyisessä käyttöliittymässä ns. Dot-paneeli on helposti irrotettavissa ja korvattavissa toisenlaisella esitysmuodolla.

7.2 Käsitemalli

Yksi ohjelman käytökelpoisuuteen suuresti vaikuttava tekijä on käsitemallin kyky ilmaista vaadittuja asioita. Koska ohjelman käyttökokemukset ovat hyvin puutteellisia on todennäköistä, että tämänhetkinen malli on puutteellinen. Tämä luo todennäköisesti paineita käsitemallin muokkaamiselle. Hyödyllisiä ominaisuuksia voisivat olla esimerkiksi nimet attribuuteille, tarkempi tyyppien tarkoituksen määrittäminen ja ilmenemis-suhde (Matti on ihmisen ilmenemismuoto).

Jälkikäteen ajateltuna projektiryhmän olisi tullut vaatia tilaajalta mahdollisimman

kattavat tiedot malliin haluttavista tiedoista. Vaikka malli kooditason toteutus vaihdettiin täysin, saatiin se toteutettua nopeasti koodin hyvän modulaarisuuden takia.

7.3 Jäsennin

Suomenkielen jäsentimen käyttäminen ohjelman kanssa parantaisi sen käyttömahdollisuuksia huomattavasti. Suurin ongelma käyttöönotossa on raha, sillä sopivia jäsentimiä ei ilmaiseksi ole saatavilla. Lisäksi pelkkä rakenteellinen jäsentäminen ei tarjoa aina riittävää lopputulosta vaan jäsenitys modullin olisi hyvä pystyä myös rajoitettuun merkityksien analysointiin. Tällä tavoin saadaan tarkempaa tietoa entiteettien välisistä suhteista.

Alkuperäinen idea Abbotin heuristiikan koneellisesta käyttämisestä on jälkikäteen ajateltuna huono, koska kyseinen heuristiikka kuvaa vain sanaluokkien merkityksen. Ihminen joutuu tekemään paljon näkymätöntä päättelyä käyttäessään kyseistä heuristiikkaa. Lisäksi ei ole luultavaa, että täysin erilaisella sanaluokkien kuvaamisella saataisiin järkevää lopputulosta, joten heuristiikan vaihtaminen ei loppujen lopuksi ole kauhean tärkeä ominaisuus. Jälkikäteen ajateltuna parempi lopputulos olisi saavutettu käyttämällä Abbotin heuristiikkaa lähtökohtana oman koneellisen analysointimenetelmän kehittämiseksi. Tällä hetkellä ohjelma saa jäsennin-modullilta sanaluokkatietoa, mutta parempi vaihtoehto olisi saada merkitykseen pohjautuvaa tietoa.

8 Lähteet

- [1] Ilari Liukko, Tuomo Pieniluoma, Vesa Pikki ja Panu suominen, "UCOT-Sovellusprojekti Käyttöohje", Jyväskylän yliopisto, tietotekniikan laitos, 2006.

A Termit

Alkuperäinen käyttötapaus	on lähteen sisältämä käyttötapaus.
Core	on sovelluksen ydin, joka ohjaa ohjelman muiden komponenttien toimintaa.
Entiteetti	on vaatimusmäärittelyssä esiintyvä toimija tai toimenpiteen kohde. Käytännössä mikä tahansa substantiivi voi olla entiteetti.
HeuristicCollection	on luokka, joka säilöö <code>HeuristicModule</code> ja.
HeuristicInterface	on rajapinta, joka määrittää miten <code>HeuristicModule</code> en kanssa kommunikoidaan.
HeuristicModule	on luokka, joka suorittaa heuristiikan sille annetulle jäsenetylle käyttötapaukselle ja palauttaa käsitellin.
InputAdapter	on luokka, joka lataa URL:llä osoitetun lähteen sisältämät alkuperäiset käyttötapaukset ja palauttaa ne jäsentämättöminä käyttötapauksina.
InputInterface	on rajapinta, joka määrittää miten <code>InputAdapter</code> in kanssa kommunikoidaan.
InputCollection	on luokka, joka säilöö input adaptereita.
Iteraatio	tarkoittaa yleisesti jonkin asian toistamista uudelleen siten, että edellisen suorituskerran tulos on seuraavan kerran syöte. Sovelluskehityksessä iteraatiolla tarkoitetaan projektin suorittamista pienissä paloissa edellisen iteraation tulosten toimiesä seuraavan iteraation toteutuksen pohjana. Tuloksilla tässä tapauksessa tarkoitetaan kaikkea ohjelmiston kehityksen tuottamaa materiaalia eikä vain lähdekoodia. Peräkkäiset iteraatiot eivät välttämättä käytä ollenkaan samaa lähdekoodia vaan koodi voidaan välillä kirjoittaa uudestaan.

Jäsennetty käyttötapaus	on käyttötapaus, jolle on suoritettu morfologinen jäsenitys.
Jäsentämätön käyttötapaus	on käyttötapausten suoritusaskeleet tekstimuodossa. Käyttötapaus on jo otettu sisään järjestelmään, mutta sitä ei ole vielä toimitettu parserille.
Käsitemalli	on heuristiikan muodostama malli jäsenetystä käyttötapauksesta.
Käyttötapaus	on kuvaus järjestelmän ja sen käyttäjän välisestä vuorovaikutuksesta tietyn tuloksen aikaansaamiseksi.
Käyttötapausten muoto	kertoo, mitä attribuutteja ja missä järjestyksessä sekä muodossa yksittäisen käyttötapausten kuvaus sisältää. Näitä attribuutteja ovat mm. tiedot pääaktorista ja muista aktoreista, tietoa järjestelmän tilasta ennen ja jälkeen käyttötapausten toiminnan sekä käyttötapausten suoritusaskeleet.
LaTeX 2ϵ	on ladontaohjelmisto, millä tämäkin dokumentti on tehty.
Moduuli	on ohjelman osa, joka piilottaa varsinaisen toiminnan toteutuksen sisäänsä. Hyvin kirjoitetun moduulin sisäistä toteutusta on helppo muuttaa. Yleensä moduuli toteuttaa jonkin rajapinnan vaatiman toiminnallisuuden. Esimerkiksi UCOT-ohjelmistossa eri läheteistä tullutta dataa voidaan lukea kun vain datan lukemista varten on toteutettu moduuli, joka täyttää UCOT-ohjelmiston syöterajapinnan määrittelyn.
Output	on luokka, joka hoitaa heuristiikan tuottaman käsitemallin esittämisen/tallentamisen.
OutputCollection	on luokka, joka säilöö Outputteja
OutputInterface	on rajapinta, joka määrittää outputin kanssa kommunikoidaan.

Parser	tarkoittaa morfologista jäsenointiä.
ParserAdapter	on luokka joka toteuttaa <code>ParserInterface</code> ja kommunikoi parserin kanssa. Ottaa vastaan jäsenoittämättömän käyttötapauksen ja palauttaa jäsenoitetyn käyttötapauksen.
ParserCollection	on luokka, joka säilöo <code>ParserAdapter</code> ereita.
ParserInterface	on rajapinta, joka määrittää miten <code>ParserAdapter</code> erin kanssa kommunikoidaan.
Projekti	tarkoittaa tämän dokumentin yhteydessä sovellusprojektiä.
Rajapinta	erottaa kaksi toisistaan erillistä ohjelman osaa toisistaan siten, että osat tietävät vain osan toisen toiminnallisuudesta. Tällöin rajapinnan takan olevaa osaa voidaan vaihtaa toisen osan häiriintymättä.
Sovellusprojekti	on tietotekniikan laitoksen opintojakso.
Syöte	on ohjelman vastaanottama data.
Syötemoduuli	on ohjelman osa, joka lukee ohjelmalle tarkoitetun syöteen ja palautta ohjelmalle jäsenoitetyn käyttötapauksen. Rakentuu <code>InputAdapter</code> ista ja <code>ParserAdapter</code> ista.
Tuloste	on ohjelman tuottama data.
UI	tarkoittaa käyttöliittymää (<i>user interface</i>). Tarkentuu myöhemmissä iteraatioissa.
UIInterface	on käyttöliittymän rajapinta.
UCOT	on tämän sovellusprojektin toteuttava ryhmä.
Vaikutussuhde	on kahden entiteetin välillä vallitseva suhde, jossa toinen käyttää toista.

B Javadoc