O'Haskell

Kosti Kuokkanen
koalkuok@jyu.fi

20.4.2008

Tiivistelmä

O'Haskell osoittaa, että olio-ohjelmointi ja funktionaalinen ohjelmointi eivät sulje toisiaan pois. Tässä raportissa luodaan katsaus olioiden ja funktionaalisten kielten yhdistämiseen, erityisesti O'Haskelliin, sekä esitetään yksinkertaisien esimerkkien avulla miten oliot sulautuvat funktionaaliseen kieleen käytännössä. Lisäksi esitellään lyhyesti reaktiivinen olio.

1 Johdanto

Kipinä O'Haskellin luomiseen syntyi Johan Nordlanderin 1990-luvun alussa käymien kahvipöytäkeskustelujen aikana. Keskustelujen aiheena olivat ihanteellisen ohjelmointikielen ominaisuudet. Vuonna 1999 Nordlander konkretisoi ihanteelliset nykyaikaisen ohjelmointikielen ominaisuudet väitöskirjassaan. Väitöskirjassa todistettiin myös, että kieli, joka toteuttaa nämä ominaisuudet on mahdollista tehdä. Todistaminen tapahtui luomalla O'Haskell. [1]

Kuten O'Haskell-nimestä voi päätellä, ovat sen juuret syvällä Haskell-kielessä. O'Haskellin O tulee sekä object-sanasta että I/O:sta, josta on poistettu "paha" I. [2]

Nordlander käyttää hajautettua sovellusta ja hajautetun sovelluksen kehittämisen haastavuutta yhtenä perusteena suunnittelemansa kielen ominaisuus- ja toteutusvalinnoille. Hajautetun sovelluksen kehittämisessä etenkin järjestelmän reaktiivisuuden ja rinnakkaisuuden hallintaan tulee kiinnittää erityistä huomiota. O'Haskell on suunniteltu etenkin näiden ongelmien helppoa ratkaisua silmällä pitäen.

O'Haskell on moniparadigmainen kieli, jossa yhdistyy funktio-, reaktiivisen- ja olio-ohjelmoinnin ideoita. Se voidaankin käsittää kolmen erillisen kielen yhdistelmänä, jonka osat ovat:

- Olioilla ja alityypityksellä (engl. subtyping) laajennettu funktionaalinen kieli Haskell.
- Imperatiivinen olio-ohjelmointikieli. Yleisistä olio-ohjelmointikielistä poiketen, O'Haskell tarjoaa Haskellista tutut ominaisuudet parametrien polymorfisuuden ja automaattisen tyyppien päättelyn.
- Rinnakkaistamisen mahdollistava kieli (engl. concurrent language), jossa on reaktiivinen kommunikointimalli, sekä asynkroninen että synkroninen viestinvälitysmahdollisuus.[2]

2 Oliot funktionaalisissa kielissä

Viittausten läpinäkyvyyden mahdollistama ohjelmien formaali todistaminen ja laiska laskenta ovat vahvoja deklaratiivisen ohjelmoinnin puolestapuhujia. Oikean elämän ongelmien kuvaaminen vaatii kuitenkin nopeasti tilojen kuvaamista ja käsittelyä. Tällöin valinta kohdistuu johonkin olio-ohjelmointikieleen ja imperatiiviseen ohjelmointiin.

Sovelluskehityksessä tarvitaan usein molempien ohjelmointitapojen ominaisuuksia. Kuitenkin deklaratiivinen ja imperatiivinen ohjelmointi kuvataan usein toistensa vastakohdiksi. Miten näiden ohjelmointitapojen yhdistäminen voidaan toteuttaa?

Kielet, jotka tukevat sekä funktionaalista että olio-ohjelmointia, voidaan jakaa kahteen kategoriaan sen mukaan, miten niissä käsitellään tilaa ja I/O:ta. [5]

- **Hybridi**-kielet, joissa tuetaan funktionaalista ohjelmointia, mutta pohjautuvat imperatiiviseen kieleen. Funktioilla voi olla mahdollisesti sivuvaikutuksia. Tällaisia kieliä ovat esim. Python ja C# 3.0
- **Virheettömät**-kielet (engl. pure), joissa tuetaan olio-ohjelmointia, mutta pohjautuvat deklaratiiviseen kieleen. Tällaisia kieliä ovat esim. O'Haskell ja O'Haskellista juontuva Timber.

O'Haskellin vastaus funktionaalisen ja olio -maailmojen yhdistämiseen ovat reaktiiviset oliot.[1] Kaiken O'Haskell-koodin pystyy, alityypitystä lukuunottamatta, muuttamaan myös validiksi Haskelliksi. Muunnoksen jälkeen oliot pystytään kuvaamaan seuraavien seitsemän määrityksen avulla:

```
return :: t -> O s t
(»=) :: O s a -> (a -> O s b) -> O s b
get :: s -> O s ()
set :: O s s
```

Loput kolme määrittelyä voidaan käsittää sellaisen abstraktin datatyypin operaatioina, joka määrittelee Ref-tyyppikonstruktorin.

```
new :: (Ref s -> s) -> (Ref s -> t) -> Template t
act :: Ref s -> O s () -> Action
req :: Ref s -> O s t -> Request
```

Jossa,

`new s e` luo uuden olion, jolla on yksilöllinen viitearvo n, alustaa olion tilan arvolla $s\ n$ ja palauttaa $e\ n$.

`act n e` käskee oliota n suorittamaan käskyn e heti kun n kerkeää ja palauttaa ().

`req n e` käskee oliota n suorittamaan käskyn e heti kun n kerkeää, odottaa käskyn suorittamista ja palauttaa sen tuloksen.

3 Tietueet

Vaikka Haskell tarjoaa vastikkeen tietueelle, on O'Haskellissa päädytty tekemään tietueet omalla toteutuksella. Omaan toteutukseen päädyttiin, jotta alityypitys olisi helpompi toteuttaa.

Seuraavassa esimerkissä määritellään Alus-tietue, joka koostuu yhdestä kokonaisluvusta ja merkkijonosta. Luodaan lisäksi muutama tietue.

```
struct Alus =
  nopeus :: Int
  nimi :: String

vene =  struct
          nopeus = 5
          nimi = "Paatti"
laiva = struct
          nopeus = 10
          nimi = "HMAS Sydney"
```

Tietueiden luonnin yhteydessä ei tarvitse kertoa minkälaista tietuetta ollaan luomassa, vaan se päätellään yksikäsitteisten tietuiden termien perusteella. Tietueiden termejä päästään käsittelemään perinteisen pistesyntaksin avulla. Haskellin piste-operaattoria voi yhä käyttää, mutta se vaatii ympärilleen välilyönnit. Seuraavassa esimerkissä käytetään pistesyntaksia.

```
:laiva.nimi
"HMAS Sydney"
:map (\x -> x.nopeus * 2) [vene, laiva]
[10,20]
```

4 Tietueesta olioksi

Jotta nopeuden muutos pysyisi tallessa, on Aluksesta tehtävä olio. Tietueessa nimetään luokan metodit, joiden varsinainen toimina määritellään vasta myöhemmin. Oliot luodaan *template* määritelmän avulla, jossa samalla annetaan oliolle alkutila ja määritellään olion viestintärajapinta (engl. communication interface). Viestintärajapinta on käytännössä yhtä kuin luokan metodit, mutta viestintärajapintaa käsitellään yhtenä arvona. *:=*-operaatiolla sijoitetaan arvo muuttujalle.[4] Riviltä 10 alkaen alkaa varsinaisten metodien määrittely. *action* ja *request*-avainsanoilla määritellään metodin synkroonisuuden tyyppi, josta lisää luvussa 5.

```
01 struct Alus =
02   kiihdyta :: Int -> Cmd ()
03   getNopeus :: Cmd Int
04   getNimi :: Cmd String
05
06 alus n= template
07           nopeus := 0
08           nimi := n
09        in struct
10           kiihdyta i = action
11             nopeus := nopeus + i
12           getNopeus = request
13             return nopeus
14           getNimi = request
15             return nimi
```

5 Reaktiivisuus ja rinnakkaisuus

Reaktiivinen olio yhdistää perinteisen olion ja prosessin ideoita. Jokainen reaktiivinen olio on autonominen suoritusyksikkö ts. sijaitsee omassa säikeessä. Reaktiivinen olio voi joko suorittaa täsmälleen yhtä sen omaa metodia tai odottaa passiivisena metodikutsua ja ylläpitää tilaansa. Reaktiivisten olioiden sijaitseminen omissa säikeissään on yhtenevä intuitiivisen käsityksen kanssa oikeasta elämästä.[6] Omista säikeistä johtuen, on mielekästä määritellä asynkroniset ja synkroniset metodikutsut erikseen. Määrittelemällä metodi *action*-avainsanalla on se asynkroninen ja *request*-avainsana tekee siitä synkronisen. Synkronisesti kutsuttaessa, kutsuvan olion säie ei jatka suoritusta ennen kuin kutsuttava metodi on suoriutunut. Asynkronisesti kut-

suttaessa, kutsuvan olion säie jatkaa suoritusta heti kutsun jälkeen, riippumatta siitä onko kutsutun metodin suoritus valmis.

6 Alityypitys

O'Haskellissa voi luoda tietueille ali- ja ylityyppejä. (engl. subtyping) Luokkia ei voi periä, mutta eräänlaista luokkien monimuotoisuutta O'Haskell kuitenkin tukee.[4] Seuraavassa esimerkissä luodaan alityyppi Alus-tietueelle.

```
struct AvaruusAlus < Alus =
  miehisto :: Int
```

Ali- ja ylityypitykset toimivat myös Haskellin perinteisille tietotyypeille. Seuraavassa esimerkissä *Color* on BW:n ylityyppi ja se sisältää myös BW:n konstruktorit.

```
data BW = Black
        | White

data Color > BW =
   Red | Orange | Yellow | Green | Blue | Violet
```

Luokkien monimuotoisuus tarkoittaa erilaisten *template*-määrittelyjen tekemistä samoille metodeille. Perinteistä olio-ohjelmoinnin perintää, jossa ylikirjoitetaan metodeja, ei tueta. Erinäisillä kiertoteillä osa perinnän tavoitteisa voidaan kuitenkin saavuttaa myös O'Haskellin tarjoamin keinoin.[4] Seuraavassa esimerkissä luodaan Alus-olio uudella, erilaisella *template* määrittelyllä. Useita eri määrittelyjä voidaan käyttää yhtä aikaa. Samasta tietue määrittelystä voidaan luoda siis erilaisesti käyttäytyviä olioita luomalla vain uusia *template*-määrittelyjä.

```
alus2 n= template
          nopeus := 0
          nimi := n
       in struct
          kiihdyta i = action
            nopeus := nopeus + i + 5
          getNopeus = request
            return nopeus
          getNimi = request
            return nimi
```

7 Yhteenveto

Tässä raportissa tutustuttiin pintaa raapaisten O'Haskell-ohjelmointikielen ominaisuusvalintoihin ja toteutusratkaisuihin. Tärkeimpänä uudistuksena Haskelliin nähden ovat tietueet, oliot, tietuiden alityypitys, luokkien monimuotoisuus, sekä asynkronisuuden mahdollistava rinnakkaistamismalli. O'Haskell ei ole valtavirran ohjelmointikieli, joskin ei sitä sellaiseksi ole tarkoitettukaan. O'Haskell kuitenkin osoittaa, että olio-ohjelmoinnin voi yhdistää funktionaaliseen kieleen säilyttäen kaikki funktionaalisen kielen vahvat ominaisuudet. Katseet kannattaa suunnata samoilta tekijöiltä tulevaan Timber kieleen. Ehkä se lyö itsensä läpi valtavirrassa.

Lähteet

[1] Johan Nordlander, *Reactive Objects and Functional Programming*, Ph.D. thesis, Dept. of Computing Science, Chalmers University of Technology, Göteborg, Ruotsi, 1999

[2] Johan Nordlander, *The O'Haskell homepage*, saatavilla WWW-muodossa <http://www.cs.chalmers.se/ nordland/ohaskell/>, viitattu 16.4.2008

[3] Johan Nordlander, *Rationale for O'Haskell*, saatavilla WWW-muodossa <http://www.cs.chalmers.se/ nordland/ohaskell/rationale.html>, viitattu 17.4.2008

[4] Johan Nordlander, *A survey of O'Haskell*, saatavilla WWW-muodossa <http://www.cs.chalmers.se/ nordland/ohaskell/survey.html>, viitattu 20.4.2008

[5] C2-wiki, *Object Functional*, saatavilla WWW-muodossa <http://c2.com/cgi/wiki?ObjectFunctional>, viitattu 19.4.2008

[6] Johan Nordlander, Mark P. Jones, Magnus Carlsson, Richard B. Kieburtz, and Andrew Black, *Reactive Objects*, OGI School of Science & Engineering at OHSU, 20000 NW Walker Road, Beaverton, OR 97006, saatavilla WWW-muodossa <http://www.cse.ogi.edu/PacSoft/projects/Timber/reactive-objects.pdf>, viitattu 19.4.