Funktionaalinen ohjelmointi front-end web-kehityksessä

TkK-tutkielma
Turun yliopisto
Tulevaisuuden teknologioiden laitos
Ohjelmistotekniikka
2019
Konsta Purtsi

TURUN YLIOPISTO
Tulevaisuuden teknologioiden laitos

Konsta Purtsi: Funktionaalinen ohjelmointi front-end web-kehityksessä

TkK-tutkielma, 12 s.
Ohjelmistotekniikka
Kesäkuu 2019

Tarkempia ohjeita tiivistelmäsivun laadintaan läytyy opiskelijan yleisoppaasta, josta alla lyhyt katkelma.
Bibliografisten tietojen jälkeen kirjoitetaan varsinainen tiivistelmä. Sen on oletettava, että lukijalla on yleiset tiedot aiheesta. Tiivistelmän tulee olla ymmärrettävissä ilman tarvetta perehtyä koko tutkielmaan. Se on kirjoitettava täydellisinä virkkeinä, väliotsakeluettelona. On käytettävä vakiintuneita termejä. Viittauksia ja lainauksia tiivistelmään ei saa sisällyttää, eikä myäskään tietoja tai väitteitä, jotka eivät sisälly itse tutkimukseen. Tiivistelmän on oltava mahdollisimman ytimekäs n. 120–250 sanan pituinen itsenäinen kokonaisuus, joka mahtuu ykkäsvälillä kirjoitettuna vaivatta tiivistelmäsivulle. Tiivistelmässä tulisi ilmetä mm. tutkielman aihe tutkimuksen kohde, populaatio, alue ja tarkoitus käytetyt tutkimusmenetelmät (mikäli tutkimus on luonteeltaan teoreettinen ja tiettyyn kirjalliseen materiaaliin, on mainittava tärkeimmät lähdeteokset; mikäli on luonteeltaan empiirinen, on mainittava käytetyt metodit) keskeiset tutkimustulokset tulosten perusteella tehdyt päätelmät ja toimenpidesuositukset asiasanat

Asiasanat: tähän, lista, avainsanoista

Sisältö

1 Johdanto

1.1 Motivaatio

1.2 Tutkimuskysymys

1.3 Metodit

1.4 Rakenne

2 Funktionaalinen ohjelmointi

2.1 Määritelmä

Funktionaalinen ohjelmointi on deklaratiivinen ohjelmointiparadigma, jossa ohjelman suoritus tapahtuu sisäkkäisten matemaattisten funktioiden evaluoimisena. Deklaratiivinen ohjelmointi on ohjelmointiparadigma, jossa kuvaillaan ohjelman lopputulos tai tila. Deklaratiivisessa ohjelmoinnissa ei ole epätarkkaa globaalia tilaa, ja siten ohjelma koostuu lausekkeista, jotka voivat sisältää lokaalia epätarkkaa tilaa. Se on vastakohta perinteisemmälle imperatiiviselle ohjelmoinnille. Funktionaalinen ohjelmakoodi pyrkii käsittelemään tilaa tarkemmin ja välttämään muuttuvia arvoja. [1]

Puhtaasti funktionaalinen ohjelmointikieli ei salli lainkaan muuttuvaa tilaa tai datan muokkaamista. Tämä vaatii sen, että funktioiden paluuarvo riippuu ainoastaan funktion saamista parametreista, eikä mikään muu kuin parametrit voi vaikuttaa funktion paluuarvoon. Funktionaaliseksi ohjelmointikieleksi lasketaan kuitenkin myös moniparadigmaiset ohjelmointikielet, jotka sisältävät funktionaalisia piirteitä. Yksinkertaisten loogisten toimintojen lisäksi realistinen ohjelmointikieli vaatii epäpuhtauksia muun muassa siirrännän (eng. Input/Output) toteuttamiseen. [2] Moderneissa ohjelmointikielissä tähän on myös funktionaalisia ratkaisuja, esimerkiksi Haskell-kielen IO-monadi [3].

2.2 Lambdakalkyyli

Funktionaalinen ohjelmointi perustuu lambdakalkyyliin - formaalin laskennan malliin jonka kehitti Alonzo Church vuonna 1928. Lambdakalkyyliä pidetään ensimmäisenä funktionaalisena ohjelmointikielenä, vaikka se keksittiinkin ennen ensimmäisiä varsinaisia tietokoneita, eikä sitä pidettykään aikanaan ohjelmointikielenä. Useita moderneja ohjelmointikieliä, kuten Lispiä, pidetään abstraktioina lambdakalkyylin päälle. Lambdakalkyyli voidaan kirjoittaa myös λ-kalkyyli kreikkalaisella lambdakirjaimella. [1]

Churchin lambdakalkyyli koostuu vain kolmesta termistä: muuttujista, abstraktioista ja sovelluksista. Muuttujat ovat yksinkertaisesti merkkejä tai merkkijonoja jotka kuvaavat jotain parametriä tai arvoa. Churchin alkuperäinen lambdakalkyyli ei tuntenut muuttujien asettamista, ainoastaan arvojen syöttämisen parametrina. Abstraktio $\lambda x[M]$ määrittelee funktion parametrille x toteutuksella M. Sovellus $\{F\}(X)$ suorittaa funktion F arvolla X. [4]

2.3 Konseptit

2.3.1 Tilattomuus

Deklaratiivisissa ohjelmointikielissä ei ole implisiittistä tilaa, mikä eroaa imperatiivisista kielistä, joissa tilaa voi muokata lauseilla (eng. commands) ohjelmakoodissa. Tilattomuus tekee laskennasta mahdollista ainoastaan lausekkeilla (eng. expression). Funktionaaliset ohjelmointikielet käyttävät lambda-abstraktioita kaiken laskennan perustana. Esimerkiksi logiikkaohjelmoinnissa vastaava rakenne on relaatio. Tilattomuus sallii vain muuttumattomat vakiot, mikä estää muuttujien arvon muuttamisen ja perinteiset silmukat. Puhtaasti funktionaalisessa ohjelmoinnissa rekursio on ainoa tapa toteuttaa toisto. Esimerkiksi funktio, joka laskee luvun n kertoman voitaisiin

toteuttaa imperatiivisesti Python-kielessä esimerkiksi näin:

```
def factorial(n):
    result = 1
    while n >= 1:
        result = result * n
        n = n - 1
    return result
```

Funktionaalisessa ohjelmointikielessä toistorakenne on toteutettava rekursion avulla, mutta tästä on usein tehty vaivatonta muun muassa yksinkertaisen syntaksin ja tehooptimoinnin avulla. [1] Funktio, joka laskee luvun n kertoman voidaan toteuttaa esimerkiksi näin funktionaalisesti Haskell-kielessä:

```
factorial :: Int -> Int
factorial n = if n < 2
              then 1
              else n * factorial (n-1)
```

2.3.2 Korkeamman asteen funktiot

Funktionaaliset ohjelmointikielet helpottavat funktionaalista ohjelmointityyliä muun muassa sallimalla korkeamman asteen funktiot. Ensimmäisen luokan kansalainen on ohjelmointikielen rakenne, jonka voi syöttää parametrina funktioille, asettaa paluuarvoksi funktiolle ja tallentaa tietorakenteisiin. Korkeamman asteen funktio eroaa tavallisesta funktiosta siten, että se voi ottaa parametrinaan tai palauttaa funktion. Ohjelmointikielissä, jotka tukevat korkeamman asteen funktioita, funktiot ovat ensimmäisen luokan kansalaisia. Korkeamman asteen funktioiden avulla ohjelmakoodi ja data ovat jossain määrin vaihdettavissa, joten niiden avulla voidaan abstrahoida kompleksisia rakenteita. [1] Map-funktio on korkeamman asteen funktio, joka suorittaa parametrina annetun funktion jollekkin tietorakenteelle, esimerkiksi listalle.

Map-funktiota käytetään usein modernissa front-end web-ohjelmoinnissa [5]. Listan renderöinti React-kirjastolla voidaan toteuttaa näin map-funktion avulla:

```
items.map(item => <Item {...item} />)
```

2.3.3 Laiska laskenta

Laiska laskenta (eng. lazy evaluation) on funktion laskentastrategia, jossa lausekkeen arvo lasketaan vasta kun sitä ensimmäisen kerran tarvitaan, mutta ei aikaisemmin. Laiska laskenta on päinvastainen laskentastrategia innokkaaseen laskentaan (eng. eager evaluation), jossa lausekkeen arvo lasketaan heti ensimmäisellä kerralla, kun lauseke esitellään. Yleisimmät ohjelmointikielet käyttävät innokasta laskentastrategiaa. Laiska laskenta vähentää suoritukseen kuluvaa aikaa ja siten voi parantaa ohjelman suorituskykyä eksponentiaalisesti esimerkiksi call by name -laskentastrategiaan verrattuna, jossa lausekkeen arvo voidaan joutua laskemaan useita kertoja. Laiskan laskennan toteuttavat useat puhtaasti funktionaaliset ohjelmointikielet, kuten Haskell. Myös jotkin moniparadigmaiset kielet toteuttavat laiskan laskennan, esimerkiksi Scala-kieli mahdollistaa sen *lazy val*-lausekkeen avulla. [6]

Laiska laskenta mahdollistaa päättymättömät tietorakenteet. Niin sanottujen laiskojen listojen loppupää evaluoidaan vasta kun sitä kutsutaan. Tämä näkyy useissa funktionaalisissa ohjelmointikielissä listan toteutuksena, jossa listaa evaluoidaan alkupäästä loppua kohti. Esimerkiksi Haskellissa voidaan määrittää lista kaikista kokonaisluvuista tietotyypin rajoissa alkaen luvusta n:

```
from :: Int -> [Int]
from n = n : from (n+1)
```

Funktion rekursiivinen kutsu aiheuttaisi innokkaasti laskevissa ohjelmointikielissä päättymättömän rekursion, mutta Haskellin tapauksessa vain määrätty osa listasta evaluoidaan. [6]

Funktionaalisissa ohjelmointikielissä häntärekursio ei aiheuta pinon ylivuotoa, joten päättymätön rekursio on toimiva vaihtoehto päättymättömälle silmukalle. Esimerkiksi Haskell-kielen main-funktioita voidaan toistaa äärettömästi näin:

```
main :: IO ()
main =
  do
    putStrLn "do something"
    main
```

2.3.4 Hahmonsovitus

Hahmonsovitus (eng. pattern matching) on funktionaaliselle ohjelmoinnille tyypillinen piirre, jossa sama funktio voidaan määritellä useita kertoja. Funktiomäärittelyistä vain yhtä sovelletaan tapauskohtaisesti. Modernit funktionaaliset ohjelmointikielet, kuten Haskell, mahdollistavat monimutkaisten tyyppien ja lausekkeiden dekonstruoimisen hahmonsovituksen avulla [3]. Hahmonsovituksen toteutus on käytännössä case-lauseke, jossa lausekkeen ehto kuvaa syötetyn parametrin tietotyyppiä tai sen rakenneta. Tämän avulla ohjelman suoritus jakautuu syötetyn muuttujan tyypin tai rakenteen mukaan. [1]

Esimerkiksi kertomafunktio voidaan toteuttaa rekursiivisesti hahmonsovituksen avulla. Tässä esimerkissä rekursion alkuarvo määritetään syötteelle *0*, ja muut syötteet käsitellään parametrina *n*.

```
factorial :: Int -> Int
factorial 0 = 1
factorial n = n * factorial (n - 1)
```

2.4 Funktionaalisen laskentamallin seuraukset

Puhtaasti funktionaalinen laskentamalli karsii ohjelmasta pois tietyt bugityypit. Puhdas sivuvaikutukseton laskenta on automaattisesti säieturvallista (eng. thread safe) tarkoittaen, ettei sitä tarvitse manuaalisesti synkronoida. Koska puhtaan funktion lopputulos riippuu ainoastaan syötteistä, ei syötteiden laskujärjestyksellä ole väliä. Laskennan lopputulosta voidaan siten pitää deterministisenä, mikä helpottaa monisäikeistystä.

Toistettavuus (eng. reproducibility) tarkoittaa, että ohjelman tietynlainen toiminta on toistettavissa. Deterministisyys ja toistettavuus parantavat ohjelman testattavuutta. Jos puhdas funktio toimii tietyllä tavalla kerran, se toimii niin myös joka kerta ajon aikana. Toistettavuus helpottaa ajon aikana löydetyn virheellisen tilan toistamista syöttämällä sama tarkkaan määritetty virheellinen tila testiympäristöön. [7]

Funktionaaliset ohjelmointikielet ovat olleet hitaampia kuin niiden perinteisemmät imperatiiviset sukulaiset, kuten C-kieli, mutta ero on ajan kuluessa kaventunut. Kääntäjäoptimoinneista huolimatta imperatiiviset ohjelmointikielt tulevat aina olemaan optimoidumpia imperatiivisella laitteistolla. Imperatiivisille tietorakenteille on nykyään olemassa monia funktionaalisia vaihtoehtoja, jotka voivat parantaa ohjelman suoritusaikaa ja säästää muistia. [8]

3 Web-kehitys

3.1 World Wide Web

World Wide Web tai WWW on järjestelmä tiedon jakamiseen, joka käyttää Internet-verkkoa. Web-sisältöä luetaan selaimella, joka hakee sisällön web-palvelimelta HTTP-siirtoprotokollan avulla. Verkkosivujen sisällön kuvaamiseen on perinteiseti ollut kolme tekniikkaa: HTML-kieli sivun sisällön kuvaamiseen, CSS-kieli sivun ulkoasun kuvaamiseen ja JavaScript-kieli sivun toiminnallisuuden toteuttamiseen. On myös mahdollista käyttää esimerkiksi Flash- ja Java-liitännäisiä, mutta nykyään on tyypillisempää käyttää pelkästään JavaScript-kieltä. [9]

3.1.1 Staattiset verkkosivut

Alkuperäinen tapa jakaa web-sisältöä on staattisena verkkosivuna (eng. static website). Staattinen verkkosivu koostuu HTML-sivuista, dokumenteista ja mediasta, jotka luetaan suoraan web-palvelimen muistista, ilman että palvelin tekee niihin muutoksia ajon aikana. Staattiset verkkosivut sisältävät usein HTML-, CSS- ja multimediasisältöä, sekä selaimessa ajettavia skriptejä tai ohjelmia, jotka on kirjoitettu JavaScript-kielellä tai muulla selaimen tulkitsemalla kielellä. [10]

3.1.2 Dynaamiset verkkosivut

Dynaaminen verkkosivu tarkoittaa sivua, joka ei ole suoraan selaimen luettavassa muodossa. Dynaamisen sivun back-end muodostaa selaimella renderöityvän sivun ajon aikana. Dynaamisen verkkosivun operaatioiden toteutus ei olekaan rajattu selaimen tulkittaviin ohjelmointikieliin, vaan käytännössä minkä tahansa kielen käyttäminen on mahdollista. Yleisimpiä kieliä dynaamisen verkkosivun toteuttamiseen ovat PHP, Python, Perl, Ruby, Java, C# ja Node.js. Dynaaminen verkkosivu tallettaa useimmiten sisältönsä tietokantaan. [10]

3.2 Back-end

Back-end tarkoittaa verkkosivun tai -sovelluksen palvelinpäässä ajettavaa osaa, joka ei näy käyttäjälle. Back-end kommunikoi front-endin kanssa jonkin yhteisen rajapinnan (eng. Application programming interface, API) kautta. Koska backend-ohjelmaa ajetaan palvelinlaitteistolla, sen toteuttamiseen on paljon enemmän vapautta kuin front-endin, jonka täytyy olla selaimen tulkittavissa. Back-end vastaa usein kaikesta varsinaisesta toiminnallisuudesta verrattuna front-endiin, joka toteutaa pelkän käyttöliittymän ohjelmalle. Backendiin kuuluu usein myös tietokanta, johon sovelluksen data tallennetaan. [11]

3.3 Front-end

Front-end tarkoittaa verkkosivun tai -sovelluksen käyttäjälle näkyvää osaa. Koska front-endiä suoritetaan selaimessa, se on pakko toteuttaa web-yhteensopivilla tekniikoilla, joista yleisimmät ovat HTML, CSS ja JavaScript. JavaScript on nykypäivänä täysiverinen ohjelmointikieli, jolla voidaan toteuttaa interaktiivisuutta sekä manipuloida verkkosovelluksen rakennetta ja ulkoasua. [11]

3.3.1 Käännöstyökalut

Modernissa front-end–kehityksessä on tyypillistä toteuttaa verkkosovelluksen ohjelmakoodi jollain korkeamman tason ohjelmointikiellä, joka muunnetaan web-natiiviksi käännösvaiheessa. Usein käytetään esimerkiksi CSS-tyylikieleksi kääntyvää SCSS-kieltä. Kääntäjiä, jotka kääntävät front-end–kieliä selainyhteensopiviksi kutsutaan käännöstyökaluiksi (eng. build tool). Esimerkiksi tämän tutkielman käsittelykohteena oleva Elm-kieli ei ole sellaisenaan selainyhteensopiva, vaan vaatii kääntämisen HTML- ja JavaScript-kielille [12]. [11]

3.3.2 Funktionaalinen reaktiivinen ohjelmointi

3.4 Tutkimuskohteet

3.4.1 Funktionaalinen ohjelmointi front-end kehityksessä

4 Tutkittavat teknologiat

4.1 React

4.2 Elm

5 Loppupäätelmät

Lähdeluettelo

[1] P. Hudak, "Conception, Evolution, and Application of Functional Programming Languages," 1989.

[2] A. Sabry, "What is a purely functional language?" *Journal of Functional Programming*, 1998.

[3] M. Lipovaca, "Learn You a Haskell for Great Good!: A Beginner's Guide," 2011.

[4] F. Cardone ja J. R. Hindley, "History of Lambda-calculus and Combinatory Logic," *Handbook of the History of Logic, vol. 5*, 2006.

[5] A. Banks ja E. Porcello, "Learning React: Functional Web Development with React and Redux," 2017.

[6] D. A. Watt, "Programming Language Design Concepts," 2006.

[7] M. Finifter, A. Mettler, N. Sastry ja D. Wagner, "Verifiable Functional Purity in Java," 2008.

[8] C. Okasaki, "Purely Functional Data Structures," 1996.

[9] D. Flanagan, "JavaScript – The definitive guide (6 ed.)," 2016.

[10] H. Pettersen, "From Static and Dynamic Websites to Static Site Generators," 2016.

[11] C. Northwood, "The Full Stack Developer," 2018.

[12] E. Czaplicki, "Elm website," 2020.