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 algoritmi kuvataan varsinaisen kontrollivuon sijaan globaalia tilaa muokkaavin funktiokutsuin. 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 paluuarvoo riippuu vain ja ainoastaan funktion saamista parametreista, eikä mikään tila voi vaikuttaa funktion paluuarvoon. Simppelien loogisten toimintojen lisäksi tosielämän ohjelmointikielet vaativat kuitenkin joitain epäpuhtauksia ollakseen oikeasti hyödyllisiä.[2]

2.2 Lambdakalkyyli

Funktionaalinen ohjelmointi perustuu vahvasti 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 ohjelmointikiele-

nä. Useita moderneja ohjelmointikieliä, kuten Lispiä, pidetään abstraktioina lambdakalkyylin päälle. Lambdakalkyyli voidaan kirjoittaa myös λ-kalkyyli kreikkalaisella lambda-kirjaimella.[1]

Yksinkertaisimmillaan 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. Hänen lambdakalkyylinsä primitiivitermit olivat abstraktio ja sovellus:

Syntaksi	Nimi	Kuvaus
$\lambda x[M]$	Abstraktio	Määrittelee funktion parametrille x toteutuksella M
$\{F\}(X)$	Sovellus	suorittaa funktion F arvolla X

[3]

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 perustuvat lambdakalkyyliin ja siten käyttävät ja lambda-abstraktioita kaiken laskennan perustana. Esimerkiksi logiikkaaohjelmoinnissa 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 Pythonkielessä 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. Funktionaaliset ohjelmointikielet sallivat funktioiden rekursion ja tekevät usein siitä myös vaivatonta.[1] Funktio, joka laskee luvun n kertoman voidaan toteutta 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 rohkaisevat ohjelmoimaan funktionaalisesti muun muassa sallimalla korkeamman asteen funktiot. Tämä tarkoittaa, että funktioita kohdellaan ensimmäisen luokan kansalaisina ja niitä voi syöttää parametrina funktioille, asettaa paluuarvoksi funktiolle ja tallentaa tietorakenteisiin. 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 tietorakennteelle, esimerkiksi listalle. Map-funktiota käytetään usein modernissa front-end web-ohjelmoinnissa [4]. 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. Tämä voi 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. Innokas laskenta (eng. eager evaluation) tarkoittaa lausekkeen arvon laskemista heti ensimmäisellä kerralla, kun lauseke esitellään. Tämä on yleisimpien ohjelmointikielten laskentastrategia. Laiskan laskennan toteuttavat useat puhtaasti funktionaaliset ohjelmointikielet, kuten Haskell. Myös jotkin moniparadigmaiset kielet toteuttavat laiskan laskennan, esimerkiksi Scala-kielen *lazy val* -lauseke.[5]

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 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.[5]

Funktionaalisissa ohjelmointikielissä rekursio 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 puhtaalle funktionaaliselle ohjelmoinnille tyypillinen piirre, jossa sama funktio voidaan määritellä useita kertoja. Funktiomäärittelyistä vain yhtä sovelletaan tapauskohtaisesti. Modernit funktionaaliset ohjelmointikielet tekevät hahmonsovitustoteutuksistaan mahdollisimman ilmaisuvoimaisia rohkaistakseen sen käyttöä. 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.

```
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), mikä tekee ohjelman rinnakkaistamisesta helppoa. Puhtaan funktion deterministisyys tarkoittaa, ettei muu samanaikainen suoritus voi vaikuttaa laskennnan oikeuteen, eikä sivuvaikutukseton funktio myöskään pysty vaikuttamaan muun samanaikaisen laskennan lopputulokseen.

Deterministisyys ja toistettavuus parantavat ohjelman testattavuutta. Jos deterministinen funktio toimii tietyillä parametreilla kerran, se toimii myös joka kerta ajon aikana. Toistettavuus helpottaa ajon aikana löydetyn virheellisen tilan toistamista syöttämällä sama tarkkaan määritetty virheellinen tila testiympäristöön.[6]

Tilattomuus ja abstraktit datatyypit käyttävät tyypillisesti enemmän prosessoritehoa ja muistia, kuin imperatiiviset ohjelmat. Pahimmassa tapauksessa tämä aiheuttaa $O(\log n)$ hakuajan funktionaaliselle datatyypille, verrattuna imperatiiviselle ohjelmoinnille tyypilliseen $O(n)$. Kuitenkin ohjelmat, jotka tekevät intensiivisiä numeerisia laskuja, ovat vain marginaalisesti hitaampia funktionaalisilla ohjelmointikielillä kuten OCaml ja Clean verrattuna C-kieleen.[7]

2.5 Ohjelmointikielet

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.[8]

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. [9]

Staattisen verkkosivun käännösvaihe

3.1.2 Dynaamiset verkkosivut

Dynaaminen verkkosivu tarkoittaa sivua, joka ei ole suoraan selaimen luettavassa muodossa. Dynaamisen sivun web- muodostaa selaimella renderöityvän sivun ajon

aikana. Dynaamisen verkkosivun operaatioiden toteutus ei olekaan rajattu selaimen tulkittaviin ohjelmointikieliin, vaan käytännössä minkä tahansa kielen käyttämninen on mahdollista. Yleisimpiä kieliä dynaamisen verkkosivun toteuttamiseen ovat PHP, Python, Perl, Ruby, Java, C# ja Node.js. Dynaaminen verkkosivu tallettaa useimmiten sisältönsä tietokantaan.[9]

3.2 Tutkimuskohteet

3.2.1 Dynaaminen sisällön hakeminen

3.2.2 Tapahtumankäsittely

4 Tutkittavat teknologiat

4.1 JavaScript ja React

4.2 Elm

5 Esimerkkisovellukset

5.1 JavaScript ja React: Juomamaatti

5.2 Elm: 2048

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

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

[5] D. A. Watt, ”Programming Language Design Concepts,” 2006.

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

[7] B. Fulgham, ”Which programs are fastest? | Computer Language Benchmarks Game,” 2011.

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

[9] H. Pettersen, ”From Static and Dynamic Websites to Static Site Generators,” 2016.