Jetro Kinnunen

ViPA – Tekoäly ja kontekstuaalinen oppiminen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

31.5.2017

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Jetro Kinnunen ViPA – Tekoäly ja kontekstuaalinen oppiminen 27 sivua + 1 liitettä 31.5.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Ohjaaja Auvo Häkkinen Tekstinohjaaja Jussi Alhorinne

Insinöörityö on tutkimus koneoppimisesta ja oppimisen sitomisesta konteksteihin. Työn pääasiallinen tavoite on selvittää algoritmien toimintaa ja löytää sopivin algoritmi projektin tarkoitukseen. Tutkimuksen toisena tavoitteena on suunnitella ohjelmistorakenne ja tietokantarakenne, joista sitten rakennetaan prototyyppi.

Projekti on tekijän itsenäinen tutkimusprojekti, joka aloitettiin vuoden 2017 alussa. Projektin tarkoitus on tutkia ja rakentaa pohjaa prototyypille virtuaaliselle assistentille. Toimivan assistentin toimintaideana on oppia käyttäjän käskyjen perusteella toimintatapoja ja optimoida näitä toimintatapoja. Työssä on tarkoitus tutkia kontekstuaalista oppimista ratkaisuna tiettyihin ongelmiin, joita muilla samantyylisillä assistenteilla on.

Työ jakautuu seuraaviin osa-alueisiin: projektin idean kuvaus, suunnittelu- ja toteutusvaihe, jatkokehitysideat sekä yhteenveto. Projektin idean kuvauksessa tullaan avaamaan lukijalle, mihin työn tutkimuksella on pyritty ja mitkä ovat olleet lähtökohdat. Suunnitteluvaiheessa kuvataan tarkemmin, millä tavoin on lähdetty ratkaisemaan projektin ideoita ohjelmistotekniikan malleja hyödyntäen. Toteutusvaiheessa kuvataan, miten prototyypin kannalta on toteutettu ohjelmisto-, tietokanta-, algoritmirakenne ja kuinka se on muuttunut suunnitteluvaiheesta. Jatkokehitys kohdassa kuvataan ominaisuuksia, joihin tulee keskittyä työn jälkeen. Lopuksi on yhteenveto, miten projekti meni.

Työssä on keskitytty algoritmien tutkimukseen ja ohjelmistorakenteen suunnitteluun jatkokehitystä varten.

Avainsanat	Kontekstuaalinen oppiminen, koneoppiminen, algoritmi

Author(s) Title Number of Pages Date	Jetro Kinnunen ViPA - Study on Artificial Intelligence and Contextual Learning 27 pages + 1 appendices 31 May 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Auvo Häkkinen, Principal Lecturer Jussi Alhorinne, Language Councelor

The goal of this Bachelor's thesis is the study of learning algorithms and tying the learning to contexts. Main points in this thesis are to study and find fitting algorithm for this project. Secondary objective is to design program and data structure, from which a prototype can be built.

The project is the writer's self-study, which begun at the start of 2017. The concept was a virtual personal assistant software and one of the goals was to create a prototype. The assistant would learn from the user's habits and optimise some structures by tying them to contexts. This prototype should demonstrate the algorithm and test concept for the context structure. The reason for studying contextual learning is to try and find solution to features current assistant software are lacking.

This thesis is divided to following parts: Description of the concept, design, development and future improvements. Description of the concept will cover what the project is about and what there is intended to be feature wise. The design part will cover the steps done to create and plan the structure required for this project. In the development part, will be covered how the prototype program, database and algorithm structure was done based on design. Future improvements cover the planned features and required changes to continue the project. Finally, there is a summary of this thesis.

The approach for this thesis is research of algorithms and designing required software structure for these algorithms.

Keywords	Contextual learning, machine learning, algorithm

Sisällys

Lyhenteet

AI	Artificial Intelligence. Tekoäly, algoritmeihin pohjautuva oppiva tietokone.
BS	Beam Search. Algoritmi, jolla voidaan tehdä puurakenteita.
COL	Context Oriented Learning. Kontekstipohjainen oppiminen, joka avataan luvussa 4.
DBO	Database object. Tietokantaobjekti on luokka, joka on tallennettavissa kantaan rivinä.
LDB	Local Database. Lokaalitietokanta on käyttäjän omassa ympäristössä oleva tietokanta.
MDF	Master Database File. SQL Serverin tiedostomuoto lokaaleille tietokannoille.
ODT	Oblique Decision Tree. Tietyntyyppinen oppimispuun muodostusstrategia.
ORM	Object-relational mapping. Ohjelmisto-olion yhdistäminen relaatiokannan tauluun.
RDB	Remote Database. Etätietokanta sijaitsee palvelimella ja siihen otetaan yhteys verkon ylitse.
ViPA	Virtual Personal Assistant. Ohjelmistoprojektin nimi.
UI	User Interface. Käyttöliittymä.
WPF	Windows Presentation Foundation. Microsoftin vuonna 2015 esittelemä työpöytäapplikaatioiden XAML-pohjainen käyttöliittymärajapinta.

1 Johdanto

Dokumentissa kuvailtu projekti on henkilökohtainen tutkimus, johon kuuluu tutkimuksen pohjalta tehty prototyyppi. Projektin tavoitteena on tehdä tutkimusta kontekstipohjaiseen oppimiseen sekä kehittää prototyyppi ViPA-sovelluksesta. ViPA on lyhenne sanoista Virtual Personal Assistant, jonka tarkoitus on hyödyntää koneoppimista optimoidakseen käyttäjän toimintoja. Projektissa tutkitaan kone-oppimista, algoritmeja sekä tietokantarakenteita, joita tarvitaan saamaan aikaiseksi sovellus, joka helpottaa käyttäjän informaationhallintaa. Tutkimus on pääasiassa teoreettinen, sen tarkoituksena ei ole tuottaa täysin toimivaa tuotetta.

Informaation määrä on nykyaikana niin valtava, että sen hallinnointiin menee paljon aikaa. Informaatio on sidottu niin moneen eri lähteeseen, että yksittäisen käyttäjän on hankalaa yrittää pitää kirjaa kaikista asiaan liittyvistä tiedonjyvistä. Projektin ideana on mahdollistaa käyttäjälle helppo tapa tehdä yksinkertaisia toimintoja muun toiminnan ohessa ja pitää kirjaa tiedonjyvistä, jotka liittyvät samaan yläkäsitteeseen. Markkinoilla on jo olemassa monia puhetta tunnistavia apulaissovelluksia, kuten Cortana Microsoftilla tai Siri Applella. Jatkuvasta kehityksestä huolimatta, suurin osa niistä ei kuitenkaan vielä osaa sitoa tietoa useista lähteistä yhteen käsitteeseen, eli kontekstiin. Tutkimuksen teon yhteydessä Samsung julkaisi Bixbyn, jonka olisi tarkoitus huomioida tämä ongelmaa. Yleisessä tutkimuksessa olevan puutteen perusteella on lähdetty tutkimaan, kuinka tätä ideaa voitaisiin toteuttaa.

Nykyiset assistentit ovat keskittyneet ease-of-access toimintoihin, eli miten käyttäjä voi mahdollisimman helposti hakea tietoa sen kautta ja yksinkertaisilla komennoilla esimerkiksi puheella. Cortanassa sekä Sirissä voi hakea puheella Bing-hakupalvelun kautta asioita sekä etsiä käyttäjän koneelta tiedostoja. Näistä nousi muutaman käyttäjän kanssa keskustellessa esille puute, että jos käyttäjä halusi töidensä ohessa etsiä projektiinsa liittyviä asioita, piti hänen erikseen hakea muistiinpanonsa, sitten sähköpostinsa ja lopuksi tiedostonsa. Jos nämä palvelut olisivat sitoneet kyseiset tiedot yhteen jollain tavalla, hän olisi voinut saada kaiken kerralla käsiinsä.

Tutkimuksen pääasiallinen tavoite on oppimisalgoritmien ymmärtäminen ja niiden soveltaminen projektissa. Tarkoitus on löytää sopiva algoritmi, joka tukee kontekstuaalisen

oppimisen mahdollisuutta. Lisäksi tarkoituksena on selvittää, mistä kontekstin tulisi koostua käytännössä, jotta sen rakenne pysyy mahdollisimman yksinkertaisena. Tutkimuksen toinen tavoite olisi suunnitella prototyyppi, jossa voidaan esitellä näitä tekniikoita tulevaisuudessa. Prototyyppiä tuskin saadaan täysin toimintakykyiseksi tutkimuksen aikana, joten tutkielman pääasiallisena tarkoituksena on suunnitella ViPA:n rakenteet jatkokehityskuntoon. Prioriteettijärjestyksessä tavoitteet ovat oppimisalgoritmin löytäminen ja ymmärtäminen, oppimisalgoritmin toteutus sekä prototyypin kehittäminen algoritmin pohjalta.

Projektia on tarkoitus laajentaa ja jatkaa myös lopputyön jälkeenkin. Laajennuksiin tulee kuulumaan ainakin seuraavia ominaisuuksia: puheentunnistus, työtuntikirjaus ja järkevä käyttöliittymä. Laajennuksien tarkoitus olisi tehdä ohjelman käytöstä helpompaa muun toiminnan ohessa. Projektia ei ole kuitenkaan tarkoitus missään vaiheessa kaupallistaa, vaan se on pelkästään tutkimustarkoituksiin ja yksityiskäyttöön. Projekti laitetaan tarpeeksi kehityttyään avoimeen jakoon, esimerkiksi GitHubiin.

Työ toteutetaan pääasiassa Microsoftin teknologioiden pohjalle. Tällöin hyödynnetään C#-kieltä, .NET-rajapintaa, SQL Serverin ominaisuuksia Visual Studion sisällä ja tulevaisuudessa Microsoftin puhetunnistusrajapintaa. Projektia suunnitellaan laajennettavan muillekin alustoille kuin Windows. Mobiililaitteet ja muut yleiset käyttöjärjestelmät, kuten Linux ja OSX, ovat tarpeellisia, jos ohjelmaa halutaan tulevaisuudessa levittää.

2 Tutkimuksen suunnitelma

Luvussa kuvaillaan tarkemmin, mitä ominaisuuksia projektin aikana pyritään saamaan osaksi ViPA:n toiminnallisuutta ja niiden merkitystä projektille. Ominaisuudet ovat seuraavat: haku, tiedostonhallinta, muistiinpanot, ilmoitukset ja kalenterimerkinnät. ViPA:n on tarkoitus yhdistää näitä toimintoja COL-logiikan (Context Oriented Learning) avulla. Toiminnot ovat suhteellisen yksinkertaisia, jotta voidaan keskittyä algoritmin toimintaan ja optimointiin.

2.1 Tutkimuksen konseptimäärittely

Tutkimuksen konseptina on kehittää digitaalinen assistentti, joka optimoi käyttäjän usein toistamia käytäntöjä sekä yhdistää useampaa palvelua tai tiedonlähdettä. Otetaan esimerkiksi jokin työprojekti, johon kuuluu useita sähköpostiviestejä, tekstidokumentteja, kooditiedostoja ja satunnaisia muistiinpanoja. ViPA:n toiminnallisuuden olisi tarkoitus lopullisessa vaiheessaan pystyä tekemään käyttäjän puolesta esimerkiksi käsky: ”Hae kaikki dokumentit liittyen kokoukseen 23.2.2016”. Operaatio palauttaisi käyttäjälle linkit kaikkiin hänen tiedostonsa: olivat ne sitten tietokoneella, pilvipalvelussa tai esimerkiksi sähköpostin liitteinä. Palveluita voidaan liittää, jos niillä on tukevat API:t ja käyttäjä liittää tunnuksensa ohjelmaan.

Sen vuoksi on lähdetty tutkimaan COL-logiikkaa eli kontekstipohjaista oppimislogiikkaa. Jokainen ViPA:lle annettu komento yhdistetään joko olemassa olevaan kontekstiin sen sisällön perusteella, tai sille luodaan uusi konteksti. Oppimisalgoritmin tehtävänä on yhdistää eri konteksteja käyttäjän tottumuksien perusteella. Tätä logiikkaa kuvataan tarkemmin luvussa 4.1.

Kontekstit ja oppimistiedot tallennetaan tietokantaan, joka toimii ORM-periaatteella (Object-relational mapping). Oppimistiedot liitetään konteksteihin ja kontekstit yhdistetään joksikin rakenteeksi, esimerkiksi puurakenne. Rakenteen luomisen jälkeen se tallennetaan tietokantaan, jotta ViPA:n ei tarvitse luoda rakenteita aina uusiksi. Rakenne nopeuttaa kontekstien käsittelyaikaa, minkä lisäksi uusien kontekstien lisääminen on helpompaa, kun valmis rakenne on jo olemassa. Rakennetta voidaan sitten optimoida tietyin väliajoin automaattisesti ohjelman toimesta. ViPA voi ajaa tietyin väliajoin puurakenteen uusiksi, jotta rakenne tulee optimoitua.

2.2 Ominaisuudet

Kuten aiemmin kuvattiin, hakutoiminto mahdollistaa tiedon hakemisen koneelta hyödyntäen kontekstipohjaa. Käyttäjän antamasta käskystä tunnistetaan mikä, tai mitkä ovat etsittävät datatyypit, ja tämän tunnistuksen jälkeen verrataan hakusanoja olemassa oleviin konteksteihin. Datatyyppejä voi olla esimerkiksi ”sähköposti” tai ”muistiinpanot”, koska nykyaikana käsittelemämme tieto sijaitsee monessa eri paikassa käyttäjästä riippuen. Jotta käyttäjä voi yhdistää nämä tiedot, täytyy kyseiset ohjelmat yhdistää kontekstiin. Kontekstista käyttäjän on mahdollista löytää yhdellä hakusanalla tietoa useasta lähteestä.

ViPAlle voidaan antaa yksinkertaisia komentoja, kuten ”poista”, ”siirrä”, ”leikkaa” jne. Prototyypissä nämä olisivat vain manuaalisesti käytettävissä, eli komentoon täytyy laittaa aluksi komentolinjakomentojen tyylisiä käskyjä. Jos halutaan esimerkiksi järjestää tiettyjä projektidokumentteja, ViPAlle voi antaa käskyn ”järjestä kaikki Projekti XyZ tiedostot polkuun C:/Projektit/Vanhat Projektit”. ViPA voi muodostaa komennoista esimerkiksi seuraavanlaisia kontekstiliitoksia ja päätelmiä: vanhat projektit kuuluvat tiettyyn polkuun, kansiorakenne käyttäjän ympäristössä tai projekti XyZ on vanha projekti. Näitä liitoksia voidaan sitten hyödyntää tulevaisuudessa samantyylisiä käskyjä ajettaessa.

Muistiinpanot ja kalenterimerkinnät ovat yksinkertainen ominaisuus, joka tulee oikeastaan omilleen vasta puheentunnistuksen myötä. Tutkimusversioon luodaan näille perusta, josta toimiva versio on helppo viimeistellä. Tarkoitus olisi luoda lyhyitä muistiinpanoja, jotka ViPA tallentaa johonkin kontekstiin tekstistä poimittujen avainsanojen perusteella. Sama logiikkaa pätee kalenterimerkintöihin. Esimerkkinä tilanne, jossa käyttäjä on laittanut muistiinpanon ”muista ostaa mökille ruoat”. Se voidaan yhdistää kontekstiin mökki, ruoka ja niiden yhteinen alikonteksti, mökkiruoka. Näistä ViPA voisi tarjota käyttäjälle, vaikka aikaisemmin tallennettuja reseptejä tai ehdottaa päivää, milloin kalenteriin sopisi lähteä perjantaina. On myös harkittavissa, että nämä olisivat yhtenäinen ominaisuus, jossa käyttäjä voi saada muistutuksia hänen muistiinpanoihinsa merkitsemistään päivämääristä. Pyytämällä ViPAa avaamaan kalenterin, voisi ViPA näyttää käyttäjälle hänen tulevia merkintöjään, jotka voivat muodostua useammasta kontekstista.

Ilmoitukset ovat itsenäinen ominaisuutensa, mutta niitä voidaan myös luoda muistiinpanojen toimesta. Ilmoitusten ei tarvitse olla pelkästään päivämäärään tai aikaan sidonnaisia. Ne voivat olla esimerkiksi huomautuksia loppuvasta tilasta kovalevyllä tai muista

käyttäjän määrittämistä huomionarvoisista asioista. ViPA voisi hyödyntää oppimisprosessia ja ilmoittaa aktiivisimpiin konteksteihin liittyvistä asioista. Ilmoituksia voitaisiin muodostaa myös sen perusteella, mitä käyttäjä usein ViPA:lta kyselee. Nämä eivät ole kontekstisidonnaisia itsessään, vaan ilmoitusten tehtävä on toimia UI:n ominaisuutena. Jatkossa tulee tutkia, mitkä olisivat hyviä tapoja esittää käyttäjälle näitä ilmoituksia ilman, että ne käyvät häiritseviksi. Näiden prioriteetti työssä on alhainen, sillä ne eivät vaikuta oppimisalgoritmin toimintaan, vaan ovat puhdas ominaisuus. Ilmoitukset eivät ole samalla tavalla sidoksissa kontekstirakenteeseen, koska ne eivät itsessään sisällä informaatiota.

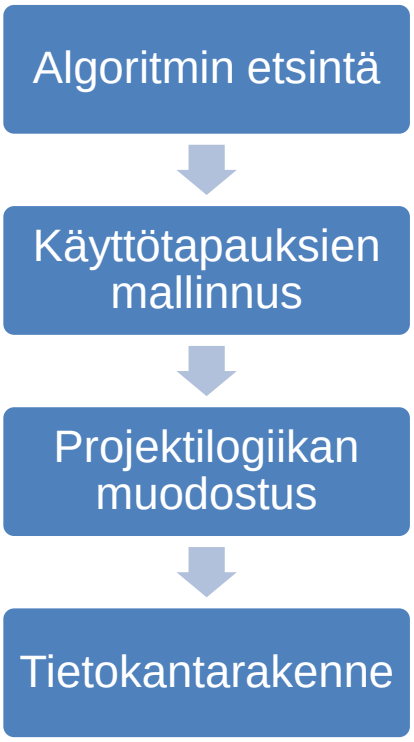
2.3 Etenemissuunnitelma

Tutkimuksen suunnitelmana on edetä ensin löytämällä sopiva algoritmi tarkoitukseen, sekä opiskelemalla algoritmilogiikkaa. Tätä varten on tarkoitus tutkia aikaisempia insinööritutkintoja ja alan kirjallisuutta. Tutkimuksen yhteydessä käytiin läpi useita koneoppimiseen ja tietokantaoptimisaation keskittyvää teosta, joista pystyttiin rakentamaan pohjaa. Teoksissa ei kuvata suoraa ratkaisua tällaiselle ongelmalle, joten pitää valita sopivin. Tiedon pohjalta lähdetään rakentamaan itse ratkaisua. Työssä ei käytetä valmiita oppimisalgoritmikirjastoja, koska tarkoituksena on oppia, miten logiikka toimii käytännössä. Lisäksi ratkaisun pitää olla tekijän vapaasti muokattavissa ja kehitettävissä tulevaisuudessa.

Projektin ensimmäisessä suunnitteluvaiheessa tulee miettiä, miten kontekstirakenteessa toimii oppiminen ja kuinka esimerkiksi kontekstiliitoksia tulee muodostaa. Sopivin projektiin olisi algoritmi, joka on hyvin dokumentoitu, jotta algoritmilogiikan muodostaminen ei käy liian vaikeaksi. Algoritmi, josta on useita käyttöesimerkkejä ja laajaa tutkimusta, olisi sopivin projektin tarkoitukseen. On kuitenkin mietittävä, millä periaatteella kontekstit tulee yhdistää ja mitä kaikkea niiden tulee sisältää. Kun projektin toimintalogiikka on valittu, pitää sen pohjalta muodostaa käyttötapauksia ja niistä voidaan rakentaa tarvittava ohjelmistologiikka. Kun käyttötapaukset ovat suhteellisen selviä, voidaan siirtyä seuraavaan vaiheeseen.

Ohjelmistorakennetta lähdetään muodostamaan MVC-rakenteen (Model View Controller) pohjalle, joka helpottaa vaihtuvuutta projektin osissa. Kehityksen kuluessa tietokannan rakenne, algoritmin käsittely ja näkymät tulevat muuttumaan. Suunnittelussa tulee

ottaa huomioon myös algoritmin asettamat tarpeet ohjelmistorakenteelle. Kun ohjelmistorakenne on muodostettu, voidaan miettiä tarkemmin sitä mitkä ovat tietorakenteet, joita järjestelmä käyttää ja rakentaa tietokantaratkaisu sen pohjalta. Kantaa suunnitellessa tulee myös harkita mitä tarvitsee tallentaa ja mitä ei, eli mitkä luokat ovat DBO (Database object) ja mitkä vain ajonaikaisia.



Kuva 1. Etenemisprosessi

3 Tekoäly ja algoritmit

Tekoälyn tehtävä projektissa on kontekstien looginen organisointi. Jotta konteksteja voidaan yhdistää, tarvitaan aluksi käyttäjältä jonkinlaista ohjeistusta logiikkaan, millä ne tulee järjestää. Kun käyttäjä on antanut tietyt säännöt ohjelmalle, millä periaatteilla yhdistää konteksteja, voi se toimia itsenäisemmin. ViPA:lle voidaan luoda jonkinlaisia perustavanlaatuisia konteksteja, joista sen on mahdollista laajentaa konteksteja omatoimisesti. Tästä esimerkkinä voisi olla ”projekti”-konteksti, johon voidaan sitoa kaikki projekteihin liittyvät alakontekstit ilman, että käyttäjän tarvitsee määritellä perustavanlaatuisia konsepteja.

3.1 Iterative Dichotomiser 3-algoritmi

ID3, eli Iterative Dichotomiser 3 on algoritmi, joka järjestää annetun informaatioon puurakenteeksi. ID3:a päädyttiin käyttämään tutkimuksessa, koska se osoittautui olevan suhteellisen tunnettu algoritmi, josta on paljon tutkimusta. Oppimispuu soveltuu hyvin tilanteisiin, joissa on käytössä selvät ominaisuus/arvo-parit tai boolean päätökset, eli tosi-epätosi vertailu. Kontekstirakenteessa on ominaisuus/arvo-pareja, kuten esimerkiksi: Nimi=”ViPA”, ”Project X”, Polku=”C:\”, ”D:\”, joista nimi valittiin alustavaksi järjestelyperiaatteeksi. Järjestys nimen pohjalta perustuu siihen, että näin yhdistetään useita käyttötapoja yhden kontekstin alle. Kontekstiin ”työ” voisi esimerkiksi liittyä useita projektinimikkeitä, internetsivuja ja komentoja, joista alikonteksteja voisi olla esimerkiksi yksittäinen projekti.

Ohjelmistokoodissa 1 esitetään pseudokoodilla, mikä on algoritmin toimintaperiaate käsitteellisellä tasolla. Pseudokoodissa esitetään, kuinka algoritmille annetaan example_set -tietokokoelma ja Properties-lista, jota sen olisi tarkoitus prosessoida. Funktio vertailee kokoelman sisällöstä kappaleet, ja jos ne ovat samaa luokkaa, palauttaa lehtisolmun. Jos näin ei ole, ja Properties-lista on tyhjä, ViPA palauttaa lehtisolmun, joka on disjunktio tietokokoelman kaikista luokista. Siinä tilanteessa, jos luokat eivät ole samoja ja Properties listalla on ominaisuuksia, valitaan yksi ominaisuus P juureksi ja poistetaan se listalta. Sitten käydään läpi jokainen arvo V ominaisuus P:n sisällä ja rakennetaan puuta kutsumalla samaa metodia rekursiivisesti. V:stä tulee solmu puuhun ja se antaa rekursiiviselle metodille ne arvot tietokokoelmasta, joihin sisältyy V. Tuloksena syntynyt alipuu sitten sidotaan solmuun V. (1, s.376)

```
function induce_tree(example_set, Properties)
begin
if all entries in example_set are in the same class
then return a leaf node labeled with that class
    else if Properties is empty
            then return leaf node labeled with disjunction of all classes in
            example_set
        else begin
            select a property P, and make it the root of the current tree
            delete P from Properties;
                for each value, V, of P,
                    begin
                        create a branch of the tree labeled with V;
                        let partition, be elements of example_set with values
                        V for property P;
                        call induce_tree(partition, Properties), attach result
                        to branch V
                    end
        end
end
```

Ohjelmistokoodi 1. ID3-algoritmin pseudokoodi (1, s.376)

ID3-algoritmin toiminta on rakennettu Claude E. Shannonin informaatioteorian pohjalle, jonka Luger (1, s.378) esittää seuraavalla tavalla: informaation määrä viestissä on $\boldsymbol{p}$:n, eli jokaisen viestin esiintymismahdollisuus, funktio. Se voidaan tiivistää muotoon $-\log_2 \boldsymbol{p}$. Jos $\boldsymbol{M} = (\boldsymbol{m_1}, \boldsymbol{m_2} \ldots \boldsymbol{m_n})$ on kokoelma viestejä ja todennäköisyys $\boldsymbol{p}(\boldsymbol{m_i})$ viestin esiintymiselle, silloin $\boldsymbol{M}$:n oletettu informaatioarvo, eli entropia voidaan laskea kaavalla:

$$\boldsymbol{I}[\mathbf{M}] = -\sum_{i=1}^{n} \boldsymbol{p}(\boldsymbol{m_i}) \log_2\big(\boldsymbol{p}(\boldsymbol{m_i})\big) = \boldsymbol{E}[-\log_2 \boldsymbol{p}(\boldsymbol{m_i})]$$

Kaava 1. Informaation määrä viestikokoelmassa (1, s.378)

Algoritmissa pyritään saamaan suurin mahdollinen informaatiokasvu järjestämällä harjoitteludataa. Informaatiokasvulla tarkoitetaan erotusta yksittäisen viestin esiintymisen ja koko tietomäärän välillä. Koko tietomäärä, joka tarvitaan tähän luokitteluun, on painotettu keskiarvo jokaisen alipuun informaatiosta. Painotus perustuu aikaisemmin mainittuun entropia-arvoon, eli tiedon esiintymisen lukumäärään prosentuaalisesti.

Mitä suurempi informaatiokasvu on, sitä harvinaisempi tiedon esiintyminen on. Käytännössä tämä tarkoittaa, kuinka usein arvo esiintyy annetussa kokoelmassa. Tieto on arvokkaampaa mitä vähemmän sitä on. Oletetaan kokoelma esimerkkitapauksia **C**. Jos asetetaan ominaisuus **P**, jossa on **n** arvoa, puun juureksi, jakaa se **C**:n pienempiin kokoelmiin {C_1, C_2, … C_n}. Tällöin oletettu informaatio, joka tarvitaan puun valmistamiseen, kun **P** on valittu juureksi, voidaan laskea kaavalla:

$$\mathbf{E}[\mathbf{P}] = \sum_{i=1}^{n} \frac{|\boldsymbol{C_i}|}{|\boldsymbol{C}|} I[\boldsymbol{C_i}]$$

Kaava 2. Tarvittava informaatio puun valmistumiseen. (1, s.379)

P:n kasvu lasketaan vähentämällä oletettu informaatiotarve **E**[**P**] informaatiokokonaisuudesta **I**[**C**], eli kaavana: $gain(\boldsymbol{P}) = \mathbf{I}[\mathbf{C}] - \mathbf{E}[\mathbf{P}]$

Arvon perusteella järjestetään kontekstit puurakenteeseen. Logiikka perustuu tietorakenteisiin, joissa on useampi tietotyyppi vertailussa yhtä aikaa. Eli jos on useampaa tietoa, joita vertaillaan yhtä aikaa, ne eivät ole toisiinsa suoraan sidonnaisia. Ne voivat olla paikkatietoja, päivämäärätietoja ja nimiä. Kyseinen vertailu ei ole tärkeää prototyypin kannalta, mutta laajentamisen kannalta kyllä. Kun ViPA:n pitää ruveta käsittelemään yhtaikaisesti monia tietotyyppejä, nousee kyseinen kaava oleelliseksi. Prototyyppiversiossa käsitellään vain tekstipohjaista tietoa.

3.2 C4.5-algoritmi

C4.5-algoritmi on jatkokehitetty versio ID3-algoritmista, joka lisää seuraavanlaisia ominaisuuksia: hyväksyy myös tyhjää datasisältöä, ratkaisee tiettyjä skaalautumisongelmia käyttämällä ”karsimista” ja eri tietotyypeille voidaan antaa painoarvoja harjoitusdatan sisällä. ID3-algoritmilla on ongelmia isojen tietomäärien kanssa, joten C4.5 oli kehitetty ratkaisemaan näitä. Seuraavat tiedot on pohjattu “A comparative study of decision tree ID3 and C4.5”-tutkimuksen (7) esittämiin vertailutuloksiin.

Tyhjää tai tuntematonta tietoa voidaan käsitellä käsittelemällä sitä niiden ominaisuuksien ympärillä, jossa kyseistä tietoa esiintyy. Nämä arvot voidaan päätöspuun avulla luokitella arvioimalla todennäköisyydet eri tuloksista. Tästä muodostuu kaava:

$$Info(p,T) = \sum_{j=1}^{n} (p_j * Entropy(p_j))$$

Kaava 2. Entropian laskeminen tuntemattoman arvon kanssa

Jossa Info(T) = Entropy(T) ja F on tunnettujen arvojen määrä tietokannassa jaettuna arvojen määrällä joukossa, joka liittyy tiettyyn ominaisuuteen. (7, s.15. A. Attributes of unknown value.) Joka esitetään kaavassa:

$$Gain\ (p) = F\ \left(Info\ (T) - Info\ (p,T)\right)$$

Kaava 3. Informaatiokasvu tuntemattomasta arvosta.

Toinen tärkeä asia, joka lisättiin C4.5-algoritmissa, on karsiminen. Karsimisella tarkoitetaan päätöspuun yhteydessä koon pienentämistä poistamalla osia, jotka ovat turhia puun määrittelyn kannalta. Samalla pystytään myös pienentämään algoritmin virhemarginaalia huomattavasti. Karsimisella puututaan ongelmaan, jossa harjoitteludatasta luodaan liian laaja puu datalle ja se on liian herkkä näytedatan melulle. Eli jos on paljon kontekssteja, jotka eivät liity toisiinsa, yhdistetään ne puuhun kuitenkin. Puun kannalta se hidastaa prosessointia erityisesti ison tietomäärän kanssa, koska puun käsittelyssä on myös irrelevantteja kontekssteja. (7, s.16. C. Pruning.)

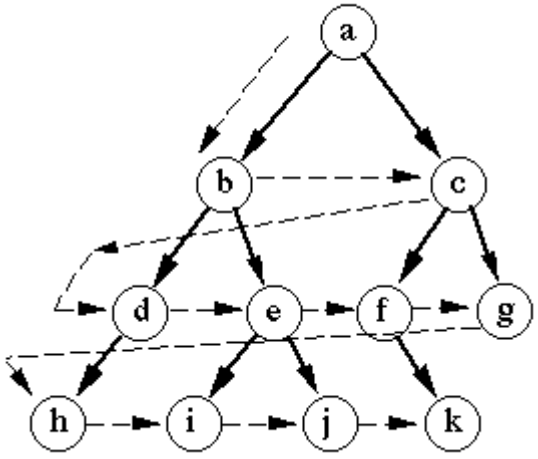
Karsimisalgoritmi perustuu pessimistiseen arvioon virhetaajuudesta joukossa, jossa on N tapausta. Joista arvot E eivät kuulu yleisimpään luokkaan. Sen sijaan, että laskettaisiin E:n suhdetta N:ään, C4.5 pyrkii määrittämään ylärajan todennäköisyydelle kun E tapahtumia seurataan N:ssä kokeita. Arvioimiseen käytetään käyttäjän määrittämää luottamusarvoa, josta perusarvona käytetään 0,25. (7, s.16. C. Pruning.)

Karsiminen aloitetaan lehtisolmuista ja edetään kohti juurta. Oletettu virhearvo lehtisolmussa, jossa on N arvoa ja E virhettä on N kertaa pessimistinen virhetaajuus, joka mainittiin aikaisemmin. Alipuuta varten C4.5 lisää haarojen arvoidut virheet ja vertaa sitä arvoituihin virheisiin, jos alipuu korvattaisiin lehtisolmulla; jos arvot ovat samat, alipuu on karsittu. (7, s.16. C. Pruning.)

3.3 Muita harkittuja algoritmirakenteita

Tutkimusprosessin aikana käytiin läpi useita eri algoritmirakenteita ja logiikoita ennen ID3 ja C4.5 rakenteeseen päätymistä. Algoritmeja, joita käytiin läpi projektin aikana, olivat tärkeimpinä ja projektiin harkittuina: Beam Search-logiikka, Gale-Shapley ja Oblique Decision Tree-logiikka.

Beam Search-logiikka perustuu logiikkaan, jossa lasketaan säteen pituus, joka kuvastaa jokaisen tason parasta arvoa (4. s.363). BS-logiikassa käytetään breadth-first hakua luomaan hakupuu. Kuvassa 2 esitetään, miten hakupuun logiikka toimii.



Kuva 2. Breadth-first-search. (8)

BS-logiikassa prosessointi lähdetään prosessoimaan juuresta vasemmassa laidassa olevaan solmuun, josta käydään läpi samalla tasolla olevat solmut. Sen jälkeen siirrytään taas vasempaan laitaan tasoa alempana ja toistetaan prosessi, kunnes kaikki solmut on käyty. BS-logiikka oli lupaava vaihtoehto, mutta jäi jälkeen käyttömahdollisuuksissa, sillä se ei ole oppimisalgoritmi, vaan algoritmi tiedon hakemiseen puurakenteesta. Sitä voidaan mahdollisesti soveltaa myöhemmin, jos tietokannan koko osoittautuu isoksi suorittamaan haku-operaatioita. Tämä algoritmi vaikuttaa hyvältä hakuperiaatteelta, jos tarvitsee käydä koko puu läpi.

Gale-Shapley-algoritmi on teoria stabiilista avioliitosta tiedon välillä, ikään kuin sidoksista. Algoritmi syntyi David Galen ja Lloyed Shapleyn toimesta, kun he todistivat, että jos on sama määrä miehiä (M) kuin naisia (W), voidaan kaikille luoda stabiili avioliitto.

Näin todistaen, että stabiili avioliitto-ongelma on ratkaistavissa. Avioliitto ei ole stabiili henkilölle A, jos a) jokin osa Asta pitää jostain osasta Btä ylitse oman parinsa b) B myös pitää Asta enemmän kuin omasta paristaan. Algoritmissa oli mielenkiintoinen idea sitouttaa tietotyyppejä, mutta ei osoittautunut tähän projektiin sopivaksi. Algoritmi on hyvin rajoittunut yhdenlaiseen ongelmaan, jota avataan ohjelmistokoodissa 2.

```
function stableMatching {
    Initialize all m ∈ M and w ∈ W to free
    while ∃ free man m who still has a woman w to propose to {
       w = first woman on m’s list to whom m has not yet proposed
       if w is free
         (m, w) become engaged
       else some pair (m', w) already exists
         if w prefers m to m'
            m' becomes free
           (m, w) become engaged
         else
           (m', w) remain engaged
    }
}
```

Ohjelmistokoodi 2. Gale-Shapley-algoritmin pseudokoodi. (9 s.54)

Algoritmi keskittyy yhdistämään jokaisen tietoyksikön vähintään yhden tietoyksikön kanssa, varmistaen kaikkien olevan ”avioliitossa”, ja pyrkimään ”parhaaseen” liittoon. Ratkaisu ei soveltunut projektiin, sillä tietoja olisi vaikea järjestää tällä periaatteella, kun käsitellään konteksteja. Algoritmi pyrkii luomaan pareja, ei puurakenteita. Se liittäisi kaksi kontekstia toisiinsa ja varmistaisi, että jokaisella kontekstilla on pari, mutta tämä ei ole haluttavaa tutkimuksen kannalta.

Oblique Decision Tree on oppimispuun käsittelyä erilaisella tavalla kuin Quinlanin logiikka ID3ssa. Puuta ei rakenneta yhtä suoraviivaisesti, vaan siinä otetaan huomioon asioita kuten solmujen etäisyys hypertasolla, jotta saadaan rakennettua parhaiten optimoitu puu. C4.5-algoritmi pohjaa tähän logiikan pohjalle puun rakentamisessa ja myös karsiminen on pohjiltaan lähtöisin ODT:n toiminnasta. (10)

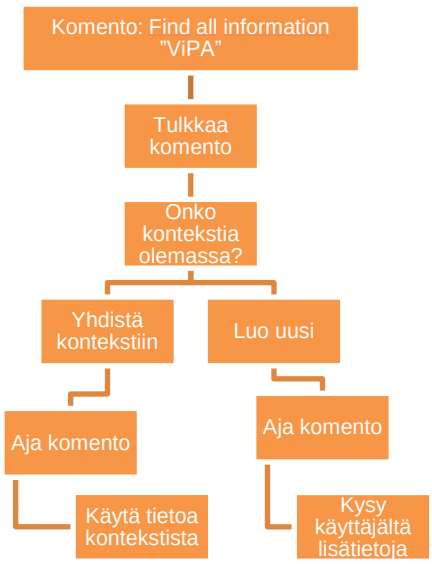
4 Tutkimuksen toteutus

Tutkimuksen toteutuksessa on lähdetty viemään aikaisemmin puhuttuja teorioita käytännön muotoon. Prosessi aloitetaan miettimällä käyttötapaukset, jotka toimivat esimerkkinä, miten käyttäjä voisi ohjelmaa käyttää. Sen jälkeen niiden pohjalta muodostetaan ohjelmistorakenne ja tietokantarakenne. Projektin jatkokehityksen kannalta on tärkeää olla hyvin suunniteltu runko, jonka avulla vältytään ylimääräiseltä työltä myöhemmissä vaiheissa.

4.1 Käyttötapaukset

Käyttötapauksilla on tarkoitus selvittää käyttäjän ja ohjelmiston interaktiota sekä mallintaa toimintalogiikkaa. Erilaisia käyttötapauksia on kolme kappaletta, joiden pohjalta voidaan lähteä kehittämään ohjelmistologiikkaa. Käyttötapaukset on valittu sen perusteella, miten ViPAn eri ominaisuuksia sekä niiden vaikutusta algoritmiin tullaan testaamaan. Ensimmäisessä tapauksessa on tarkoitus mallintaa hakuominaisuutta, toisessa tiedoston siirtoa ja kolmannessa ilmoituksen luomista.

Hakuominaisuuden suunnittelemisesta syntyi kaksijakoinen kaavio, jossa nousee esille kaksi tilannetta: uuteen kontekstiin ja olemassa olevaan kontekstiin perustuva haku. Kuvassa 3 kolme ensimmäistä prosessia toistuvat jokaisessa käyttötapauksessa.



Kuva 3. Käyttötapaus: Projektitietojen hakeminen

Käyttäjä haluaa hakea kaikki tiedot liittyen projektiin ”ViPA”. Hän kirjoittaa ViPA-komentoikkunaan komennon:

find all information ViPA

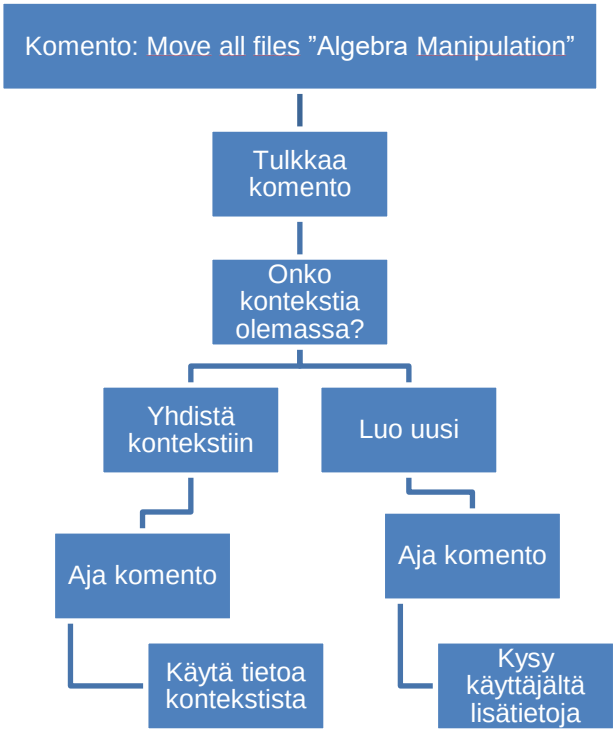
Ohjelma tulkkaa käskyn jäsentäen sen: komento = find, rajoitin = all, tyyppi = information ja hakutermit = ViPA.

Seuraavaksi otetaan yhteys tietokantaan ja katsotaan, onko hakutermillä ViPA olemassa kontekstia. Jos ei ole, tulee semmoinen luoda. ViPA luo uuden kontekstiolion nimellä ViPA ja tallentaa sen kantaan mukanaan ensimmäisellä kerralla käytetty käsky. Jos konteksti on olemassa, tulee hakea lähin tai lähimmät hakutulosta vastaavat. Tähän kontekstiin on linkitetty kaikki tiedostot ja muistiinpanot, jotka on yhdistetty kontekstiin joko tittelinsä tai sisältönsä perusteella.

Seuraavaksi ajetaan ”find”-komento, joka palauttaa osoitteen jokaiseen kontekstiin liittyvään asiaan. Tulokset esitetään käyttäjälle käyttöliittymässä sellaisessa muodossa, että käyttäjä voi niistä päästä suoraan käsiksi tietoihin.

Jos komennon ajamisessa tapahtuu virheitä, ViPA antaa käyttäjälle ilmoituksen. Jatkokehityksessä on otettava huomioon, miten toimia tilanteissa, joissa käyttäjä esimerkiksi kirjoittaa komennon väärin. On myös mahdollista, että käyttäjä on kirjoittanut väärin kontekstin määrittelevän osuuden ja ViPA luo toisen kontekstin, joka on väärinkirjoitettu. Yksinkertaisena ratkaisuna voisi olla käyttäjältä varmistus kontekstin luonnin ja komennon ajon yhteydessä. Myöhemmin ratkaisuna voisi miettiä jonkinlaista vertailua, jossa katsotaan, onko sana todella lähellä jotain toista sanaa.

Seuraavaksi, käydään läpi tilannetta, jossa tehdään yksinkertainen tiedostonkäsittelyoperaatio. Kuvassa 4 voidaan nähdä, että rakenne tosiaan pysyy samanlaisena. Erot ilmenevät vasta komennon ajamisen yhteydessä. Käyttötapaus 1 ja 2 eroavat käytännössä siinä, mitä tietoja tarvitsevat lopullisen komennon suorittamiseen.



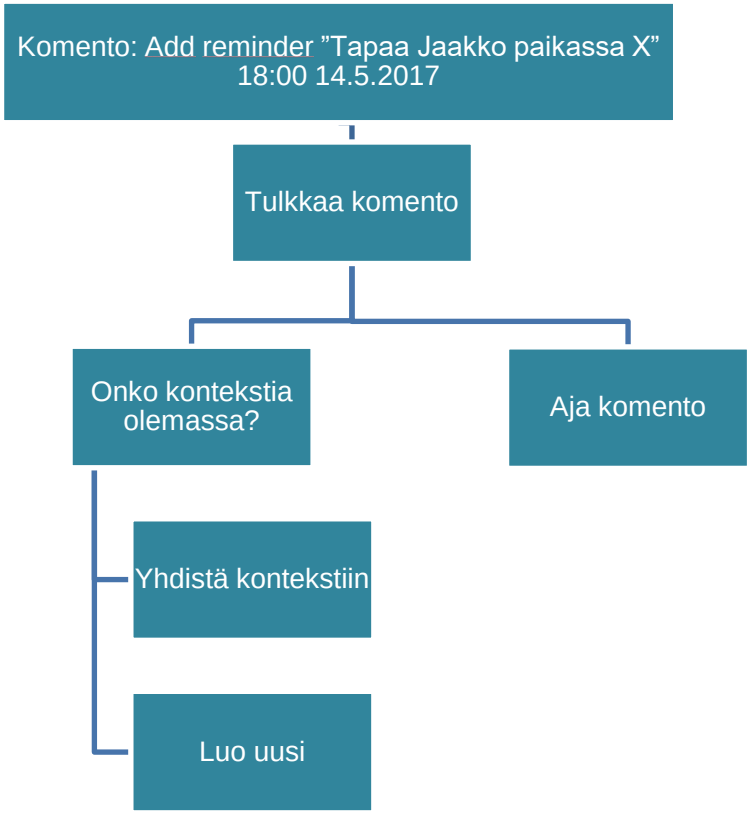
Kuva 4. Käyttötapaus: Tiedostojen siirto

Käyttäjä haluaa siirtää tiedostonsa uuteen paikkaan, jotka liittyvät kontekstiin ”Algebra Manipulation”. Hän antaa komennon muodossa:

move all files Algebra Manipulation

Käsky tulkataan ja jaetaan osioihin. Sitten selvitetään, onko kontekstia olemassa ja luodaan uusi tarvittaessa. Jos kontekstia ei ole, ohjelma kysyy käyttäjältä, mistä tiedostot voi löytää. Lopuksi kysytään, mihin siirron tulee tapahtua. Tiedot siirretään käyttämällä sisäistä käskyä ja palautetaan käyttäjälle, jos operaatio onnistuu. Jos operaatio epäonnistuu, ohjelma yrittää pari kertaa uudestaan. Käyttäjälle näytetään ilmoitus ja syy, miksi operaatio epäonnistui. Kirjoitusvirheiden lisäksi, on mahdollista, että komento epäonnistuu esimerkiksi käyttöoikeuksien puutteesta. On myös mahdollista, että kansiopolku on muuttunut tai sitä ei ole enää olemassa.

Muistutus eroaa käyttötapauksena kahdesta aikaisemmasta, koska komento voidaan ajaa yhtä aikaa, kun kontekstia käsitellään. Kuvassa 5 esitetään rakenne rinnakkaisajolle. Muistutus kannattaa olla erillään, jota seurataan, kunnes hälytysaika on mennyt. Kontekstiin lisätään vain tekstimuodossa muistutuksen tiedot.



Kuva 5. Käyttötapaus: Muistutus ja Muistiinpano

Käyttäjä haluaa lisätä muistutuksen tapaamisestaan. Hän syöttää komennon:

add reminder ”Tapaa Jaakko paikassa X” 18:00 14.5.2017

Paloittelussa otetaan viestiksi kommenteissa olevaa tekstiä. Sen jälkeen tulee kellonaika ja sitten päivämäärä. Ohjelma lisää muistiinpanon tietokantaan ja vertaa sanoja olemassa oleviin konteksteihin. Jos on olemassa konteksti, johon muistiinpano liittyy, tehdään linkitys niiden välillä. Jos muistiinpanoissa on päivämäärä ja aika, luodaan samalla muistutus. Luonnin yhteydessä ViPA kysyy, milloin halutaan ilmoitus.

4.2 Algoritmirakenne

Prototyyppiä varten luotiin seuraavat metodit:

- Double Entropy(List<ViPAContext> data)
- SplitEntropy(List<ViPAContext> data, ViPAContext context)

- ContextTree BuildContextTree(List<ViPAContext> data, bool prune)

- Double InformationGain(List<ViPAContext> data, ViPAContext comparison)

Entropy-metodi laskee jokaisen kontekstien entropia-arvon periaatteella: mitä useammin kontekstin nimi esiintyy muissa konteksteissa, sitä suurempi sen entropia-arvo on. Nämä arvot ja listan koko palautetaan sitten jatkokäsittelyä varten. Metodi palauttaa double-taulun ja kontekstien määrän, jotka annetaan eteenpäin. Seuraava askel on jokaisen arvon kohdalla laskea esiintymismäärä kontekstien määrällä. Jos arvo on eri kuin nolla, voidaan suorittaa $esiintymismäärä * \log_2(esiintymismäärä)$. Jokaiselle entropia-arvolle lasketaan summa, joka sitten summataan yhteen muuttujaan, muutetaan käänteisluvuksi ja palautetaan.

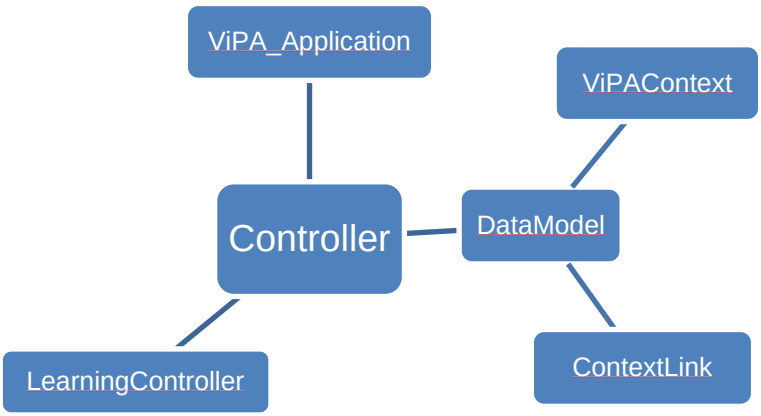
SplitEntropy-metodi laskee entropia-arvon yksittäiselle kontekstille verrattuna muuhun dataan. Metodissa siis verrataan yksittäisen kontekstin entropia erotuksena Entropy-metodin laskemasta kokonaisentropiasta.

BuildContextTree-metodi rakentaa puuta sen perusteella, mikä on informaatioarvon kasvu. Mitä lähempänä arvo on nollaa, sitä lähempänä puun juurta konteksti on. InformationGain-metodia käytetään informaatioarvon laskemiseen. BuildContextTree-metodia tehdessä tuli vastaan tarve luoda erillinen objekti, johon säilötään ViPAContext-olioiden puurakenne. On järkevämpää olla olemassa luokka, johon voidaan säilöä koko puurakenne kuin alkuperäisessä ideassa, jossa puurakenne säilöttiin kontekstiin alikonteksteina. Muutoin tulee yksittäiseen kontekstiolioon turhan paljon painetta.

InformationGain laskee, kuten luvussa 3.1 on esitetty, entropian erotuksen koko muun datan entropiasummasta. Toisin sanottuna metodi laskee Entropy-arvon ja SplitEntropy-arvon erotuksen. Tämä on se vertailuarvo, jolla puurakennetta muodostetaan. Eli kuten aikaisemmin kuvattiin, juureksi valitaan suurimman informaatioarvon antava konteksti, joka projektin tapauksessa on tietojoukossa useimmiten esiintyvä termi. Termillä tarkoitetaan kontekstin nimeä, joka esiintyy osana alikontekstin nimeä.

4.3 Ohjelmistorakenne

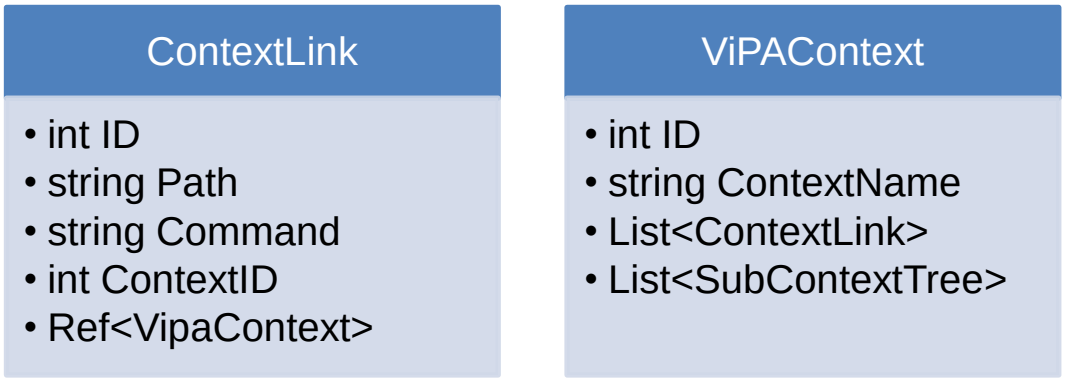
Ohjelmistorakenteen mallinnus aloitetaan ensin MVC-mallista. Malli koostuu kolmesta luokasta, jotka ovat tietokantamalli DataModel, näkymä ViPA_Application ja ohjain. Mallin käyttö on projektissa tärkeää, koska UI pitää päivittää, kun projektia jatketaan. Myös tietokantamalleja joudutaan kehittymään tulevaisuudessa, joten se antaa hyvän pohjan jatkokehitykselle. Mallin yhteyteen tarvitaan DBO-luokkia ainakin kaksi kappaletta: ViPAContext ja ContextLink.



Kuva 6. MVC-malli

Kuvassa 6 esitetään miten nämä luokat yhdistyvät: MVC-mallin mukaan, Controller-luokka toimii välittäjänä näkymän ja tietokantamallin välillä. Myös LearningController on sidottu Controlleriin. DBO-luokat ovat käytännössä yhteydessä kaikkiin luokkiin, mutta kuvassa on tarkoitus ilmaista, että vain tietokantamallin kautta voidaan muokata ja tallentaa näiden luokkien olioita.

Kuvassa 7 esitetään kahden aikaisemmin mainitun DBO-luokan rakenne. ContextLink-luokkaan muodostui tarve kolmelle omalle muuttujalle: ID tietokantatunnistusta varten, Path johon voi tallentaa käskyyn liitetyt tiedostopolut ja Command johon säilötään itse komento kokonaisuudessaan. ContextID yhdistää linkin tiettyyn kontekstiin. Ref-muuttuja on lyhennelmä sanasta referance ja se tarkoittaa viittausta olioon luokasta ViPAContext. ViPAContext sisältää listan näitä ContextLink olioita, string-muotoisen nimen ja tietokantatunnisteen. Lisäksi siihen kuuluu lista konteksteista, jotka liittyvät siihen.

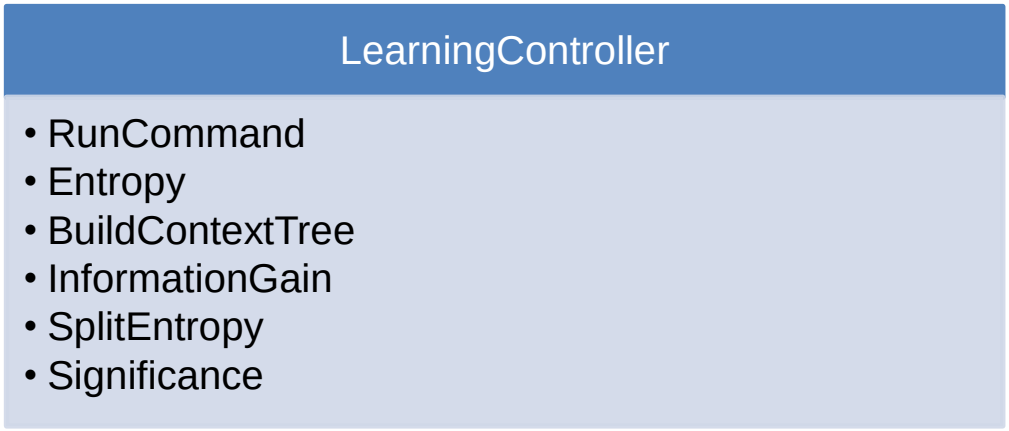


Kuva 7. ContextLink ja ViPAContext

ContextLink on ViPAContext:in käyttämä alaluokka, johon tallennetaan aikaisemmin kontekstiin liittyviä komentoja sekä tieto siitä, mihin polkuun ne on kohdistettu. Luokan tarkoitus olisi olla yksittäisiä komentoja, muistiinpanoja tai kalenterimerkintöjä, jotka on sidottu tiettyyn kontekstiin. Prototyyppiä varten siitä on tehty mahdollisimman yksinkertainen, mutta tulevaisuudessa voidaan laajentaa kyseistä luokkaa sisältämään erilaista dataa. ContextLink-luokan tarkoituksena on pitää kontekstikohtaista tietoa helpommin hallinnoitavissa kontekstin sisällä.

ViPAContext koostuu näistä ContextLink-olioista ja ala-konteksteista. Olion erottamiseen käytetään joko ID-arvoa kannan puolella tai nimeä, kun niitä käsitellään algoritmin toimesta. Kontekstin nimi on prototyypissä se arvo, jota algoritmissa tullaan vertailemaan ja jonka perusteella puurakenne tullaan muodostamaan. Jokaisella kontekstilla on lista alikonteksteista, jotka muodostavat tarvittavan puurakenteen.

Sen jälkeen kehitetään luokka oppimisalgoritmille, LearningController. Algoritmissa pitää olla ainakin seuraavat metodit: komentojen ajaminen, vertailu entropialle, kontekstien järjestely sekä yhdistäminen. Suunnitteluvaiheessa esimerkkitoteutuksia tutkiessa algoritmin ohjelmistorakenteesta nousi esiin myös metodit SplitEntropy ja Significance. SplitEntropy-metodin olisi tarkoitus laskea yksittäisen kontekstin entropia-arvo eroteltuna muista konteksteista. Significance-metodin tarkoitus olisi asettaa kertoimia arvoille, määritellen niiden arvokkuuden prosessin kannalta. Jatkokehityksessä voidaan harkita kertoimien skaalaa ja logiikkaa, prototyyppiin asetetaan vain rakenteet tätä varten.



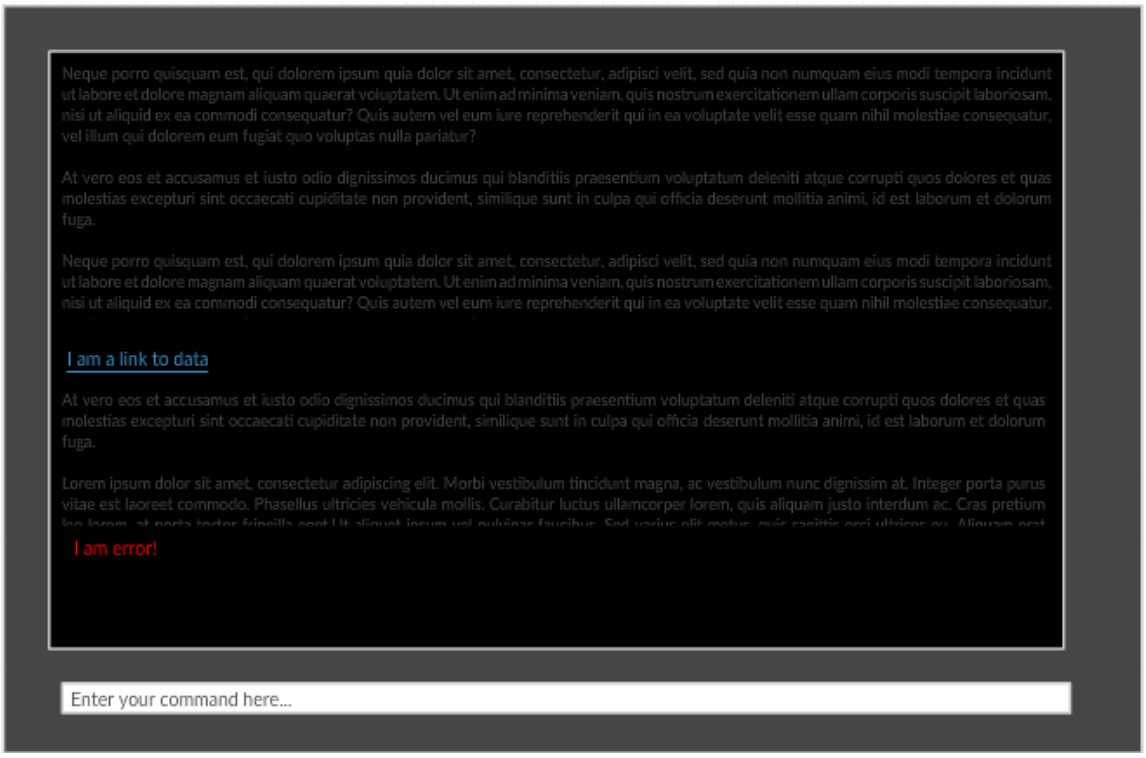
Kuva 8. LearningController

Jatkokehitystä varten tarvitaan myös luokka, joka tulkitsee annetun syötteen, CommandTranslationin. Tähän luokkaan rakennetaan puheentunnistus ja mahdolliset erikoisoperaatiot tekstisyötteelle. Tekstisyöte pitäisi tulkita ensin String-muotoon ja sen jälkeen ajaa kuten normaali komento. Luokka on rakenteen kannalta olennainen, joten se voidaan lisätä nyt, mutta siihen tulevat ominaisuudet ovat puheentunnistusta varten. Tämän tutkimuksen aikana ei vielä lähdetä perehtymään puhetutkimukseen sen laajuuden vuoksi ja mahdollisesti tullaan ulkoistamaan valmiille puheentunnistuskirjastoille tulevaisuudessa.

Tietokantayhteys tullaan toteuttamaan LDB-pohjalle, jonka rakenne luodaan ohjelman ensimmäisellä käynnistyksellä sen perusteella mikä on sen hetkinen ORM. Oliot tulee konvertoida tietokantaan, jotta vältytään monimutkaisilta SQL-lauseilta. Varsinainen toteutus jatkokehityksessä tullaan kuvaamaan luvussa 5.

4.4 Ulkoasu

Prototyypin ulkoasu tulee olemaan yksinkertainen komentorivipääte, jossa voidaan antaa ”hyperlinkkejä” esimerkiksi tiedostoihin. Näkymä koostuu seuraavista komponenteista: Input-kenttä ja Output-kenttä. Input-kenttä on yhden rivin TextBox-kenttä. Output-alue koostuu TextBox-kentästä, jota ei voi muokata sekä alueesta, jossa voidaan esittää ”tuloksia”, eli esimerkiksi linkkejä tiedostoihin. Koska prototyyppi suunnitellaan vain Windows-pohjalle, tehdään ensimmäinen UI WPF-pohjalle. Kuvassa 9 esitetään suunnitelmaa, miltä ulkoasu voisi esimerkiksi näyttää.



Kuva 9. Ulkoasu suunnitelma

Projektin UI on alustavasti olemassa, mutta tulee harkita, josko jatkokehityksessä yritettäisiin sulauttaa ViPA:n toimintaa enemmän taustalle ja pyrkiä vain yksinkertaisiin datanäkymiin eri tarkoituksille. Ui:n jatkokehitystä pohditaan tarkemmin luvussa 5.

4.5 Tietokanta

Tietokantarakenne prototyypissä pysyi LDB:nä, mutta yksi uusi luokka täytyi lisätä, kuten luvussa 4.3 mainittiin. Lisätty luokka oli ViPAContextTree, johon tallennetaan kontekstipuu. Kaksi muuta toteutettiin mallinnuksen mukaisesti, paitsi että ViPAContext-luokasta poistui SubContext-osuus. Tietokantaratkaisuja tutkittiin Holtzmanin (2) ja Rakamkrishnanin (3) kirjoista, joissa käsiteltiin koneoppimisen tietokantaratkaisuja. Näissä esitetyt esimerkit olivat kuitenkin tähän vaiheeseen turhan monimutkaisia, mutta niihin tullaan palaamaan myöhemmin.

Prototyyppiä varten luotiin nämä kaksi luokkaa, ViPAContext ja ContextLink. Koska kyseessä on LDB, on siihen yhteys hyvin ongelmaton muodostaa. Lähteeksi asetetaan vain polku .MDF-tiedostoon, joka sijaitsee projektitiedostojen yhteydessä. Kantaa pitää seurata skaalautumisen puolesta, eli kuinka nopeasti se kasvaa tiedostokoossa.

ViPAContextTree-luokka tarvittiin, koska oli hieman epäkäytännöllistä tallentaa mahdolliset liitokset erikseen jokaiselle kontekstille samaan tauluun. Luokka sisältää logiikan, joka oli SubContext-osuudessa aikaisemmin. Taulukossa 1 on esitetty, mikä on kantarakenne tälle luokalle.

ID	IsRoot	Context	SubContext
1	False	AlphaGoProject	TetraProject, HourReport
2	True	ViPA	AlphaGoProject

Taulukko 1. ViPAContextTree-luokan rakenne tietokannassa

Toteutuksen yhteydessä nousi kysymys, olisiko nopeampaa ja tehokkaampaa kuitenkin prosessoida puu ajon yhteydessä ja hakea kaikki data kerralla. Jos data haetaan joka kertaa uusiksi, tulisi myös jokaisen ajon alkaessa prosessoitua puu uusiksi ja näin optimoitua puurakennetta. Jatkokehityksessä pitää ajaa testejä isoilla tietomäärillä, ja testata onko käytännön eroa näiden kahden välillä.

.NET-kirjastossa DBO:t tulee merkitä tietyllä tavalla, jotta LINQ pystyy niitä tulkitsemaan. Tätä varten piti ViPAContext- ja ContextLink-oliot merkitä, kuten esitetään ohjelmistokoodissa 3. Merkinnät tulee asettaa aina merkittävän kohdan yläpuolelle ja hakasulkeisiin.

```
[Table(Name = "ContextLinks")]
public class ContextLink
{
            ...
    [Column(IsPrimaryKey = true, Storage = "_ID", IsDbGenerated = true,
     DbType = "Int NOT NULL IDENTITY")]
     public int ID
     {
            ...
     }

            ...
```

```
    [Association(Storage = "_Context", ThisKey = "ContextID")]
    public ViPAContext ViPAContext
    {
            ...
}
```

Ohjelmistokoodi 3. ContextLink-olion

Taulun nimi määritetään luokan alkuun ja jokaiselle attribuutille kolumnimääritykset. Niihin tulee sen alla olevan objektin määritelmä kantaan. Näiden avulla pystytään oliot kääntämään suoraan SQL:ään ilman suurempia lisätoimia. Nämä määritykset yhdistävät luokan ominaisuudet kannassa oleviin sarakkeisiin.

Ohjelmistokoodi 4:ssä esitetään esimerkkioperaatio objektin ja kannan välisestä toiminnasta koodissa. Objekti serialisoidaan kantaan LINQ-kirjastojen toimesta luokkaan määritettyjen kohtien perusteella.

```
db.ViPAContexts.InsertOnSubmit(context);

try
{
   db.SubmitChanges();
}
catch (Exception e)
{
            //ERROR HANDLING
}
```

Ohjelmistokoodi 4. Objektin syöttäminen tietokantaan.

Tietokantaoliosta kutsutaan ViPAContexts-taulua, joka sisältää kontekstioliot. Taulu-oliolle on olemassa InsertOnSubmit()-metodi, joka kirjoittaa lisätyn olion kantaan, kun submit-komento ajetaan. Olisi mahdollista myös tehdä oma serilisaatioluokka, joka hoitaisi operaation, jos halutaan vaihtaa pois LINQ-kielen käytöstä. Jatkokehityksessä voisi harkita esimerkiksi JSON-pohjaisiin kantoihin siirtymistä, jotka myös toimivat samalla periaatteella objektien tallentamisesta kantaan serilisaation avulla.

5 Jatkokehitys

Työssä käsitellään teoriaa ja prototyypin rakennetta, mutta työtä on aikaisemmin mainitun mukaan tarkoitus jatkaa. Jo suunnitellessa nousi ylös useampia jatkokehityskohteita, joita pitää toteuttaa ViPAan. Myös prototyypin toteutusta tehdessä, jouduttiin osa ominaisuuksista tiputtamaan pois, jotka lisätään jatkokehityksen yhteydessä. Tässä luvussa käsitellään asioita, mitä on suunniteltu lisättävän aikaisemmin mainittujen asioiden lisäksi.

Ulkoasua olisi tarkoitus muuttaa näyttämään tietoja sekä tuloksia selvemmin. Esimerkiksi harkittavia ominaisuuksia olisivat kuvakkeita ja informatiivisia listoja. Toinen vaihtoehto olisi pyörittää ohjelmaa prosessina tai ilmoituspalkissa, kunnes sitä kutsutaan. Tällainen näkymä toimisi vasta, kun olisi mahdollisuus joko tuoda ViPA esiin näppäinyhdistelmien kautta, tai kun puhetunnistusominaisuus on otettu käyttöön. Ulkoasun päivitykseen on harkittu vaihtoehtoina erilaisten tuloksien palauttamista omiin, niitä varten tehtyihin näkymiin. Esimerkiksi, jos haet tiedostoja, voitaisiin haun tulokset näyttää kaikki listana, josta tiedoston voi avata. Lisäksi voisi olla myös kontekstinäkymä, jossa osuvimman kontekstin tiedot olisivat helposti näkyvillä.

UI-päivityksessä haasteeksi nousee myös web-käyttöliittymien tarve. Kyky käyttää sovellusta mistä tahansa, miltä tahansa laitteelta. Yksi harkituista web-pohjaisen ohjelman toteutuksista, olisi laajentaa projektia toimimaan pilvipalvelun pohjalta, esimerkiksi Microsoftin Azure. Tämä avaisi mahdollisuuden helppoon tapaan saada ViPA toimimaan selaimen kautta. Jos ViPA kehitettäisiin toimimaan pilvipalvelun pohjalta, voitaisiin myös harkita nykyään suosittua tapaa, jossa ”verkkosivu” ladataan applikaatioon. Muutos kuitenkin luo kysymyksiä tietoturvasta ja ongelmista offline-käytössä. Vaikka kaikilla on nykyään lähtökohtaisesti pääsy internettiin, se ei aina ole mahdollista. Esimerkiksi ihmiset, jotka tekevät töitä lentokoneessa, eivät välttämättä ole lennolla jossa olisi yhteyttä.

Pilvipalvelut tarjoaisivat samalle alustalle myös erilaisia tietokantaratkaisuja ja luvussa 4.5 mietittiin vaihtoehtona JSON-pohjaisten kantoihin siirtymistä. Sopivia palveluja olisivat esimerkiksi Azuressa tarjolla oleva CosmosDB tai MongoDB, suosittu pilvitietokantapalvelu. Jatkokehityksessä otetaan selvää, mikä hyötyä olisi JSON-rakenteeseen siirtymisestä. Pilvessä oleva tietokanta avaisi mahdollisuuden saman oppimiskannan käytön usealla eri laitteella.

Työtuntikirjaus on ominaisuus, jota on kehitetty toisessa itsenäisessä projektissa, joka olisi tarkoitus yhdistää tähän projektiin. Alkuperäisen työtuntikirjausprojektin tarkoituksena oli pitää kirjaa, kuinka paljon aikaa olet käyttänyt työtehtävääsi. Ominaisuuden idea, jossa voi kertoa ViPA:lle: ”Aloitan nyt työstämään tikettiä 45B.” olisi hyödyllinen esimerkiksi töissä käyvän henkilön tarkoituksiin. ViPA pitäisi automaattisesti kirjaa käytetystä ajasta, kunnes toteat: ”valmis”. Tieto sitten sidotaan kontekstiin, johon liittyy joko tiketti-koodi tai tikettikonteksti, joka voisi olla esimerkiksi asiakkuus.

Tiedostojen organisointi voitaisiin automatisoida, kun ViPA on kerännyt tarpeeksi dataa ja ja selvä kontekstirakenne on olemassa. Ominaisuuden tarkoitus olisi sijoittaa uudet tiedostot parhaiten sopivan kontekstin alle, joka on samalla ViPA:n omaa optimointia, sillä tällaisen organisoinnin jälkeen myös ViPA toimii tehokkaammin. Ominaisuuden täytyy olla rajattu käyttäjän haluamiin alueisiin, jotta ei aiheudu ohjelmistokonflikteja. Toinen vaihtoehto organisoinnin järjestämiselle on asettaa kansioita, jotka jätetään huomiotta.

Puheentunnistus ominaisuutena mahdollistaisi käyttäjälle kyvyn käyttää ViPA:n ominaisuuksia samalla, kun hän tekee muuta. Ensimmäinen puheentunnistuksen kehitysvaihe on järkevää aloittaa siinä vaiheessa, kun on todettu, että ohjelma on kykenevä käsittelemään elävämpää kieltä. Kuten on aikaisemmin mainittu, käyttäjä voi helposti kirjoittaessa tai puhuessaan tehdä virheitä, joihin ohjelman pitää kyetä reagoimaan muutenkin kuin virheviestillä. Puheentunnistus vaatii laajaa lisätutkimusta, mutta oppimisalgoritmi antaa tälle jonkinlaista pohjaa.

6 Yhteenveto

Projektin tavoitteena oli opiskella algoritmien toimintaa ja tutkia, minkälaisia algoritmeja kannattaisi harkita tämän tyylisen projektin kannalta. Tutkimustyö vaati aikaa ja vaivaa, mutta hyvä vaihtoehto löytyi. Algoritmivaihtoehtoja ja rakenteita oli monenlaisia joita olisi voinut harkita, mutta ID3 osoittautui olemaan tutkituin ja soveltuvin. On tietenkin mahdollista, että testit ja datan määrä voivat muuttaa tarpeita algoritmille tulevaisuudessa, joten se piti ottaa huomioon rakennetta suunnitellessa. Aikaa vievin osuus tässä prosessissa oli kontekstirakenteen simulointi käsitteellisellä tasolla, mitkä kaikki arvot olivat oleellisia ja miten niitä voitaisiin käsitellä. Ongelmaksi muodostui se, että ei ole mitään selvää ratkaisua olemassa, vaan rakenne pitää kehittää sellaiselle pohjalle, että sitä voidaan helposti muuttaa tarpeen mukaan.

Projektin rakenne muotoutui hyvin projektin edetessä ja suunnitteluvaiheesta saatiin aikaiseksi hyvä rakenne, josta voidaan prototyyppiä rakentaa. Käyttötapauksien avulla pystyttiin luomaan hyvä logiikkapohja, josta on mahdollista ruveta kehittämään ohjelmistorakennetta. Ohjelmistoa päädyttiin rakentamaan MVC-mallin pohjalle, minkä lisäksi tuli kolme DBO-luokkaa sekä oma luokka algoritmille. MVC-rakenteen pohjalta on helppo lähteä tekemään muutoksia tarpeiden mukaan niin DBO-luokkiin kuin algoritmiin. Harkittavissa oleva vaihtoehto olisi, että ViPA voisi käyttää eri algoritmeja eri tilanteisiin. Algoritmin toteutus ohjelmallisesti prototyyppiin on mahdollista, sillä ID3 ja C4.5 ovat hyvin kuvattuja ja tutkittuja algoritmeja tietotekniikassa. Tietokannaksi valittiin SQL Serverin LDB, jossa kannan ja ohjelmiston välillä kommunikoidaan käyttäen LINQ-To-SQL-kirjastoa.

Tulevaisuuden kehityskohtia ovat puheentunnistus ja automatisaation lisääminen. On myös luultavaa, että tulee tilanne, jossa pitää ajaa useaa erilaista algoritmia samanaikaisesti. Joten pitää harkita miten useamman kuin yhden algoritmin kanssa voidaan toteuttaa asynkroninen toiminta. Myös ohjelman käytettävyyttä ja ulkoasua tulee päivittää, kunhan ominaisuudet sallivat joko käytön ilman näppäimistöä tai jonkin muun ratkaisun. Tästä mainittiin puheentunnistus harkittuna ominaisuutena jatkokehitykseen.

Lähteet

1 Luger, George F. 2002. Artificial Intelligence – Structures and Strategies for Complex Problem Solving, Fourth Edition. Pearson Education.

2 Holtzman, Samuel. 1988. Intelligent Decision Systems. Addison-Wesley.

3 Ramakrishnan, Raghu. 1995. Applications of logic databases. Kluwer Academic Publishers.

4 Kulkarni, Siddhivinayak. 2012. Machine learning algorithms for problem solving in computational applications.

5 Hellström, Annu. 2016. Machine learning in finance management: Case OpusCapita. Insinöörityö.

6 Xiuyang, Li. 2016. Improved AdaBoost Algorithm and Object Recognition Based On Haar-Like Training. Insinöörityö.

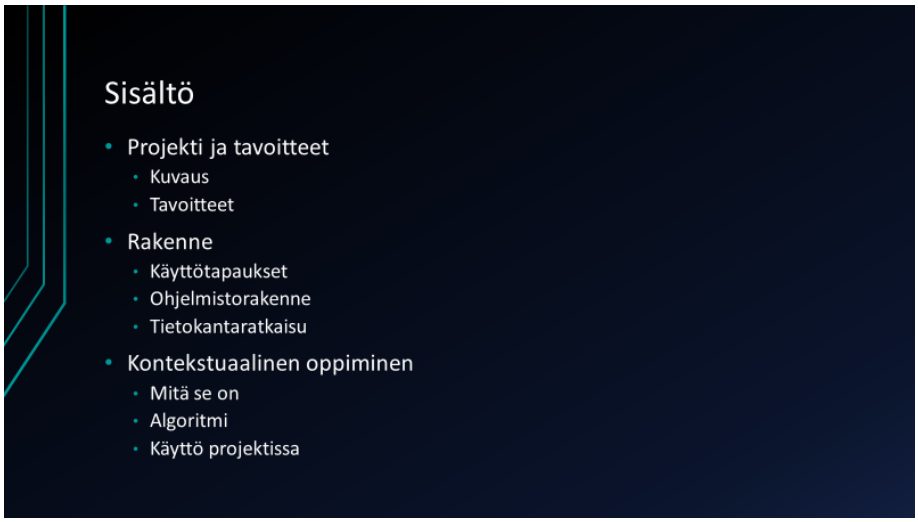
7 Hssina, Badr. 2014. A comparative study of decision tree ID3 and C4.5. Tutkimus.

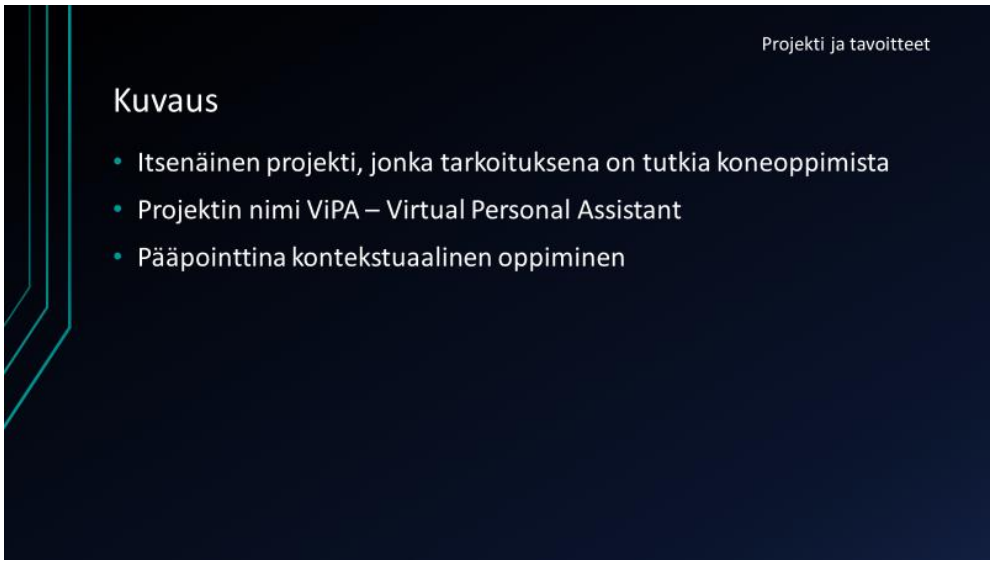
8 UNSW Engineering, 2001. Verkkodokumentti. Methods of Search. <http://www.cse.unsw.edu.au/~billw/Justsearch.html>. Luettu 25.5.2017

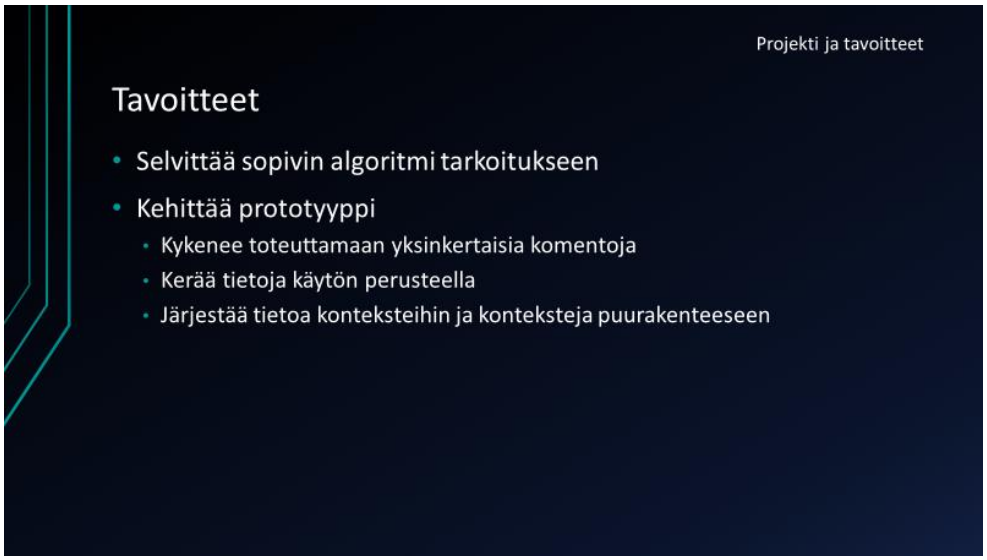
9 Maggs, B., Sitaraman, R., 2015. Algorithmic Nuggets in Content Delivery. Tutkimus.

10 Murthy, S. Kasif, S & Salzberg, S. 1994. A System for Induction of Oblique Decision Trees. Journal of Artifcial Intelligence Research 2.

Työn esitelmä




Projekti ja tavoitteet
Kuvaus
• Itsenäinen projekti, jonka tarkoituksena on tutkia koneoppimista
• Projektin nimi ViPA – Virtual Personal Assistant
• Pääpointtina kontekstuaalinen oppiminen


Projekti ja tavoitteet
Tavoitteet
• Selvittää sopivin algoritmi tarkoitukseen
• Kehittää prototyyppi
• Kykenee toteuttamaan yksinkertaisia komentoja
• Kerää tietoja käytön perusteella
• Järjestää tietoa konteksteihin ja kontekseja puurakenteeseen

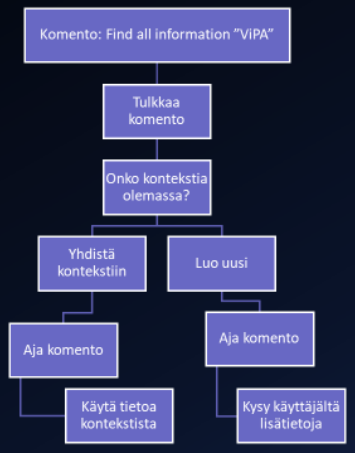

Rakenne

Käyttötapaukset

- Projektitietojen hakeminen
- Siirrä tiedostot olemassa olevaan kansioon
- Muistuta minua tapaamisesta

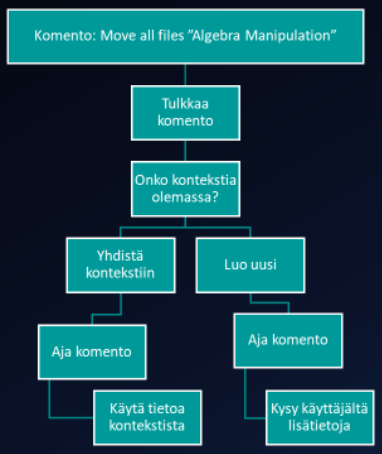
Projektitietojen hakeminen

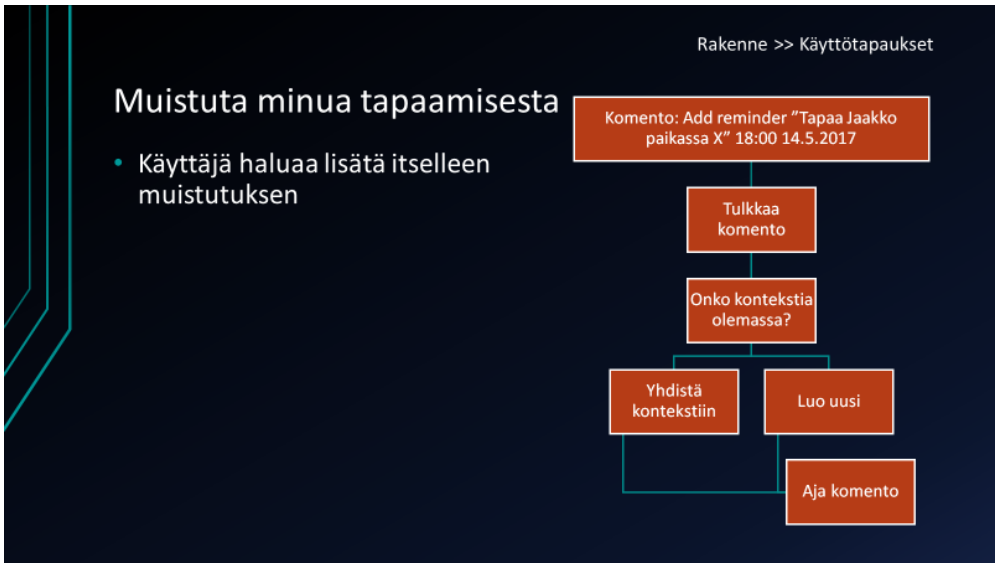
- Käyttäjä haluaa hakea kaikki tiedot projektista ViPA
- Komento paloitellaan osiin
- ViPA käy läpi aikaisemmin ajatetut kontekstit nimellä ViPA
- Jos ei löydy, tulee luoda uusi ja kysyä käyttäjältä kontekstista
- Ajaa komento ja kysyä lisätietoja, esim. mistä etsiä

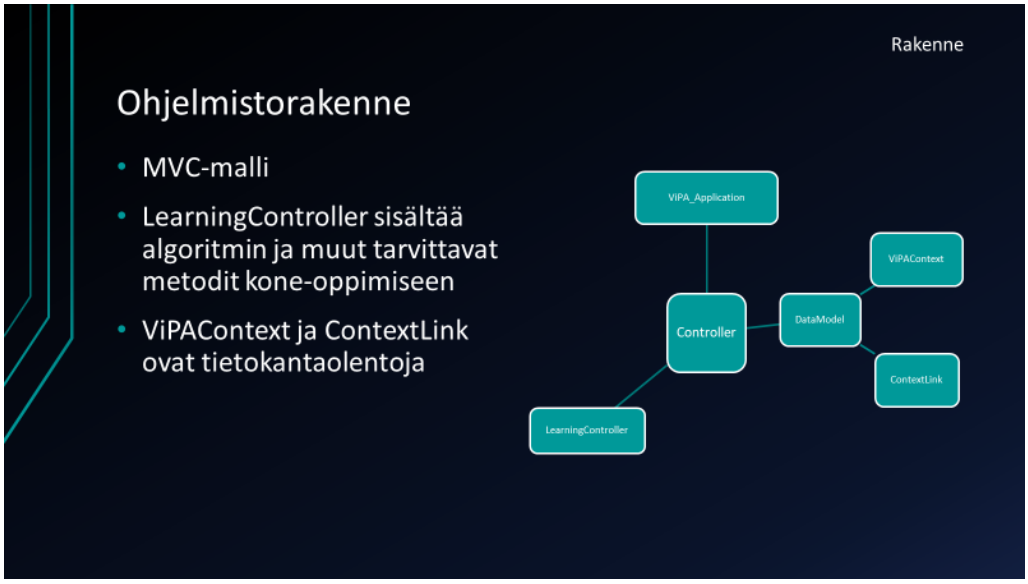


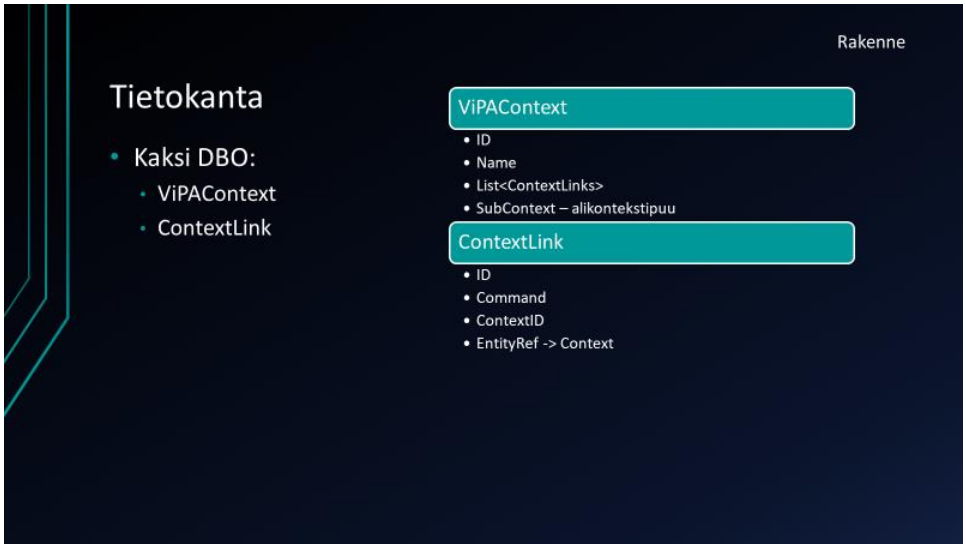
Siirrä tiedostot olemassa olevaan kansioon

- Käyttäjä haluaa liikuttaa kaikki Algebra Manipulation-projektinsa tiedostot yhteen paikkaan




Rakenne >> Käyttötapaukset
Muistuta minua tapaamisesta
Käyttäjä haluaa lisätä itselleen muistutuksen
Komento: Add reminder "Tapaa Jaakko paikassa X" 18:00 14.5.2017
Tulkkaa komento
Onko kontekstia olemassa?
Yhdistä kontekstiin
Luo uusi
Aja komento


Rakenne
Ohjelmistorakenne
MVC-malli
LearningController sisältää algoritmin ja muut tarvittavat metodit kone-oppimiseen
ViPAContext ja ContextLink ovat tietokantaolentoja
ViPA_Application
ViPAContext
Controller
DataModel
ContextLink
LearningController


Rakenne
Tietokanta
Kaksi DBO:
ViPAContext
ContextLink
ViPAContext
ID
Name
List<ContextLinks>
SubContext – alikontekstipuu
ContextLink
ID
Command
ContextID
EntityRef -> Context

Kontekstuaalinen oppiminen

Mitä se on

- Kun käyttäjä syöttää komentoja ViPAan, niitä sidotaan konteksteihin
- Jos kontekstia ei ole, sellainen luodaan
- Kontekstit järjestetään puurakenteeseen

Algoritmi

- ID3
  - Perustuu Shannonin informaatioteoriaan.
  - Lasketaan entropiaa ja informaatioarvoa
- C4.5
  - Parannus ID3-algoritmiin.
  - Hyväksyy myös tyhjää dataa
  - Ratkaisee skaalautumisongelmia
  - Käyttää karsimista

$$I[\mathbf{M}] = -\sum_{i=1}^{n} p(m_i)\log_2(p(m_i)) = E[-\log_2 p(m_i)]$$

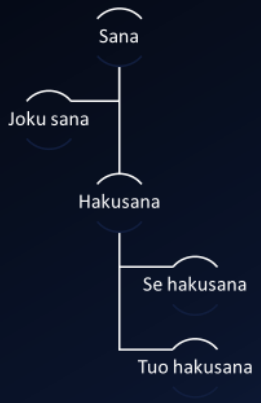
$$\mathbf{E}[\mathbf{P}] = \sum_{i=1}^{n} \frac{|C_i|}{|C|} I[C_i]$$

$$gain(P) = \mathbf{I}[\mathbf{C}] - \mathbf{E}[\mathbf{P}]$$

Kontekstuaalinen oppiminen

Käyttö projektissa

- Logiikka järjestelyyn
  - Yksittäinen sana on juuri
  - Sen alle sijoittuu sanat tai lauseet joissa sana esiintyy
- Karsimista täytyy miettiä jatkokehityksessä



Yhteenveto

- Projektin tavoitteet ja suunnitelma täytettyjä
- Käyttötapauksien avulla lähestyttiin rakennetta
- Ohjelmisto – ja tietokantarakenne yksinkertaisia prototyyppia varten
- Algoritmina C4.5 – perustuu entropiaa ja informaatiokasvuun
- Miten projektissa toimii