KYMENLAAKSON AMMATTIKORKEAKOULU
Tietotekniikka / Ohjelmistotekniikka

Jussi Raunio

MOBIILISATAMAN TIETOKONEAVUSTETTU LIIKENNEJÄRJESTELMÄ

Opinnäytetyö 2012

TIIVISTELMÄ

KYMENLAAKSON AMMATTIKORKEAKOULU
Tietotekniikka

RAUNIO, JUSSI

MOBIILISATAMAN TIETOKONEAVUSTETTU LIIKENNEJÄRJESTELMÄ

Opinnäytetyö 34 sivua + 30 liitesivua
Työn ohjaaja Yliopettaja Paula Posio
Toimeksiantaja North European Logistics Institute
Toukokuu 2012
Avainsanat NP-täydellisyys, web-ohjelmointi, reittihaku, polunetsintä

Tietotekniikan näkökulmasta logistiikan ongelmiin vastaaminen on erittäin haastavaa. Useat logistiikan ongelmat ovat NP-täydellisiä ja niiden täydellinen ratkaisu on lähes mahdotonta tietoteknisesti. Yleensä käytetään tietokoneavusteisia logistisia järjestelmiä, jossa ratkaistaan osa logistisesta kokonaisuudesta. Osittaisratkaisussa ongelman täydelliseen ratkaisuun ei pyritä, joten se vaatii aina ihmisen tekemään lopullisen päätöksen.

Työn tavoitteena oli tutkia tietokoneavusteisten logististen kuljetusmahdollisuuksien etsimistä ja parantamista HaminaKotka satamassa. Tutkimuksen tulos toteutettiin web-sovelluksena. Sovelluksella on mahdollista tarkastella virtuaalista karttanäkymää satama-alueesta, säilöä tietokantaan satamakalustojen tietoja ja suorittaa tiestön ominaisuuksiin mukautuvaa reittihakua minkä tahansa pisteiden välillä satama-alueen sisällä. Kartan tiestön ja reittihaun ominaisuuksien mukauttaminen on mahdollista ajonaikaisesti.

Sovellus toteutettiin Java-ohjelmointikielellä Liferay-palvelinympäristöön. Käyttöliittymän ohjelmointiin käytettiin Vaadin-kirjastoa ja Google Web Toolkitia. Tietokanta rakennettiin käyttäen Liferayn palvelurakentajaa ja tietokannan hallintaan käytettiin Liferayn tarjoamia rajapintoja. Karttanäkymä satama-alueesta on tallennettu vektorigrafiikkamuodossa, josta ajonaikaisesti luodaan käyttöliittymässä näytettävät bittikarttamuotoiset lohkokuvat. Kaikki kartan tiestöt on tallennettu tietokantaan, jotta niiden käsittely ja ylläpito ohjelmallisesti olisi helpompaa.

Lopputuloksena saatiin toimiva sovellus, joka ratkaisee osan logistisista kuljetusongelmista. Se on toteutettu käyttäen hyviä oliosuunnittelun periaatteita, joten sen ylläpito ja jatkokehitys on helppoa. Lisäksi toteutuksessa on noudatettu JSR 286 -portlettistandardin alustariippumattomuuden periaatteita, jotta sovelluksen siirto muihin palvelinalustoihin olisi mahdollista. Sovellus integroidaan osaksi Mobiilisatama-hankkeen informaatiokeskusta, jossa sitä käytetään satama-alueen sisällä tapahtuvien erikoiskuljetuksien suunnitteluun.

ABSTRACT

KYMENLAAKSON AMMATTIKORKEAKOULU
University of Applied Sciences
Information Technology

RAUNIO, JUSSI	Computer Aided Transportation System in Port Environment
Bachelor's Thesis	34 pages + 30 pages of appendices
Supervisor	Paula Posio, Senior Lecturer
Commissioned by	North European Logistics Institute
May 2012	
Keywords	NP-completeness, web programming, routing, pathfinding

In the perspective of information technology, logistics problems are very challenging. Many logistics problems are NP-complete and fully solving them computationally is almost impossible. Often, computer aided logistics systems are used where a part of the logistics problems is solved computationally. The partial solution does not seek a complete solution of the problem, and a human is always required to find the final solution.

The aim of this work was to populate possibilities of solving transportation problems in the Port of HaminaKotka with computer aided problem solving. A web based application that applies those research principles in practice was developed along this study. The application allows users to view a virtual map view of the port area, use the database to store information about available port equipment and do routing that adapts to properties of roads in the port area. Modifying map routes and routing parameters is possible at runtime.

The application was programmed with the Java programming language and deployed to Liferay server environment. Vaadin library and Google Web Toolkit were used for user interface programming. The database was built by using Liferay's Service Builder and database was accessed by using interfaces provided by Liferay. The map view of the port area was stored in vector graphics file format where bitmap tile images for user interface use are generated at runtime. Every road of the map was stored in the database for simplifying data processing and maintenance.

The result of the work was an application that solves a part of the problems related to transportation. The application was developed by using the best practices of the object oriented software designing ensuring easy maintenance and further development. In addition, the software respects cross-platform and portability principles defined by the JSR 286 portlet standard. The application will be integrated into the information center of the Mobile Port project as a part of it.

SISÄLLYS

1 TERMIT JA LYHENTEET

Ajax	Akronyymi sanoista ***A**synchronous **J**avaScript **A**nd **X**ML* tarkoittaa palvelimen ja selaimen välillä tapahtuvaa saumatonta keskustelua. Alkuperästään huolimatta, ei välttämättä rajoitu vain JavaScriptiin ja XML:ään.
Emergenssi	Kuvastaa ilmiötä, jossa jo olemassa olevaan monimutkaiseen systeemiin ilmaantuu uusia ominaisuuksia ilman erillisiä ponnisteluja tai suunnitteluja.
Heuristiikkafunktio	Funktio, jota käytetään algoritmeissa ennustamaan seuraavaa suoritusvaihetta.
JSON	Akronyymi sanoista ***J**ava**S**cript **O**bject **N**otation*. Se on JavaScript-kielestä peräisin oleva objektien esitysmuoto, jota käytetään puurakenteellisen tiedon kuvaamiseen. Nimestään huolimatta, sitä käytetään yksinkertaisuutensa vuoksi laajalti tiedonsiirtoon XML:n tapaan.
NP-täydellisyys	Ongelmat, jotka ovat laskennallisesti erittäin vaativia. Ongelmiin ei ole olemassa tunnettuja polynomiajallisia ratkaisualgoritmeja.
SVG	Lyhenne sanoista ***S**calable **V**ector **G**raphics*. Se on kaksiulotteisen vektorigrafiikan XML-pohjainen kuvauskieli. Moni ohjelmisto tukee SVG:tä suoraan, mukaan lukien useat WWW-selaimet.
XML	Lyhenne sanoista *EX**t**ensible **M**arkup **L**anguage*. Se on standardisoitunut merkintäkieli, jota käytetään usein alustariippumattomaan tiedonvälitykseen ja puurakenteellisen tiedon kuvaamiseen.

2 JOHDANTO

Opinnäytetyö on toteutettu Mobiilisatama-hankkeen yhteydessä. Sovelluksen määrittely ja taustatutkimus alkoivat lokakuussa 2011 ja sovellus toteutettiin vuoden 2012 aikana. Työn tulos integroidaan osaksi Mobiilisatama-hankkeen informaatiokeskusta.

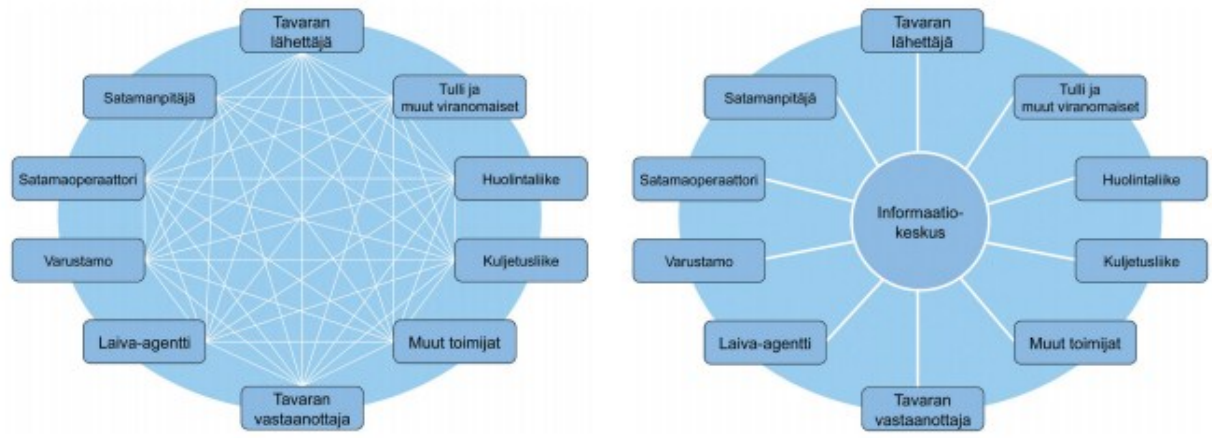
Työssä on tarkoitus tutkia mahdollisuuksia helpottaa sataman logististen kuljetusmahdollisuuksien etsintää tietokoneavusteisesti HaminaKotka satamassa. Toteutettu sovellus sisältää pääpiirteissään virtuaalisen karttanäkymän satama-alueesta, kuljetuskalustojen ja muiden satamakalustojen lisäämisen, poistamisen ja muokkaamisen sekä kaluston ominaisuuksia hyödyntävän reittihaun sataman sisällä tavaran liikuttelemiseen. Informaatiokeskus on Liferay-alustalla toimiva web-pohjainen portaali, johon sovellus integroidaan servleteinä ja portleteina.

Luvussa 3 esitellään Mobiilisatama-hankkeen taustat ja tavoitteet. Luku 4 käsittelee logistiikkaa nykypäivän yritysmaailman näkökulmasta, tietotekniikan mahdollistamia logistia ratkaisuja sekä tässä työssä käsiteltyjen asioiden lähtökohdan ja tulevaisuuden näkymät. Luvussa paneudutaan myös logistiikan tuomiin haasteisiin ja tutustutaan NP-täydellisyyteen. Luvussa 5 käsitellään web-sovellusten taustaa ja niiden tämänhetkistä asemaa. Lisäksi luvussa käydään läpi tässä työssä käytetyt tekniikat ja niiden hyödyntämistä yleisellä tasolla. Luku 6 käsittää sovelluksen toiminnan sekä sovelluksen taustalla olevan teorian läpikäynnin ja sen hyödyntämisen käytäntöön. Luku 7 päättää opinnäytetyön ja esittää loppupäätelmän työlle.

3 MOBIILISATAMA-HANKE

Mobiilisatama-hankkeen tarkoitus on luoda informaatiokeskus satamatoimijoiden tiedonvaihtoon. Sen tarkoitus on toimia lennonjohtokeskuksen tapaan, jossa sataman toimintaa hallitaan keskitetysti. Keskitetyn hallintakeskuksen tavoite on lisätä satamaliikenteen sujuvuutta, vähentää onnettomuusriskejä, estää ruuhkautumista ja pienentää päästöjä. Uusi toimintamalli nostaa myös satamamaakunnan kilpailukykyä ja kustannustehokkuutta. Tällaista sataman informaatiokeskusta ei ole käytössä vielä Suomessa, mutta ulkomailta vastaavia ratkaisuja löytyy. Hankkeen pilottina on Kotkan alueen satamat. (1.)

Hanke tarkastelee koko satamatoimintaa kokonaisuutena eri toimijoiden näkökulmasta. Tiedonvaihto ja tiedon luotettavuus toimijoiden kesken on ensiarvoisen tärkeässä osassa keskuksen toimintaa. Mobiilisatamassa pääkohtina on toimijoiden tiedonvaihdon keskittäminen, jossa jokainen toimija saa tietoa sataman nykyisestä tilasta reaaliajassa. Muun muassa häiriöistä satamassa voidaan viestiä kaikille informaatiokeskuksen kautta erittäin nopeasti. Tavoitteena on myös saada keskus toimimaan rajajonotusjärjestelmänä, jossa keskukseen voisi ilmoittaa etukäteen rajalle tulevasta liikenteestä (2). Klassinen tiedonvälitys satamassa on hajautunut, eli jokainen toimija välittää tiedon suoraan toiselle toimijalle. Informaatiokeskuksessa tieto on keskitetysti kaikkien saatavilla. Keskus on myös aina ajan tasalla automaattisesti, kun eri toimijat käyttävät sitä ensisijaisesti tiedonvälitykseen. Kuva 1 esittää klassisen tiedonvälityksen satamassa ja sitä vastaavan informaatiokeskusmallin.



Kuva 1: Klassinen, hajautunut tiedonvälitys satamassa (vasemmalla) ja keskitetty informaatiokeskusmalli. (1.)

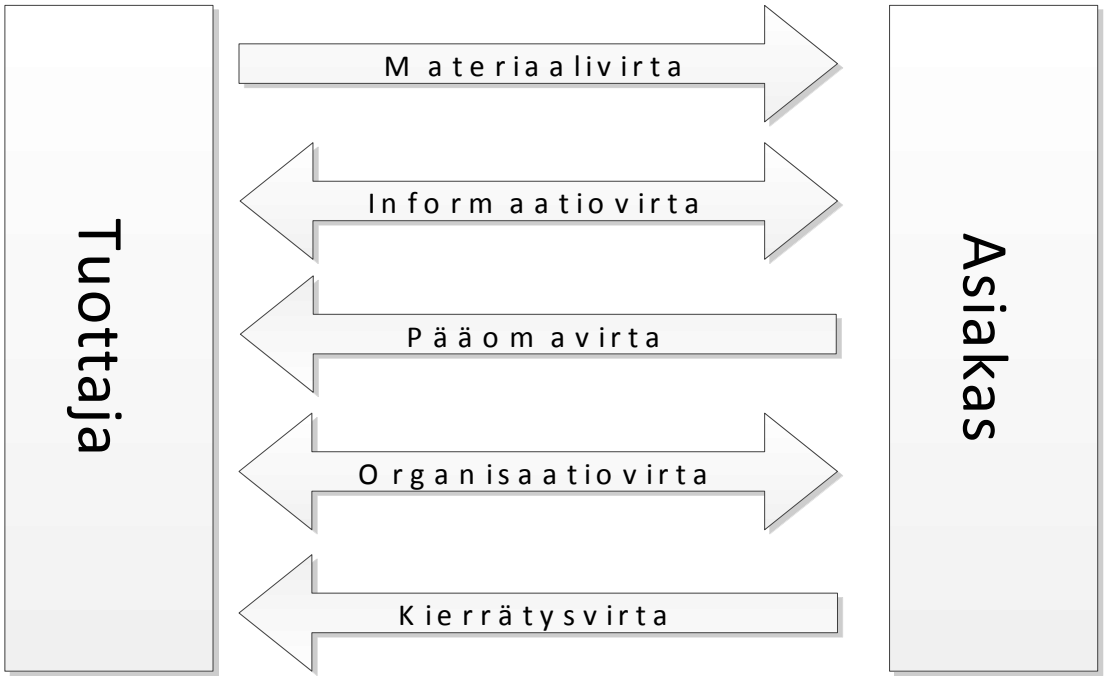
Hanke aloitettiin vuoden 2009 elokuussa ja sen tutkimusosuus päättyi vuoden 2011 lopussa. Yhteistyöhankkeessa on mukana Turun yliopiston Merenkulkualan koulutus- ja tutkimuskeskus (MKK), Kymenlaakson ammattikorkeakoulu (KyAMK), Lappeenrannan teknillisen yliopiston Kouvolan yksikkö ja Merikotka-tutkimuskeskus. Rahoittajina hankkeessa toimii Euroopan aluekehitysrahasto ja Tekes sekä lukuisia yrityksiä. (2.)

4 TIETOTEKNIIKAN MAHDOLLISTAMAT LOGISTISET RATKAISUT

Logistiikka määritellään usein tuotannollisen, materiaali- ja tietovirtojen sekä asiakasvirtojen hallintaan erikoistuneeksi tieteenhaaraksi (3, 1-5). Logistiikka jakautuu moneksi osa-alueeksi, jotka käsittävät eri osia logistisesta prosessista. Logistiikka käsittää

koko informaatioketjun valmistajalta asiakkaalle asti. Se ottaa myös huomioon yritysten väliset verkostoitumiset, organisaatiotason johtamisen ja kassavirrat (3, 1-2). Logistiikka on laaja määritelmä, joka tarkoittaa kaikkia näitä tai vain osaa logistisesta prosessista. Terminä logistiikkaa pidetään uutena, vaikka varsinainen prosessi on ollut olemassa jo pitkään. Käsite on kehittynyt 1960-luvulta asti aina 1990-luvulle, jolloin moni logistiikan osa-alue yhdistettiin saman termin alle. Samaan aikaan logistiikka tuli vahvasti osaksi yritystoimintaa ja johtamista. Määritelmä eroaa hyvinkin paljon riippuen tarkastelijasta: kauppias näkee logistiikan eri tavalla kuin yrityksen johtaja. Tarkastelijat tarkastelevat logistiikkaa omasta sijainnistaan lähtöisin, jonka keskikohtana he ovat. Ei voida siis määrittää tarkkaa logistista ketjua, koska sen alku- ja päätepisteet vaihtelevat tarkastelijasta riippuen. Varsinainen prosessi on käyttäjäläheinen, koska logistiikan tarkoitus on varmistaa informaatiovirran paras mahdollinen ohjaus toimitusketjun läpi asiakkaalle. Logistiikka on siis aktiivinen prosessi organisaation ja asiakkaan välillä. (4, 1-3.)

Logistiikka koostuu eri tietovirroista. Alla olevassa kuvassa (Kuva 2) on kuvattu Niemen mukaan keskeisimmät logistiset virrat.



Kuva 2: Koska logistiikka on prosessi, voidaan sitä tarkastella mistä tahansa prosessin kohdasta. Kuvassa on tarkasteltu logistisia virtoja tuottajan ja asiakkaan välillä. (4, 6.)

Ketju kulkee ylhäältä alaspäin. Materiaalivirta alkaa alkutuotannosta ja päätyy loppukäyttäjille lopullisina tuotteina. Kun tuotteen elinkaari päättyy, alkaa sama prosessi uudelleen kierrätysvirran kautta, eli tuotteesta jalostetaan uusi tuote. Osa virroista on

palautteellisia, eli virta kulkee molempiin suuntiin. Informaatiovirta sisältää materiaali- ja pääomavirtojen hallintaan tarvittavan tiedonkulun. Pääomavirta kulkee ketjussa vastakkaiseen suuntaan kuin materiaalivirta. Se kuvastaa sitä, että kun prosessi tuottaa jotakin materiaalia, palautuu sen arvo takaisin tuottajalle eri muodossa. Samaten kierrätysvirta kulkee vastakkaiseen suuntaan, eli jo tuotetun tuotteen arvo palautuu takaisin prosessiin uudelleenjalostettavaksi. Organisaatiovirta on myös palautteellinen virta, joka käsittää yritysten väliset suhdanteet. Esimerkiksi kilpailutilanne palautuu organisaatiovirran kautta takaisin tuottajalle asiakkaan ostokäyttäytymisen perusteella. (3, 3.)

4.1 Graafit

Tietotekniikassa graafeille on monenlaisia sovellutuksia, kuten reittihaku, tiedon organisointi ja päätöksenteko. Tässä opinnäytetyössä on tarkasteltu graafeja vain oleellisilta osin reittihaun näkökulmasta. Reittihaussa on tärkeää, että potentiaalisia reittejä voidaan löytää ja hallita mielekkäästi. Tarvitaan esitystapa, joka on intuitiivinen ja jonka esittäminen on tietoteknisesti tehokasta. Graafien ominaisuuksia voidaan hyödyntää hyvin reittihaussa – onhan niiden teoriakin lähtöisin samankaltaisesta ongelmasta.

Graafit muodostuvat pisteistä ja viivoista. Matemaattisesti ne koostuvat pareista (V, E), missä V on pistejoukko ja E on pisteiden muodostamien kaarien joukko. Graafiteorissa on määritelty **polut** ja **piirit**, jotka ovat **reittejä** eli kulkutapoja graafin läpi. Reitti määritellään siten, että siinä esiintyy yksi kaari vain kerran. Polussa esiintyy reitin jokainen piste vain kerran. Piiri on samankaltainen kuin polku, mutta se on suljettu eli sen alku- ja loppupisteet ovat samat. Graafin pisteet voivat olla myös suunnattuja, joka tarkoittaa, että niitä voidaan kulkea vain yhteen suuntaan. Lisäksi kaaret voivat olla painotettuja, joten kulkua niiden kautta muihin kaariin nähden voidaan mitata. (5, 1-10.)

4.2 NP-täydellisyys ja logistiikan haasteet

Tietotekniikan näkökulmasta logistiikan ongelmiin vastaaminen on erittäin haastavaa, koska moneen ongelmaan ei ole polynomiajallista ratkaisua tiedossa tai edes olemassa (6, 107). Esimerkiksi pysähtymisongelmassa algoritmin tulisi selvittää, loppuuko annetun ohjelman suoritus koskaan. Ongelma on mahdoton ratkaista algoritmisesti (6,

106). Monet logistiikan ongelmat ovatkin NP-täydellisiä. NP-täydellisten ongelmien luonteen mukaan tunnetaan vain ei-polynomiajallisia ratkaisualgoritmeja, jotka ovat käyttökelvottomia isolla, oikean elämän datamäärällä. Tietoteknillisesti ei ole edes mielekästä yrittää ratkaista kaikkia logistiikkaan liittyviä ongelmia, koska osa prosessin vaiheista on niin monimutkaisia, ettei nykyinen tietämys riitä niiden ratkaisemiseen. Siksi tietotekniikassa logistiikkaongelmat usein käsittävät vain tavaran kuljetusongelmia ja niiden osittaisratkaisuja. Nykyinen tietotekniikka pystyy pureutumaan siis vain hyvin pieneen osaan logistisesta prosessista.

Ehkä tunnetuin NP-täydellinen ongelma tulee myös logistiikan alalta. Kauppamatkustajan ongelmassa on kuvitteellinen kauppiashenkilö, jonka täytyy käydä ennalta määritetyissä kaupungeissa kauppaamassa tuotteitaan. Ongelmassa kauppamatkustaja tietää kaupunkien väliset etäisyydet, mutta ei tiedä lyhintä reittiä, joka käy kaikissa kaupungeissa ja palaa takaisin kotikaupunkiin (lähtöpisteeseen) (6, 110). Ihmiselle tällaisen ongelman ratkaisu on osana arkipäivää, ja siihen löytää *riittävän hyvän* ratkaisu muutamassa minuutissa lähes täysin riippumatta kaupunkien määrästä. Moni ohjelmoija keksii myös välittömästi triviaalin ja ainoan virallisesti tunnetun algoritmin ongelmaan: käydään kaikki kaupungit läpi ja valitaan lopuksi lyhin reitti. Tämän algoritmin aikavaatimus kasvaa kuitenkin eksponentiaalisesti kaupunkien lukumäärän kasvaessa. Alla olevassa taulukossa (Taulukko 1) on esimerkkejä kaupunkien lukumääristä ja lukumäärää vastaavista reittien lukumäärästä. Kaikki reitit olisi tutkittava ennen kuin voitaisiin varmasti todeta lyhin reitti.

Taulukko 1: Kauppamatkustajan ongelma: tukittavien reittien lukumäärän eksponentiaalinen kasvu, kun kaupunkien määrä kasvaa.

Kaupunkien lukumäärä	Mahdollisten reittien lukumäärä
4	6
8	5 040
10	362 880
12	39 916 800
14	6 227 020 800
16	1 307 674 368 000

Matemaattisesti tarkoitus on etsiä painotetulle graafille täydellinen Hamiltonin piiri, jolla on pienin kokonaispaino. Kuten edellä todettiin, mahdollisia ratkaisuvaihtoehtoja

on $(n-1)!$ kappaletta, jossa n on solmujen (tässä: kaupunkien) määrä. Ongelman suurusluokka kasvaa eksponentiaalisesti, kun uusia solmuja lisätään graafiin. Lisäksi tässä ei ole otettu huomioon logistiikassa tyypillisiä muita muuttujia, kuten kustannuksia, pysähdysten määrää tai matkan kokonaiskestoa. Ne luonnollisesti monimutkaistavat ongelmaa entisestään. On myös syytä huomata, että logistiikassa paras reitti ei välttämättä ole maantieteellisesti lyhin ja voi joissakin tapauksissa vaatia tietyssä solmussa käymistä useamman kuin yhden kerran.

Useat NP-täydelliset ongelmat ovat muodossa, jotka ovat klassiselle tietojenkäsittelylle erittäin vaativia. Tietojenkäsittelyn perustana on eksakti laskenta, joka noudattaa täsmällisesti ohjelmoituja ohjeita. Monet arkipäiväiset ongelmat ovat myös NP-täydellisiä, joten näissä ongelmissa tietotekniikan on täytynyt suuntautua kohti *riittävän hyviä* ratkaisuja. Parhaan mahdollisen ratkaisun hakeminen ei ole mielekästä tai edes mahdollista annetuilla resursseilla. Viime aikoina onkin ollut puhetta siitä, että kaikkiin yksinkertaisiin ongelmiin olisi löydetty jo parhaat mahdolliset tietotekniset ratkaisut. Jäljellä on suuri joukko arkisia NP-täydellisiä ongelmia, joiden ratkaisu vaatii täysin uutta teknologiaa ja lähestymistapaa. Jo pitkään kehitteillä olleet kvanttitietokoneet suuntaavat ongelmien ratkaisuun klassisin keinoin, tuhatkertaistamalla laskentatehon. NP-täydellisten ongelmien luonteen takia kvanttitietokoneita ei voida pitää täydellisenä ratkaisuna, koska ongelmien suuruusluokkaa voidaan kasvattaa helposti, joten myös kvanttitietokoneiden laskentateho loppuu tietyssä pisteessä.

Ongelmia on lähestytty myös eri näkökulmista kehittämällä uudenlaisia tekniikoita riittävän hyvien ratkaisujen hakemiseen. Useat lähestymistavat ovat peräisin biologiasta. Esimerkiksi jo 1940-luvulla esiteltiin ensimmäiset versiot matemaattisista neuroverkoista (7). Neuroverkkojen tarkoitus on matkia biologisten neuroverkkojen, aivojen, piirteitä sekä toimintaa ja niiden tapaa prosessoida tietoa. Neuroverkkoja voidaan käyttää signaalinprosessointiin, jossa verkon sisään annetaan tietty määrä sisääntuloja ja lopuksi verkon läpi kulkeneet signaalit muodostavat ulostulon (7). Signaali muuttaa muotoaan, kun se kulkee verkon läpi ja siten on mahdollista myös opettaa verkkoa. Esimerkiksi neuroverkkoja voitaisiin käyttää kuvantunnistukseen ja robottiteknologissa. Neuroverkot olivat yksi ensimmäisistä tekoälytieteen oppivista systeemeistä, vaikka niitä pystyttiin opettamaan vasta 1980-luvulla tehokkaasti (7). Potentiaalistaan huolimatta, neuroverkkoja ei ole vielä onnistuttu valjastamaan arkipäiväisten ongelmien ratkaisuun.

Biologinen lähestymistapa on tuottanut myös monia muita algoritmeja erinäisten ongelmien ratkaisuun. Useat näistä perustuvat eläinten parvikäyttäytymiseen ja niiden tapaan populoida useita potentiaalisia ratkaisuja, joista lopulta emergoituu paras. Esimerkiksi mehiläisalgoritmi (*Bees algorithm*) on hakualgoritmi, joka matkii mehiläisten tapaa etsiä mettä parvessa. Algoritmi käyttää hyväkseen faktaa, jossa moni mehiläinen etsii mettä lentäen isoilla ja satunnaisilla alueilla. Mehiläiset käyttävät suunnistamiseen ilmaan pölyttynyttä siitepölyä. Meden löytämisen jälkeen mehiläiset lentävät takaisin pesään, mutta samalla ne levittävät kukasta lähtenyttä siitepölyä ilmaan. Lyhimmän reitin löytäneet mehiläiset ehtivät käydä hakemassa mettä ja pesässä useamman kerran, joten ilmaan jää enemmän siitepölyä juuri tälle välille. Periaatteessa vähän ajan kuluttua kaikkien mehiläisten pitäisi ruveta käyttämään tätä lyhintä reittiä automaattisesti. Huomionarvoista on, ettei näin löytynyt reitti välttämättä ole paras mahdollinen, mutta se on riittävän hyvä juuri sillä hetkellä. (8, 6-13.)

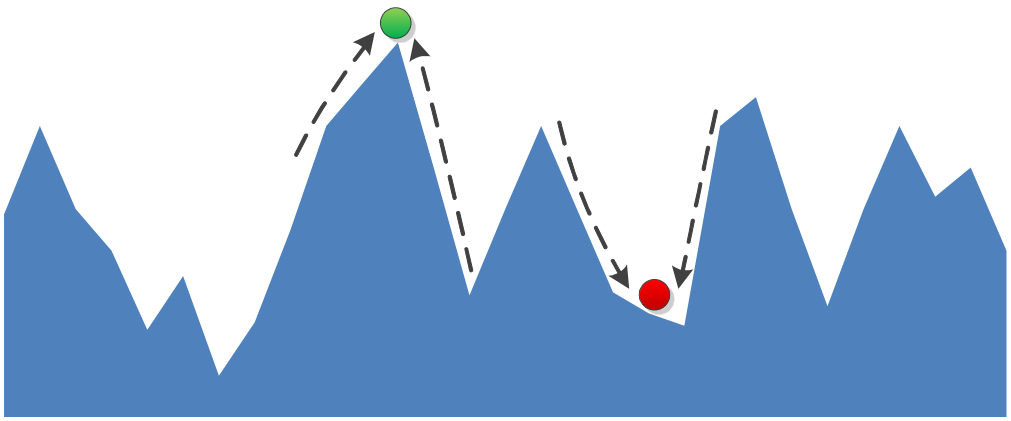
Kauppamatkustajan ongelmaa on tutkittu hyvin paljon ja sille on olemassa useita erilaisia ratkaisumalleja, jotka antavat suurellekin datamäärälle lähes täydellisen ratkaisun hyvin nopeasti. NP-täydellisten ongelmien uudet ratkaisuehdotelmat tuovat aina uusia mahdollisuuksia optimoida logistista kuljetusketjua. Kauppamatkustajan ongelma on läsnä kuljetuslogistiikassa joka päivä, mutta esiintyy siellä paljon monimutkaisemmassa muodossa. Esimerkiksi tavaroiden tehokas kuljettaminen tehtaalta tukkureille monien satojen rekka-autojen voimin monistaa kauppamatkustajan ongelman moninkertaisesti. Lisäksi tässä on otettava huomioon muun muassa polttoainekustannukset, tavaran säilymisajat, kuljettajien tauot, lastauspisteet, tiestön rajoitukset ja monet muut muuttujat, jotka kasvattavat ongelman suuruusluokkaa entisestään. Usein käytetäänkin tietokoneavusteisia logistiikkaratkaisuja. Algoritmi ratkaisee tietyn osan ongelmasta ja luo mahdollisen osittaisratkaisun kokonaisongelmalle, jonka perusteella sovelluksen käyttäjä voi tehdä tarvittavat päätelmät ja tehdä lopullisen ratkaisun. Tulevaisuudessa tulemme ehkä näkemään oppivia logistisia systeemejä, jotka auttavat ongelmien ratkaisussa kaikilla logistiikan osa-alueilla ja löytävät riittävän hyviä ratkaisuja nopeasti ilman ihmisen apua.

4.3 Ratkaisun hyvyyden arviointi

Ongelmia ratkaistaessa on tärkeää pystyä vertailemaan ratkaisun hyvyyttä muihin ratkaisuihin nähden. Kauppamatkustajan ongelman yhden ratkaisun hyvyys voisi olla

esimerkiksi reitin kokonaispituus, joka koostuu reitin painojen yhteenlasketuista arvoista. Hyvyyden arviointia voidaan tarkastella kahdella tapaa: yksittäisen ratkaisun soveltuvuus tai yksittäisen ratkaisun toteuttamisen kustannus. Soveltuvuudella mitataan, kuinka hyvin ratkaisu ratkaisee ongelman, kun taas kustannuksilla mitataan, kuinka paljon ratkaisun toteuttaminen maksaisi resursseissa. Nämä kaksi tapaa mittaavat samaa asiaa, mutta niiden lähtökohta on erilainen. Soveltuvuudella mitataan nimenomaan ratkaisun hyvyyttä, joten $f(x) \to \infty$, missä f on **hyvyysfunktio**. Kustannuksilla mitataan ratkaisun toteuttamisen hintaa, joten $f(x) \to 0$, missä f on **kustannusfunktio**. Oleellinen ero näiden välillä on, että hyvyyttä on mielekäs mitata vain potentiaalisten ratkaisuehdotelmien joukossa. Kustannuksilla yleensä mitataan ratkaisun yleistä hintaa, joka ei vaihtele potentiaalisten ratkaisujen määrästä riippuen. Hyvyyden mittaustyylin valinta riippuu käsiteltävästä ongelmasta. Se on kuitenkin tärkeässä osassa ongelman ratkaisujen löytymisessä.

Algoritmien hyvyyden arviointifunktioissa on otettava huomioon paikalliset minimit ja maksimit, jotta algoritmi löytää hyviä ratkaisuja. Tämä pätee varsinkin silloin, kun yritetään ratkaista epälineaarista ongelmaan, jossa on paljon paikallisia huippuja. Algoritmin ei tulisi jumittua paikallisiin arvoihin, vaan ratkaisussa pitäisi suunnata kohti globaalia minimiä tai maksimia (Kuva 3). Täydellinen algoritmi löytää aina ratkaisun, jos sellainen on olemassa, kun taas optimaalinen algoritmi löytää aina ratkaisun globaalin minimin tai maksimin. Paikallisten huippujen erottaminen globaaleista on vaikeaa, ja siihen on olemassa erilaisia tekniikoita. Heuristiikka auttaa algoritmia pois paikallisista huipuista, mutta hyvän heuristiikan lisääminen arviointifunktioon vaatii ongelman tarkkaa analysointia ja optimointia. (9, 111.)



Kuva 3: Epälineaarisessa ongelmassa funktion pitää arvioida ongelman ratkaisun globaalia hyvyyttä. Kuvassa ylempi pallo kuvastaa algoritmia, joka on löytänyt ongelman globaalin maksimin. Alempi pallo kuvastaa algoritmia, joka on jumittunut paikalliseen minimiin.

Heuristiikkafunktioiden määrittämisessä täytyy huomioida muutama seikka, jotta funktio todella parantaa algoritmin suoritusta. Väärin käytettynä heuristiikka voi huonontaa algoritmin suoritustehoa dramaattisesti. Hyvä (engl. admissible) heuristiikkafunktio ei koskaan yliarvioi todellista hyvyyttä, joten todellinen hyvyys on aina yhtä suuri tai suurempi kuin heuristinen arvio (9, 97). Esimeriksi reittihaun yhteydessä hyvä heuristiikka arvioi reitin pituuden optimisesti aina lyhyemmäksi kuin se todellisesti on. Heuristiikkafunktion yhteneväisyys (engl. consistent) on myös tärkeä ominaisuus heuristiikassa. Se otetaan huomioon usein optimointi- ja reittihakuongelmissa. Yhteneväinen heuristiikka lähestyy ratkaisua inkrementaalisesti, eikä liiku koskaan taaksepäin (9, 99). Esimerkiksi reittihaun yhteydessä yhteneväistä heuristiikka käyttävä algoritmi ei koskaan etene kohdepistettä vastakkaiseen suuntaan.

5 TEKNOLOGIAT

Tämä opinnäytetyö on web-sovellus, jota voidaan käyttää WWW-selaimella. Teknisen puolen toteuttamiseen käytettiin Java-pohjaista Liferay-palvelinohjelmistoa. Käyttöliittymä toteutettiin Vaadin-kirjaston ja Google Web Toolkitin avulla.

Matalalla tasolla http on pyyntöpohjainen siirtoprotokolla, jossa TCP-pyynnöillä haetaan palvelimelta dataa. Palvelin vastaa pyyntöön lähettämällä usein HTML-muodossa olevan dokumentin tai binääritietoa. Vastaanottaja (asiakas, tässä: selain) käsittelee vastauksen ja muodostaa siitä konkreettisen esityksen. Esimerkiksi WWW-selain muuntaa HTML-muodossa olevan dokumentin graafiseen näytöllä esitettävään muo-

toon. Pitkälle tähän päivään asti WWW-sivut ovat perinteisesti olleet staattisia esityksiä, joiden esittämiseen http sopii hyvin. Tarve WWW:n interaktiivisuudelle on kuitenkin aina ollut läsnä, koska sivua ei voitu hakemisen jälkeen juuri muuttaa. Tähän on kehitetty useita palvelimenpuolen tekniikoita, joista ensimmäisiä oli **CGI**-rajapinta. CGI mahdollistaa vastauksien prosessoimisen palvelimella ennen vastauksen muodostusta. Näin oli mahdollista reagoida käyttäjän syötteeseen esimerkiksi HTML-lomakkeiden POST-pyyntöjen mukaan ja muodostaa vastaus (HTML-dokumentti) syötteestä riippuen. Myöhemmin kehitettiin CGI:n rinnalle muun muassa **Java Servletit**, joilla dynaamisia WWW-sivuja pystyi luomaan alustariippumattomasti.

WWW-sivun muodostus on kaksiosainen sopimus selaimen (asiakas) ja palvelimen välillä. Selain on ollut merkittävässä osassa interaktiivisen WWW:n kehityksessä. Perinteisesti palvelin muodostaa vastauksen, lähettää sen ja lopuksi yhteys suljetaan. Jo olemassa olevan vastauksen muuttaminen on siis hankalaa, koska palvelinyhteys on suljettu ja uuden datan vastaanottamiseen tarvittaisiin uusi pyyntö. Tähän kehitettiin **Ajax**-tekniikka, jossa selaimessa toimivalla skriptikielellä on mahdollista tehdä pyyntöjä taustalla ilman, että varsinainen sivu muodostettaisiin uudelleen. Tämä mahdollistaa jo olemassa olevan vastauksen muuttamisen ja täydentämisen täysin dynaamisesti. Nykypäivän WWW:ssä on paljon interaktiivisuutta, jossa palvelinyhteys on avoinna lähes koko ajan. WWW:n käyttötavat ovat muuttuneet staattisista dokumenteista hyvin rikkaisiin käyttöliittymiin ja interaktiiviseen sisältöön. Uusi HTML5-standardi on kehitetty vastaamaan muuttuneita tarpeita. Se määrittelee esitysmuodot muun muassa äänelle, videolle ja blog-kirjoituksille.

5.1 Java Servlet

Servletit ovat palvelimessa toimivia Java-luokkia, joilla laajennetaan dynaamisesti palvelimen toiminnallisuutta. Servletit tarvitsevat palvelimelta Java-tuen, jotta niiden ohjelmakoodi voidaan suorittaa. Servleteille on määritelty Servlet API, jota noudattamalla palvelin kutsuu servletejä. Niiden toiminnallisuus ei ole rajoittunut vain http:hen: niitä voidaan käyttää minkä tahansa palvelinohjelmiston tiedon prosessointiin. (10.)

Java Servletien ajaminen palvelinpäässä eroaa hieman CGI:n tavasta. Servletit ajetaan samassa prosessiavaruudessa ja ne ladataan palvelimen käynnistymisen yhteydessä. CGI joutuu luomaan jokaista pyyntöä kohden uuden prosessin, joka käsittelee pyyn-

nön. Uuden prosessin luomien on raskasta ja hidastaa yksittäisen pyynnön käsittelyä etenkin, jos pyyntöjä tulee samalla hetkellä useita. Tämä tuo myös muitakin rajoitteita kuin suorituskyvyn. Esimerkiksi servletit voivat jakaa saman tietokantayhteyden useamman pyynnön kanssa, joka on CGI:llä mahdotonta johtuen käyttöjärjestelmän prosessirajoista. Servletit noudattavat Javan siirrettävyyden periaatetta, jossa samaa koodia voidaan ajaa ilman muutoksia ympäristöstä riippumatta. Lisäksi digitaalisesti allekirjoittamattomat servletit käsitellään epäluotettavina ja niitä ajetaan erityisessä hiekkalaatikkotilassa, jossa pääsy ohjelman ulkopuolisiin resursseihin on evätty. Se parantaa tietoturvaa, koska servletien alkuperä voidaan todentaa, eivätkä todentamattomat servletit pääse käsiksi kriittisiin resursseihin. CGI-tekniikka ei ota lainkaan kantaa suoritettavien ohjelmien alkuperään, eikä niiden turvallisuuteen (10).

API on varsin yksinkertainen. Oman servletin toteuttamiseen tarvitsee toteuttaa abstraktin *GenericServlet*-luokan metodit. Toinen vaihtoehto on toteuttaa jokin erikoistunut servlet-luokka, esimerkiksi *HttpServlet*. Ero näiden kahden välillä on, että *GenericServlet* tarkoittaa mitä tahansa pyyntö-vastaus-palvelinsovellusta, kun taas *HttpServlet* on erikoistunut http:hen. Listaus 1 esittelee yksinkertaisen esimerkkitoteutuksen *GenericServlet*istä.

```
public class OmaServlet extends GenericServlet {
  public void service(ServletRequest req,
                      ServletResponse resp)
  {
    // Servletin koko toiminnallisuus.
  }
}
```

Listaus 1: Yksinkertainen *GenericServlet*. Servletin pitää toteuttaa ainakin *GenericServlet*in määrittelemä abstrakti *service*-metodi. Se sisältää servletin koko toiminnallisuuden, jossa pyyntöihin vastataan.

HttpServlet on peritty *GenericServelt*istä ja se toteuttaa jo sen. Sen sijaan *HttpServlet*issä on muita metodeja, joita voidaan ylikirjoittaa pyynnön tyypistä riippuen. Listaus 2 esittelee yksinkertaisen *HttpServlet*-toteutuksen, joka käsittelee http:n GET- ja POST-pyyntöjä.

```
public class OmaHttpServlet extends HttpServlet {
  @Override
   protected void doGet(HttpServletRequest req,
                        HttpServletResponse resp)
   {
     // Servletin toiminnallisuus käsittelemään
     // http:n GET-pyyntöjä.
   }

  @Override
   protected void doPost(HttpServletRequest req,
                         HttpServletResponse resp)
   {
     // Servletin toiminnallisuus käsittelemään
     // http:n POST-pyyntöjä.
   }
}
```

Listaus 2: Yksinkertainen *HttpServlet*. Servletin ei tarvitse välttämättä toteuttaa yhtäkään metodia, koska abstrakti *service*-metodi on toteutettu jo *HttpServlet*in toimesta. Tässä on ylikirjoitettu *doGet*- ja *doPost*-metodit, joita kutsutaan kun tietyntyyppinen pyyntö vastaanotetaan.

5.2 Portaali

Yritysmaailmaan suunnatut web-palvelut tarvitsevat vankan alustan sisäisten ja ulkoisten resurssien jakamiseen, hallintaan ja ylläpitoon. Erilaisia resursseja ovat muun muassa sähköposti, wiki, foorumi ja intranet. Yksi tärkeä osa portaaleissa on pääsynhallinta ja eri resurssien rajaaminen vain tietylle joukolle. Web-palveluiden yleistyessä 1990-luvulla ja yritysten tarpeiden kasvaessa vakiintui portaali-termi kuvaamaan tällaisia palveluja. Portaalit tarjoavat yritykselle standardisoidun kanavan julkaista erilaista tietoa sisäisesti, asiakkaille ja yhteistyökumppaneille. Portaalit eivät kuitenkaan rajoitu vain yritysmaailmaan. Portaalit voidaan lajitella niiden tarjoamien palveluiden ja ominaisuuksien mukaan. Kategorioinnille ei ole olemassa standardia, joten on olemassa laaja joukko erilaisia ryhmittelyjä. (11.)

Loppukäyttäjän näkökulmasta portaalit eivät eroa muista web-sivuista lainkaan. Portaaleiden peruslähtökohta on organisoida, esittää ja hallita tietoa liiketoiminnan näkökulmasta. Teknologista määritelmää portaaleille ei ole. Ainoastaan vain ohjeistuksia siitä, mitä portaalin tulisi tarjota loppukäyttäjille. Se antaa siis täyden vapauden toteuttaa portaalin alustasta ja ohjelmointikielistä riippumatta. On kuitenkin muodostunut *de facto* siitä, miten portaalit usein toteutetaan. Hyvät tekniset valmiudet niiden toteuttamiseen tarjoaa Java-ohjelmointikieli. Java sisältää oman teknisen standardin portaaleille, jota noudattamalla voidaan taata hyvä siirrettävyys ja pitkä elinkaari. JSR 168 -

standardi määrittelee portaalin portletien ominaisuudet, jotka mahdollistavat koko portaalin siirtämisen toiseen standardia tukevaan ympäristöön. Javan tapauksessa portletit toteutetaan palvelimella usein servletien avulla. Servletit vastaanottavat http-pyynnöt ja muodostavat vastauksen niihin käyttämällä portaalisovelluksen tarjoamia tietoja. Portletit eivät suoraan pysty vaikuttamaan matalantasoisiin http-vastauksiin tai käsittelemään pyyntöjä: kaikki on abstraktoitu palvelintekniikan ja portaalisovelluksen välille. (11.)

5.3 Liferay

Liferay on vapaan lähdekoodin portaalien kehitys- ja hallintaohjelmisto. Se on toteutettu Java Servlet -rajapinnan päälle, jonka lisäksi se sisältää lukuisia valmiita portletteja ja ohjelmointia helpottavia työkaluja. Liferay on mahdollista ladata paketoituna http-palvelimen kanssa, eikä mitään erillisiä asennuksia tarvita. Tuettuja palvelinalustoja ovat muun muassa Tomcat, Geronimo, Glassfish ja JBoss. Liferaysta on saatavilla vapaan lähdekoodin versio ja yrityskäyttöön suunnattu maksullinen *enterprise*-versio.

Liferayllä rakennetut web-sivut koostuvat portleteista, jotka ovat täysin itsenäisiä sivun osia. Portaalin ylläpitäjä voi asentaa, lisätä ja poistaa portletteja ilman palvelinohjelmiston uudelleenkäynnistystä tai ohjelmointikokemusta. Liferayn portleit noudattavat uutta JSR 286 -portlettistandardia. Lisäksi Liferay-alustalla on mahdollista ajaa Java Servletejä, jotka voivat käyttää Liferayn tarjoamia palveluja. Esimerkiksi on mahdollista käyttää käyttäjäntodennusta, tietokantaa ja kolmannen osapuolen palvelimelle asennettuja palveluita.

Liferayssä on vahvat pääsyn- ja käyttäjänhallintatyökalut, joilla portaalin palveluiden saatavuutta on mahdollista mukauttaa monipuolisesti. Kaikki portaalin rekisteröityneet käyttäjät kuuluvat *organisaatioon*, joka muodostaa hierarkkisen rakenteen. Liferayn versiosta 6.1 lähtien on ollut mahdollista hallita kahta tai useampaa portaalia samalla palvelimella täysin toisistaan erillään (12). Sen toteuttamiseksi vaaditaan kuitenkin, että palvelun entiteetit (katso 5.4) ottavat tämän tietojen omistajuudessa huomioon.

Kehitystyö Liferayllä tapahtuu jatkuvana integraationa, jossa muutokset päivitetään automaattisesti palvelimelle. SDK tukee Apache Ant -sovellusta, joka suorittaa käännöstyön ja kopioi tarvittavat tiedostot palvelimelle. Apache Ant tekee erilaisia kääntämiseen liittyviä tehtäviä, jotka määritellään *build.xml*-tiedostossa. Se esimerkiksi

kopioi tiedostot oikeaan paikkaan ja kääntää tarvittavat lähdekoodit. Liferayn mukana toimitetaan valmis *build.xml*-tiedosto. Tiedosto sisältää automatisoidut toimenpiteet, jotka on suoritettava, jotta suunniteltu Liferay-kokoonpano toteutuisi. Esimerkiksi säännöstö *deploy* suorittaa toimet, jolla koko portletti siirretään ja rekisteröidään palvelimelle.

5.4 Liferayn palvelurakenne ja -rakentaja

Liferayssä on automaattinen palveluiden rakentaja (engl. Service Builder), jolla voidaan luoda sisäisiä tai ulkoisia palveluita (portaaleiden resursseja). Rakentaja luo tarvittavat Java-luokat XML-tiedoston pohjalta, jotka abstraktoivat yleisimmät palvelutoimet, kuten tiedon muokkaaminen, poistaminen ja hakeminen. Se generoi palveluun myös standardisoituneet rajapinnat siirrettävyyden parantamiseksi, kuten Bean ja SOAP.

Tietoyksiköt (engl. data models) ovat entiteettejä, jotka ovat esityksiä tietokantaan. Liferayn palveluarkkitehtuuri käsittelee kaikkia tietokantaoperaatioita entiteetteinä. Entiteetit sisältävät muun muassa tietokannan sarakemääritykset, lajittelusäännöt ja viittaukset muihin entiteetteihin. Jokainen entiteetti muodostaa tietokuvauksen palveluun, jonka ympärille koko palvelu rakentuu. MVC-mallin näkökulmasta entiteetit ovat näkymiä tietokannassa oleviin malleihin. Palvelurakentaja tekee entiteettien pohjalta tarvittavat Java-luokat ja tietokannan taulut. Pääsy entiteetteihin on kaksitasoinen: joko paikallinen tai globaali. Paikallisia palveluita on mahdollista käyttää vain saman Java-virtuaalikoneen sisältä, kun taas globaalit palvelut ovat saatavissa kaikkialta Internetin välityksellä. Globaaleihin palveluihin generoidaan myös erinäisiä tietoturvaominaisuuksia, joita ei paikallisissa palveluissa tarvita. Listaus 3 esittelee esimerkin *service.xml*-tiedostosta, jonka perusteella generoitaisiin yksinkertainen palvelu.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE service-builder PUBLIC "-//Liferay//DTD Service Builder
6.0.0//EN" "http://www.liferay.com/dtd/liferay-service-builder_6_0_0.dtd">
<service-builder package-path="fi.kyamk.neli">
  <author>Jussi Raunio</author>
  <namespace>NeliApp</namespace>
  <entity name="RoutePoint" local-service="true" remote-service="false">
    <column name="pointId" type="long" primary="true"></column>
    <column name="roadId" type="long"></column>
    <column name="x" type="double" primary="false"></column>
    <column name="y" type="double" primary="false"></column>
    <order by="asc">
      <order-column name="pointId"></order-column>
    </order>
    <finder name="Coordinate" return-type="Collection">
      <finder-column name="x" comparator="=" arrayable-
operator="AND"></finder-column>
      <finder-column name="y" comparator="=" arrayable-
operator="AND"></finder-column>
    </finder>
  </entity>
</service-builder>
```

Listaus 3: Yksinkertainen *service.xml*-tiedosto. Tiedostossa määritellään yksi entiteetti (*RoutePoint*), jolla on kentät *pointId*, *roadId*, *x* ja *y*. Lisäksi määritellään oletusjärjestys, jota käytetään entiteettien järjestämiseen. Lopuksi määritellään hakija, jolla entiteettejä voidaan hakea X- ja Y-koordinaattien perusteella.

Palvelurakentaja luo paljon luokkia *service.xml*-tiedoston pohjalta. Suurinta osaa näistä luokista ei saa muokata manuaalisesti, sillä ne generoidaan aina uudestaan kun palvelurakentaja ajetaan. Osaan luokista voidaan lisätä tarvittavaa lisälogiikkaa palvelun käyttämiseen. Palveluita käyttämällä ei tarvitse huolehtia tietokantojen eroavaisuuksista eri alustoilla, sillä Liferayn palvelunrakentaja hoitaa yhteensopivuusasiat. Luonnollisesti palvelunrakentaja ei voi ottaa huomioon kaikkia tietokannan pääsytarpeita, joten tietokannassa on mahdollista ajaa myös lähes suoraan kyselyjä. Liferayn versiosta 5.1 lähtien kyselyt ovat abstraktoitu *DynamicQuery*-luokan taakse, joten myös mukautetut kyselyt ovat yhteensopiva eri tietokantojen välillä (13). Alla olevassa listauksessa on esitetty esimerkki mukautettujen kyselyjen luomisesta.

```
DynamicQuery query = DynamicQueryFactoryUtil.forClass(RoutePoint.class)
  .add(RestrictionsFactoryUtil.and(
    PropertyFactoryUtil.forName("x").ge(new Double(x - range)),
    PropertyFactoryUtil.forName("y").ge(new Double(y - range))

  ))
  .add(RestrictionsFactoryUtil.and(
    PropertyFactoryUtil.forName("x").le(new Double(x + range)),
    PropertyFactoryUtil.forName("y").le(new Double(y + range))
  ));
```

Listaus 4: Mukautettu kysely, joka hakee *RoutePoint*-entiteettejä (ks. Listaus 3) tietyltä alueelta. Varsinainen kysely voidaan suorittaa esimerkiksi *RoutePointLocalServiceUtil*-luokassa sijaitsevan *dynamicQuery*-metodin avulla.

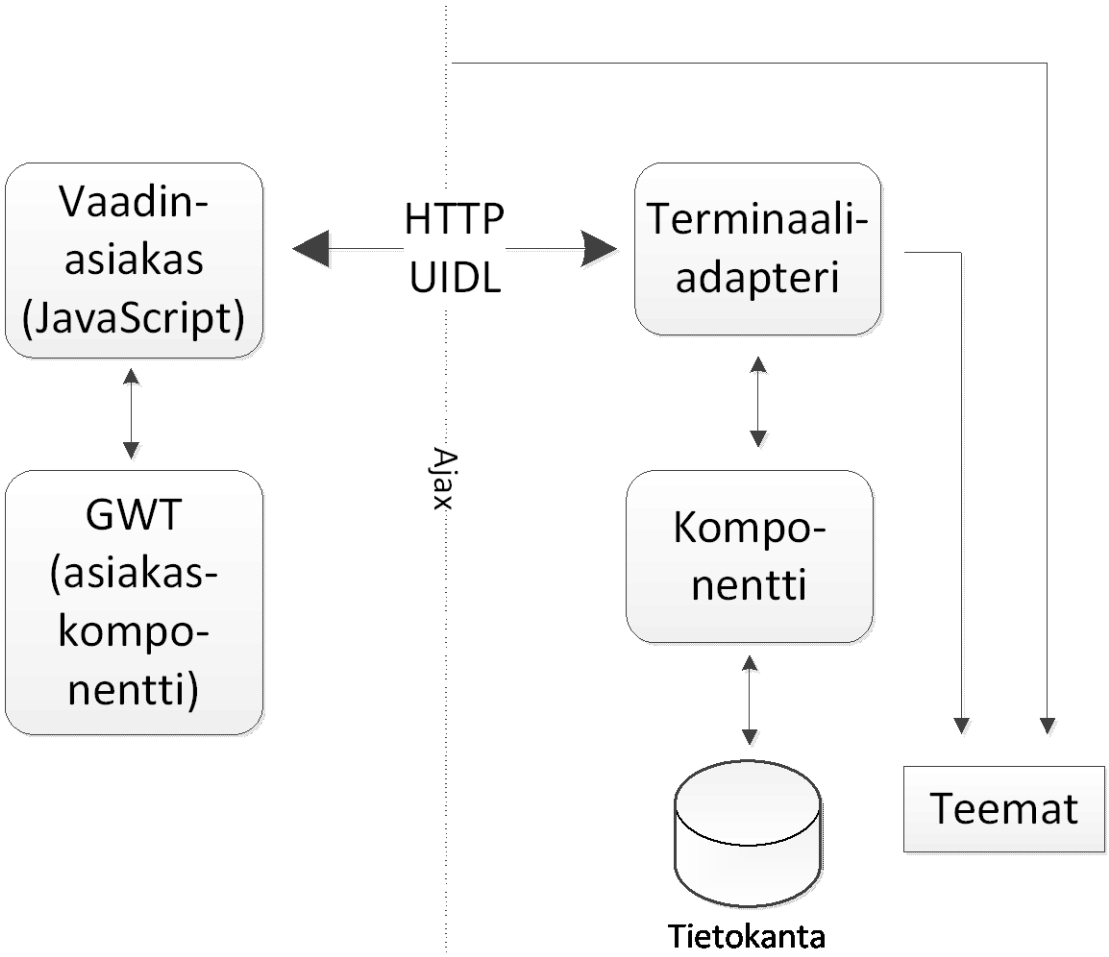
5.5 Vaadin

Vaadin on Java-pohjainen sovelluskehityskirjasto, joka on tarkoitettu ensisijaisesti web-sovellusten kehitykseen. Vaadinin voidaan katsoa keskittyvän pääsääntöisesti asiakaspuolen käyttöliittymän suunnitteluun ja rakentamiseen. Siten se muistuttaa hieman samaan tarkoitukseen kehitettyjä JavaScript-pohjaisia kirjastoja, kuten jQuery ja Alloy UI. Osittain nämä kilpailevatkin Vaadinin kanssa vaikka teknisesti lähestyvät ongelmaa erilaisesta näkökulmasta. Vaadinin erikoisuus on sen palvelinpohjainen suunnittelu, jossa pääosa logiikasta sijaitsee palvelimella. Google Web Toolkit -kirjaston avulla palvelinpuolen komponenteille tehdään asiakaspuolen vastakappaleet. Keskusteluun asiakkaan ja palvelimen välillä Vaadin käyttää UIDL-teknikkaa (engl. User Interface Definition Language), johon data serialisoidaan palvelimen päässä ja puretaan asiakkaan puolella. Asiakaspuolen komponentti parsii UIDL:n ja suorittaa niin kutsutut piirtorutiinit. Web-sivun ruudulle piirtämisen hoitaa varsinaisesti selain, mutta Vaadinissa käytetään piirto-termiä, jotta terminologia muiden työpöytäkirjastojen kanssa säilyy. Jokaisella komponentilla on omat piirtorutiinit, joten kun jokin komponentti piirretään uudelleen, muita komponentteja ei tarvitse piirtää uudelleen. Version 5 jälkeen UIDL:ssä on käytetty serialisointiin JSONia. (14, 1-3; 411.)

Vaadin muistuttaa paljolti muita Javan käyttöliittymän ohjelmointiin tarkoitettuja kirjastoja, kuten Swing ja AWT. Vaadinin yksi suunnittelupäämääristä onkin saada aikaan vakaa työpöytäsovelluksia muistuttava pohja web-sovellusten käyttöliittymän ohjelmointiin ja suunnitteluun (14, 1). Vaadinilla sovelluksia ohjelmoidaan käyttäen Javaa täysin ilman HTML-sommittelua, CSS-tyylimäärittelyjä tai JavaScript-ohjelmointia. Kirjasto abstraktoi kaiken web-logiikan rajapintojen taakse, joten web-sovelluksia voidaan rakentaa samaan tapaa kuin työpöytäsovelluksia. Loppukäyttäjälle Vaadinin palvelinpuoleisuus ei näy, koska lopullinen käyttöliittymä on HTML-

muodossa ja käyttöliittymän päivitys tapahtuu käyttäen Ajax-tekniikkaa. Vaadinin sivuilla mainitaan, että palvelimella täysin ajetut web-sovellukset lisäävät turvallisuutta, joka on ensiarvoisen tärkeä osa sovellusta yritystoiminnan näkökulmasta (15). Korkean abstraktoinnin ansiosta Vaadin voi ottaa käyttöön uusimpia HTML-standardien ominaisuuksia ilman, että kirjastoa käyttävään koodiin täytyisi tehdä muutoksia. Lisäksi Vaadin-sovellusten virheenetsintä on Java-virheenetsijöiden ansiosta huomattavasti helpompaa kuin tavanomaisten web-sovellusten.

Vaadin on ollut integroituna Liferayhin versiosta 6 lähtien (16). Tämän dokumentin kirjoittamishetkellä tuki ei ollut kuitenkaan vielä täydellinen. Vaadin-sovellusten käyttäminen onnistuu Liferay-portaaleissa ongelmitta, mutta koko portaalin pitää käyttää samoja asiakaspuolen komponentteja. Rajoitus johtuu siitä, että samalla sivulla ei voi olla useampaa Google Web Toolkit -komponenttikirjastoa (engl. widgetset) käytössä, mutta silti sivulla voi olla useampi portletti. Ongelmaksi se muodostuu silloin, jos halutaan käyttää Liferayn automaattisia asennustoimintoja, sillä aina kun uusi portletti asennettaan, on kaikki asiakaspuolen komponentit generoitava uudelleen, yhdistettävä ne yhdeksi ja asennettava palvelimelle globaalisti. Tähän tarkoitukseen on tosin kehitetty lisäosa, joka tekee sen automaattisesti. Ekbladin mukaan lisäosa integroidaan osaksi Liferayta tulevissa versioissa (16).



Kuva 4: Vaadinin toimintalogiikka. Terminaaliadapteri vastaa keskustelusta servletien ja portletien sekä asiakaspuolen kanssa. Palvelinpuolen komponentti keskustelee terminaaliadapterin kanssa tapahtumien avulla. Palvelinpuolen komponentti sisältää koko sovelluslogiikan. Vaadin-asiakas hoitaa terminaaliadapterin lähettämän UIDL:n parsimisen ja käyttöliittymän muodostamisen sen perusteella.

5.6 Google Web Toolkit

Suurin ongelma web-käyttöliittymien toteuttamisessa on WWW-selainten väliset toiminnalliset erot ja vaikeahko virheenetsintä. Osa selaimista saattaa tulkita standardia eri tavalla ja osa jopa virheellisesti. Eniten eroja on kuitenkin JavaScriptin toiminnassa, sillä selaimet käyttävät erilaisia metodeja muun muassa Ajax-pyyntöjen tekemiseen. Google Web Toolkit on Googlen kehittämä kokoelma työkaluja, jolla voidaan kehittää rikkaita ja monimutkaisia web-sovelluksia helposti. GWT ottaa huomioon selainten väliset eroavaisuudet ja abstraktoi toiminnat yhteneväisen rajapinnan taakse. Lisäksi GWT tuo mukanaan toisenlaisenkin edun: ohjelmointi tapahtuu Java-ohjelmointikielellä, joka käännetään optimoiduksi JavaScriptiksi. Siten saavutetaan osa käännettävien kielien eduista, eikä ohjelmoijan tarvitse keskittyä optimointiin JavaScript-kielellä.

Google Web Toolkit on integroitu Vaadiniin, missä sitä käytetään palvelin- ja asiakaspuolen yhdistämiseen.

6 SOVELLUKSEN TOIMINTA JA ARKKITEHTUURI

Satamaan tulee paljon ulkopuolista tavaraa. Tavaran siirtelyyn vaaditaan monien tahojen välistä tiedonvaihtoa sekä kuljetustavan valintaa ja reittihakua tavaran säilöön kuljettamiseksi. Tässä opinnäytetyössä tehdyssä sovelluksessa on keskitytty tavaran kuljettamiseen ja käsittelyyn liittyviin ongelmiin. Sovellus on osa informaatiokeskusta, jonka kautta tiedonvaihto toimijoiden välillä tapahtuu. Sillä pystytään hallitsemaan satamassa olevia kuljetuskalustoja ja satamanostureita, tarkastelemaan virtuaalista karttaa satama-alueesta sekä laskemaan reittejä sataman sisällä. Reittihakua käytetään pääsääntöisesti reitin etsimiseen sataman sisällä siirrettäville kappaleille. Reittihaussa huomioidaan kaluston ominaisuudet ja tiestön rajoitukset.

Pääpiirteissään sovelluksen toiminta-alue voidaan jakaa seuraaviin osiin: puhdas tietovarasto, varastoidun tiedon hyväksikäyttäminen reittihaussa ja karttanäkymän esittäminen. Koska sovellus integroidaan osaksi sataman informaatiokeskusta, on otettava huomioon mahdolliset eri toimijoiden tarpeet. Esimerkiksi kuljetuskalustojen lisääminen voidaan rajata jollekin tietylle käyttäjäryhmälle ja sallia muille vain järjestelmässä olevien tietojen tarkastelu. Kuljetuskaluston tarkkaa määritelmää vältetään, koska ei tiedetä konkreettisia siirrettäviä kappaleita, josta seuraa se, ettei varsinaista laitetta tai konetta välttämättä voida määrittää. Sen sijaan sovelluksessa käytetään oleellisia tietoja kuljetuskalustojen ominaisuuksista, joita eri kalustot toteuttavat. Näitä ovat esimerkiksi suurin lastauskuorma, pituus, leveys ja massa. Satamanostureita käytetään ilmaisemaan käyttäjälle, missä tavaraa on mahdollista lastata ja jatkokäsitellä.

Tässä luvussa esitellään sovelluksen taustalla oleva teoria ja perustellaan sovelluksen toteutuksessa tehdyt tekniset ratkaisut. Luku ei sisällä sovelluksen ominaisuuksien läpikäyntiä tai toimintojen kuvausta käyttäjän näkökulmasta. Liitteenä olevassa vaatimusmäärittelyssä (Liite 1) kuvataan sovelluksen toiminta käyttäjän näkökulmasta ja esitellään käyttöliittymät. Vaatimusmäärittely on kirjoitettu niin täsmällisesti, että sitä voidaan käyttää myös sovelluksen pienenä käyttöohjeena.

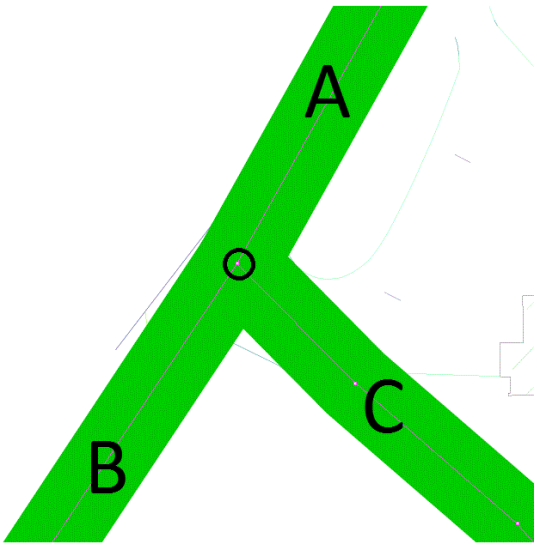
6.1 Virtuaalinen kartta

Virtuaalinen karttanäkymä satama-alueesta toteutettiin jakamalla kartta pieniin lohkoihin (engl. tile), jotka esitetään web-sivulla vierekkäin aseteltuna. Lohkoihin jakaminen mahdollistaa muun muassa eri osien käsittelemisen kartasta samanaikaisesti, näkymättömien alueiden piilottamisen ja joustavan välimuistittamisen selaimen toimesta. Lohkot ovat bittikarttakuvaformaatissa, joten selain ei tiedä, missä muodossa varsinainen pohjakartta on. Jokainen lohko generoidaan ajonaikaisesti isosta karttakuvasta. Lähdekartan resoluution täytyy tällöin olla riittävän korkea, että kartan koon muuttaminen ja joustava käsittely on mahdollista. HaminaKotka satamalla on käytössä MicroStation CAD-piirustusohjelmisto, jolla saatu karttakuva satamasta oli tehty. Ohjelmisto tukee viemistä AutoCAD-yhteensopiviin formaatteihin, kuten DWG ja DXF. Kartta vietiin DXF-muotoon, josta se vietiin Adobe Illustrator -ohjelmistolla edelleen SVG-muotoon. Molemmat muodot ovat vektorigrafiikkapohjaisia, mutta SVG:n hallinta on helpompaa, koska se ei ole suljettu formaatti niin kuin DXF. Tämän johdosta hyvin monet ohjelmistot tukevat sitä yhteneväisesti. Tämän jälkeen karttakuva vielä pakattiin gzip-pakkauksella tilan säästämiseksi. Noin neljän megatavun karttakuva pieneni 700 kilotavuun pakkauksen ansiosta.

Alkuperäisessä kartassa tiet eivät olleet yhteneväisiä, joten jokainen tie jouduttiin rakentamaan uudelleen Adobe Illustratorilla. Reittihaun kannalta tiestön pitää olla yhteneväinen. Tämä tarkoittaa sitä, että jokaisesta liitäntäkohdasta (risteyksestä) pitää lähteä uusi tie, joka kytkeytyy edelliseen tiehen. Vain siten voidaan reiteistä muodostaa graafi, jonka avulla reittihaku on mahdollista suorittaa. Lisäksi teiden pisteet pitää olla ryhmiteltynä, jotta teitä voidaan ohjelmallisesti käsitellä. Samalla tiet päätettiin erottaa varsinaisesta karttapohjasta ja tallentaa ne erillisenä tietokantaan. Kaikki tiet liitetään ladattuun karttapohjaan ajonaikaisesti. Etuna tässä on, että työlästä karttapohjan viemisprosessia ei tarvitse toistaa, vaikka tiet muuttuisivat. Lisäksi teiden muuttaminen ja hallinta yksinkertaistuu huomattavasti.

Jokaisesta tiestä tehtiin Adobe Illustratorilla avoin polku erilliselle tasolle karttakuvan päälle. Polut yhdistettiin risteyskohdista toisiinsa. Tämä eroaa hieman arkipäivän ajattelutavasta, jossa on jokin päätie ja siihen liittyy sivuteitä. Tässä vaiheessa kaikki tiet voidaan ajatella samanarvoiseksi. Varsinainen teiden sopivuus ja suosinta reiteille arvioidaan vasta reittihaun yhteydessä. Alla olevassa kuvassa (Kuva 5) voitaisiin ajatel-

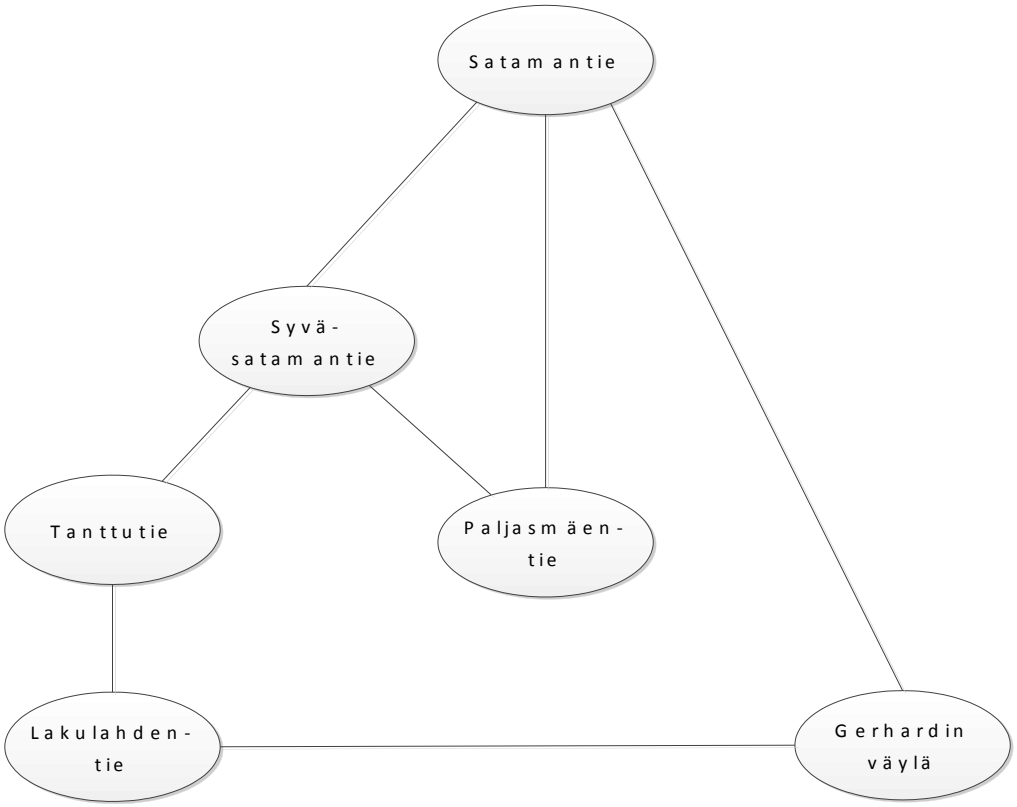
la, että tie A on päätie, mutta C-tien kytkeytymisen takia myös A-tie haarautuu uudeksi tieksi (B). Lopuksi jokaisen reitin piste vietiin tietokantaan. Ajonaikaisesti tietokannasta haetaan pisteet, muodostetaan niistä *polyline*-elementit ja yhdistetään näin saadut tiet karttapohjaan. SVG:n XML-muotoisuuden ansiosta karttapohjan muokkaaminen ajonaikaisesti käy vaivattomasti. Reittihaun kannalta tällä ei ole merkitystä, koska varsinainen reittihaku on täysin riippumaton karttapohjasta ja sen esityksestä. Näin kartan visuaalisen esityksen yhteneväisyys säilyy, koska generoiduissa karttakuvissa on kaikki sataman tiet juuri siten kuin reittihaku käsittelee niitä. Prosessi on lisäksi varsin nopea, joten se voidaan suorittaa aina kartan lataamisen yhteydessä.



Kuva 5: Tiet A, B ja C ovat yhdistetty, joten niiden kautta on mahdollista kulkea. Tarkasteltuna A-tien näkökulmasta, sen päätepiste on teiden B ja C alkupiste (yhdyskohta ympyröitynä).

6.2 Reittihaku

Virtuaalista karttaa käytetään esittämään reittihaun tulokset visuaalisesti. Varsinainen reittihaku tapahtuu kuitenkin tietokantaan tallennettujen pisteiden perusteella. Jokainen tie koostuu pisteistä ja pistepareista muodostuu kaaria. Tiet ja niiden väliset riippuvuudet voidaan siten esittää graafin avulla. Kuva 6 esittää yksinkertaistettua graafikuvausta muutamasta sataman tiestä.



Kuva 6: Joitain HaminaKotka sataman teiden välisiä suhteita Satamantien näkökulmasta.

Kaikki tiet ovat tietokannassa pisteinä. Tietyt pisteet on ryhmitelty, joista muodostuu varsinaiset tiet. Teillä on lisäksi tietoja muun muassa maksimikuormasta, leveydestä ja suurimmasta alikulkukorkeudesta. Jokaisella pisteellä on automaattisesti kasvava yksilöllinen numerotunnus, jonka avulla samassa koordinaatissa sijaitsevat pisteet on mahdollista erottaa. Sen lisäksi numeeristen tunnusten avulla on mahdollista käsitellä tien pisteitä vieruslistamaisesti, sillä tien ensimmäisellä pisteellä on pienin tunnus ja viimeisellä suurin. Muiden pisteiden tunnukset tällä välillä suurenevat järjestyksessä kohti viimeistä pistettä. Tien pisteet yhdistetään viivoin, joten suurempi pistemäärä tarkoittaa monimutkaisempaa (enemmän mutkia) tietä. Tosin tämä ei päde 90 asteen käännöksissä. Erikoistapauksena joudutaan käsittelemään risteyskohdat, jossa tieltä on mahdollista haarautua toiselle tielle. Risteyskohtien esiintyminen voidaan rajata vain teiden alku- ja päätepisteisiin, jos jokainen tie on muodostettu siten, että kaikista mahdollisista risteyskohdista alkaa aina uusi tie. Risteyskohdissa täytyy tarkistaa koordinaattien perusteella, onko pisteestä mahdollista siirtyä johonkin toiseen pisteeseen eli onko piste risteyskohdassa.

6.3 Algoritmit ja reittien hyvyys

Perusajatus reittihaussa on etsiä *lyhin* polku graafin läpi solmujen *A* ja *B* välille. Tämä tarkoittaa sitä, että painotetussa graafissa polku on lyhin, kun sillä on pienin kokonaispaino. Arkipäivän ongelmissa graafin painot eivät usein ole pelkästään maantieteellisiä etäisyyksiä, joten lyhin polku graafin läpi ei tarkoita samaa kuin maantieteellisesti lyhin polku. Tästä syystä tällaista lyhintä polkua graafin läpi kutsutaan tässä kappaleessa *parhaaksi* poluksi. Tavanomaisilla graafin läpikäyntimetodeilla, kuten syvyyshaku (engl. Depth-first Search, DFS) ja leveyshaku (engl. Breadth-first Search, BFS), voidaan selvittää lyhin polku kahden solmun välille (9, 73). Nämä algoritmit eivät kuitenkaan sovellu isoihin hakuympäristöihin, jossa läpikäytävä hakualueen kompleksisuus on eksponentiaalinen (9, 73). Vuonna 1959 julkaistu Dijkstran algoritmin kehitti arvostettu tietojenkäsittelyteoreetikko Edsger Dijkstra. Algoritmi etsii parhaan polun kahden solmun välille painotetussa graafissa, joka sisältää vain positiivisia painoja. Se ei kuitenkaan käytä lainkaan heuristiikkaa, joten haku suoritetaan täysin lokaalisti. Tästä seuraa se, että algoritmi hidastuu isolla tietomäärällä, koska läpikäytäviä solmuja on paljon, eikä algoritmi pysty mitenkään arvioimaan, missä päätepiste sijaitsee. Dijkstran algoritmin pohjalta on kehitetty A* hakualgoritmi, jossa on heuristiikka mukana. Heuristiikan ansiosta algoritmin läpikäytävien solmujen määrä pienenee. Ihanteellisesti algoritmi löytää päätepisteen etenemällä sitä kohti jokaisella iteraatiolla eli se on inkrementaalinen. Lisäksi sen käyttäytymistä voidaan muuttaa riippuen tilanteesta, jopa kesken hakuprosessin. Tässä työssä toteutetussa sovelluksen reittihaussa on käytetty A* hakualgoritmia reittien etsimiseen. Seuraavassa käydään läpi algoritmin ominaisuuksia ja niiden hyödyntämistä reittihaussa.

A* on täydellinen, joten se löytää aina ratkaisun, jos sellainen on olemassa. Heuristiikalla on suuri merkitys polun hyvyyden määrittämisessä. Heuristiikan avulla on mahdollista parantaa algoritmin toimintaa ja löytää optimaalisia ratkaisuja. Heuristiikkafunktion määrittelyssä pitää kuitenkin ottaa huomioon muutama seikka, jotta algoritmin todellinen hyöty saadaan käyttöön ja sen tuottamat ratkaisut ovat aina optimaalisia. Ratkaisujen hyvyyttä arvioidaan kustannuspohjaisesti, jolloin algoritmi etsii aina pienimmän kustannuksen omaavia siirtymiä ja polkuja. Kustannusfunktiota merkitään $f(x)$:llä, jossa x on solmu. Funktio määritellään $g(x)$:n ja heuristiikkafunktion, $h(x)$, summana. Funktio $g(x)$ on polkukustannus solmuun x. Se sisältää siis kumuloituneen siirtymäkustannuksen alkusolmusta johonkin muuhun solmuun, joten alkusolmulle

$g(x) = 0$. Heuristiikkafunktio mittaa aina hyvyyttä käsiteltävän solmun ja päätesolmun välillä. Sitä voitaisiin luonnehtia polun loppupään etäisyysarvioksi, koska optimaalinen A* ei koskaan etene taaksepäin. Jotta algoritmi olisi optimaalinen, täytyy heuristiikkafunktion olla hyvä ja yhteneväinen eli $h(x) \leq d(x, y) + h(y)$, missä $d(x, y)$ on solmujen x ja y välinen siirtymä. Muussa tapauksessa A* saattaa löytää polkuja, jotka eivät ole parhaita. Toisaalta arvioimalla polkujen pituudet yläkanttiin, algoritmi löytää keskimääräisesti nopeammin ratkaisun kuin parhaimman polun. Optimaalisen polun etsiminen voi todella isoissa graafeissa olla hidasta, joten heuristiikkafunktion saattaa olla hyvä ottaa riskiarviointeja näissä tapauksissa. A* on lisäksi täysin yhteensopiva Dijkstran algoritmin kanssa. Tapaus $h(x) = 0$ kaikille x:lle on erikoistapaus, jossa heuristiikka ei ole käytössä, jolloin algoritmi palautuu Dijkstran algoritmiksi. (9, 97-101.)

Lähtötiedoksi A* tarvitsee alkupisteen ja päätepisteen. Algoritmi pitää yllä listaa solmuista, jotka täytyy tutkia seuraavalla iteraatiolla. Oikeastaan lista on minimiprioriteettijono, joka on järjestetty $f(x)$:n arvojen mukaan. Algoritmissa on mukana vielä suljettu lista, joka sisältää jo käsitellyt solmut, joita ei tarvitse siis käsitellä enää toistamiseen. Alla olevassa listauksessa (Listaus 5) on esitelty algoritmin pseudokoodimuotoinen toteutus.

```
function A*(start, goal)
  closedSet = []
  openSet = [start]

  start.G = 0.0
  start.H = h(start, goal)

  while (!openSet.empty())
    current = openSet.findLowestF()

    if (current == goal)
      return reconstructPath(current)

    openSet.remove(current)
    closedSet.push(current)

    foreach (node : current.adjacents())
      if (closedSet.contains(node))
        continue

      gScore = current.G + d(current, node)

      if (!openSet.contains(node))
        openSet.push(node)
        node.H = h(node, goal)

      else if (gScore >= node.G)
        continue

      node.Parent = current
      node.G = gScore

  return []

function reconstructPath(node)
  path = []

  while (node != null)
    path.unshift(node)
    node = node.Parent

   return path
```

Listaus 5: A* hakualgoritmin pseudokoodimuotoinen toteutus. Toteutus ylläpitää jokaiselle käsitellylle solmulle $g(x)$- ja $h(x)$-kustannusten lisäksi linkkiä isäntäsolmuun (*Parent*), jota seuraamalla saadaan paras polku käsiteltävään solmuun. Tästä seuraa se, että seuraamalla päätesolmun isäntälinkkejä, on mahdollista muodostaa paras polku alkupisteen ja päätepisteen välille. Jokaisen naapurisolmun kohdalla tarkistetaan, onko polku mahdollisesti parempi tähän solmuun kuin jo tunnettu ja vaihdetaan se tarvittaessa.

Satama-alueen teillä on seuraavanlaisia rajoitteita, jotka täytyy huomioida reittihaussa: suurin sallittu kuorma, leveys ja alikulkukorkeus. Vastaavasti kuljetuskalustoille ja kuljetettaville kappaleille on määritelty vastaavat arvot. Teiden arvot ovat maksimeja, eli vain kuljetuskalustot, joiden jokin ominaisuus ylittää jonkin tien rajoitteista, täytyy hylätä. Muut kuljetuskalustot voivat kulkea tietä pitkin riippumatta mitoista. Kuitenkin halutaan, että kaikki kuljetuskalustot käyttäisivät pääteitä. Pääteillä on yleensä parhaat ominaisuudet ja kestävyys. Voidaan olettaa, että suuremmat tien parametrit

tarkoittavat soveltuvampaa tietä. Näillä teillä on pienempi kustannus, joten algoritmi suosii niitä. Siirtymäkustannus teiden välillä lasketaan jokaiselle tielle neliöllisen virheen (engl. Root-mean-square deviation, RMS) perusteella seuraavasti:

$$d_{tie}(kuorma, leveys, alikulku) = \sqrt{\frac{\left(\frac{1}{kuorma} \times 120\right)^2 + \left(\frac{1}{leveys} \times 1200\right)^2 + \left(\frac{1}{alikulku} \times 380\right)^2}{3}}$$

Kustannus ottaa huomioon ihanteellisemmat arvot ja niissä halutaan minimoida poikkeamat. Periaatteena on, että tiestön siirtymäkustannus pienenee lähestyen nollaa, kun tiestön ominaisuudet ovat parempia (tässä: suurempia) kuin ihannearvot. Vastaavasti ihannearvojen alapuolella olevien tiestöjen kulkukustannus kasvaa. Haminan sataman liikennejärjestelyselvityksen perusteella on päätelty ihanteellisimmat arvot pääteiden ominaisuuksille, mutta arvoja voi myös muuttaa sovelluksessa ajonaikaisesti (17). Siirtymäkustannus kaikkien pisteiden välillä on niiden euklidinen etäisyys. Lopullinen siirtymäfunktio on siis seuraavanlainen:

$$d(x, y) = \begin{cases} d_{tie}(y_{kuorma}, y_{leveys}, y_{alikulku}) & kun\ risteys \\ \|y - x\| & muulloin \end{cases}$$

Heuristinen arvio on euklidinen etäisyys alku- ja päätepisteen välillä. Heuristiikka ei koskaan yliarvioi kustannuksia, koska tapauksessa, jossa alkupiste ja päätepiste sijaitsevat samalla tiellä, eikä niiden välissä ole yhtään muuta pistettä, heuristiikka antaa yhtä suuren arvon kuin todellinen kustannus. Kaikissa muissa tapauksissa heuristiikka arvioi kustannukset aina pienemmiksi kuin ne todellisuudessa ovat.

7 LOPPUPÄÄTELMÄ JA YHTEENVETO

Tietotekniikan hyödyntäminen logistiikan kuljetusongelmien ratkaisussa on kiehtovaa. Logistiikkaa esiintyy lähes joka alalla, ja usein sen ongelmat käsitetään niin arkipäiväisiksi, että tietotekniikan oletetaan ratkaisevan ne helposti ja nopeasti. Nykyinen tietojenkäsittely ei kuitenkaan pysty pureutumaan logistiikan ongelmiin sillä tasolla, jolla ihmiset ne pystyvät ratkaisemaan. Useaan ongelmaan voidaan etsiä tietokoneavusteisesti osittaisratkaisu, joka helpottaa lopullista, ihmisen tekemää, päätöstä. Ongelmien ratkaisussa täytyy hakeutua kohti ihmismäisempää ratkaisumallia, jossa ongelmien

ratkaisussa käytetään klassiselle tietojenkäsittelylle vielä vieraita menetelmiä, kuten satunnaisuutta ja emergenssiä.

Opinnäytetyön teorian aihepiiri oli minulle pintapuolisesti tuttu entuudestaan. Olen aina ollut kiinnostunut tietotekniikan hyödyntämisestä arkipäiväisten ongelmien ratkaisussa ja NP-täydellisyydestä. Sovelluksen palvelinympäristö oli määrätty ennalta, mikä asetti tekniselle toteutukselle tietyt rajoitteet. Käytetyistä tekniikoista minulla ei ollut lainkaan aiempaa kokemusta. Moni asia toteutettiin useampaan kertaan yrityksen ja erehdyksen kautta. Lisäksi ainakin Liferayn vapaan lähdekoodin versiossa dokumentaatio on todella huonoa. Eniten ongelmia tuli karttakuvan esittämisessä järkevästi. HaminaKotkalta saatu karttapohja vaati suurta muokkausta ja käsittelyä, jotta sen soveltaminen reittihakuun oli edes mahdollista.

Sovellus vastaa sille asetettuja vaatimuksia ja se ratkaisee osan logistisesta kuljetusongelmien joukosta. Sillä voidaan myös tarkastella virtuaalista karttakuvaa satama-alueesta ja se toimii tietovarastona satamakalustoille. Reittihaku mukautuu automaattisesti kuljetuskalustojen, siirrettävien kappaleiden ja tiestön ominaisuuksien mukaan. Lisäksi reittihaun parametreja on mahdollista mukauttaa ajonaikaisesti ilman ohjelmointia. Sovelluksen karttanäkymään voisi jatkokehittää muokkaimen, jolla olisi mahdollista muokata järjestelmässä olevien teiden fyysisiä ominaisuuksia, poistaa tiestöjä ja lisätä uusia. Lisäksi karttakuvan hakeminen ja käsittely ei skaalaudu isompiin alueisiin, koska koko kartta on koko ajan ladattuna. Sovellus on rakennettu siten, että sen jatkokehittäminen on helppoa, koska siinä käytetään vahvaa kapselointia ja muita oliosuunnittelun hyviä periaatteita.

LÄHTEET

1. **Häkkinen, Jani; Posti, Antti; Tapaninen, Ulla; Heikkinen Juhani ja Himola Olli-Pekka.** *Mobiilisatamaesite.* 2010.

2. **Merikotka.** Mobiilistama - Mobile Port - MOPO. *Merikotka.* [Online] [Viitattu: 30. huhtikuuta 2012.] http://www.merikotka.fi/mopo/.

3. *A253504 Logistiikan perusteet - Kymenlaakson ammattikorkeakoulu.* **Niemi, Kyllikki.** 2012. Kurssimateriaali saatavissa Kymenlaakson ammattikorkeakoulun Moodle-alustalla.

4. **Karrus, Kaij E.** Logistiikan perusteet. *Tampereen teknillinen yliopisto.* [Online] [Viitattu: 3. toukokuuta 2012.] http://www.pori.tut.fi/infohakemisto/di/kurssimateriaalit/logistiikka/Logistiikka_1.pdf .

5. **Ruohonen, Keijo.** Ruohonen - Graafiteoria. *Tampereen teknillinen yliopisto.* [Online] 2006. [Viitattu: 30. huhtikuuta 2012.] http://math.tut.fi/~ruohonen/GT.pdf.

6. **Vesanen, Ari.** Algoritmit, NP-täydellisyydestä. *Tietojenkäsittelyn laitos - Oulun yliopisto.* [Online] 2008. [Viitattu: 30. huhtikuuta 2012.] http://www.tol.oulu.fi/kurssit/811386A/Luennot/Alg_NPTaydellisyys.pdf.

7. **Stergiou, Christos ja Siganos, Dimitrios.** Neural networks. *Imperial College.* [Online] 1996. [Viitattu: 30. huhtikuuta 2012.] http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html.

8. **Chern, Maw-Sheng.** Bee Algorithm. *Department of Industrial Engineering and Engineering Management - National Tsing Hua University.* [Online] [Viitattu: 26. toukokuuta 2012.] http://chern.ie.nthu.edu.tw/gen/10.pdf.

9. **Russell, Stuart ja Norvig, Peter.** *Artificial Intelligence: A Modern Approach (2nd Edition).* s.l. : Prentice Hall, 2002. ISBN 0137903952 .

10. **Katajisto, Teemu.** Java-servletit. *Tietotekniikan LuK-tutkielma - Jyväskylän yliopisto.* [Online] 14. Joulukuu 2000. [Viitattu: 3. toukokuuta 2012.] http://www.mit.jyu.fi/opetus/opinnayte/LuK/Java-servletit/.

11. **Vainio, Tuomo.** Web-portaalit ja niiden hyödyntäminen liiketoiminnassa. *Diplomityö - Tampereen teknillinen yliopisto.* [Online] Maaliskuu 2012. [Viitattu: 3. toukokuuta 2012.] http://dspace.cc.tut.fi/dpub/bitstream/handle/123456789/20906/vainio.pdf.

12. **Liferay Inc.** Liferay Portal 6.1 - User Guide. *Liferay Documentation.* [Online] [Viitattu: 30. huhtikuuta 2012.] http://www.liferay.com/documentation/liferay-portal/6.1/user-guide/-/ai/portal-architectu-5.

13. —. Dynamic Query API. *Liferay Wiki.* [Online] 26. Huhtikuu 2012. [Viitattu: 30. huhtikuuta 2012.] http://www.liferay.com/community/wiki/-/wiki/Main/Queries+2%3A+DynamicQuery+API.

14. **Grönroos, Marko.** Book of Vaadin: 4th Edition. *Vaadin.* [Online] 2012. [Viitattu: 30. huhtikuuta 2012.] https://vaadin.com/download/book-of-vaadin/4.0.0/pdf/book-of-vaadin.pdf.

15. **Vaadin Oy.** What is Vaadin? *Vaadin.* [Online] [Viitattu: 3. toukokuuta 2012.] https://vaadin.com/home.

16. **Ekblad, Sami.** Vaadin application in Liferay. *Vaadin.* [Online] [Viitattu: 3. toukokuuta 2012.] https://vaadin.com/web/sami/liferay.

17. **LT Konsultit Oy.** *Haminan sataman liikennejärjestelyt - Yleissuunnitelma.* 2001.

KYMENLAAKSON AMMATTIKORKEAKOULU
Tietotekniikka / Ohjelmistotekniikka

Jussi Raunio

NELIroute

Vaatimusmäärittely (SRS) 1.0

SISÄLLYSLUETTELO

1 JOHDANTO

Tässä dokumentissa kuvattu sovellus on osa Mobiilisatama-hankkeen informaatiokeskusta. Sovellus on kehitetty opinnäytetyönä Kymenlaakson ammattikorkeakoulun toimesta. Sovelluksen taustalla olevat teoriaosuudet ja valittujen tekniikoiden perustelut on käsitelty itse opinnäytetyössä. Informaatiokeskus on Liferay-alustalla toimiva web-pohjainen portaali, johon sovellus kehitetään servleteinä ja portleteina.

Satamaan tulee paljon ulkopuolista tavaraa. Tavaran siirtelyyn vaaditaan monien tahojen välistä tiedonvaihtoa sekä kuljetustavan valintaa ja reittihakua tavaran säilöön kuljettamiseksi. Sovellus on tarkoitettu helpottamaan sataman logistisia kuljetusmahdollisuuksien etsimistä tietokoneavusteisesti. Sillä pystytään hallitsemaan satamassa olevia kuljetuskalustoja ja satamanostureita, tarkastelemaan virtuaalista karttaa satama-alueesta sekä laskemaan reittejä sataman sisällä. Reittihakua käytetään pääsääntöisesti reitin etsimiseen sataman sisällä siirrettävälle kappaleelle. Reittihaussa huomioidaan kaluston ominaisuudet sekä tiestön rajoitukset.

Pääpiirteissään sovelluksen toiminta-alue voidaan jakaa seuraaviin osiin: puhdas tietovarasto, varastoidun tiedon hyväksikäyttäminen reittihaussa ja karttanäkymän esittäminen. Koska sovellus integroidaan osaksi sataman informaatiokeskusta, on otettava huomioon mahdolliset eri toimijoiden tarpeet. Esimerkiksi kuljetuskalustojen lisääminen voidaan rajata jollekin tietylle käyttäjäryhmälle ja sallia muille vain järjestelmässä olevien tietojen tarkastelu. Kuljetuskaluston tarkkaa määritelmää vältetään, koska ei tiedetä konkreettisia siirrettäviä kappaleita, josta seuraa se, ettei varsinaista laitetta tai konetta välttämättä voida määrittää. Sen sijaan sovelluksessa käytetään geneerisiä tietoja kuljetuskalustojen ominaisuuksista, joita erilaiset kalustot toteuttavat. Näitä ovat esimerkiksi suurin lastauskuorma, pituus, leveys ja massa. Satamanostureita käytetään ilmaisemaan käyttäjälle, missä tavaraa on mahdollista lastata ja jatkokäsitellä.

1.1 DOKUMENTIN TARKOITUS

Tämä dokumentti on tarkoitettu suunnittelun lähtökohdaksi, sopimukseksi toimijoiden välillä sekä testitapauksien suunnitteluun. Se sisältää kehitettävän ohjelmiston määrittelyn loppukäyttäjän näkökulmasta. Kuvaus on pyritty tekemään mahdollisimman yksityiskohtaisesti ja täsmällisesti, joten määrittely voi toimia myös osittain sovelluksen käyttöohjeena. Mahdolliset lisäkehitystarpeet ja -mahdollisuudet on myös kuvattu tässä dokumentissa.

1.2 TUOTE

Ohjelman nimi on *NELIroute*. Se on tarkoitettu ylläpitämään tietoja sataman kuljetuskalustoista ja satamanostureista sekä helpottamaan kuljetuslogistiikkaa. Sovelluksella on mahdollista etsiä paras reitti minkä tahansa pisteen välillä satama-alueen sisällä. Reittihakua käytetään pääsääntöisesti eri kappaleiden siirtelyn suunnitteluun satama-alueen sisällä. Se ottaa huomioon kaluston ominaisuudet sekä tiestön rajoitteet.

Sovelluksen käyttäjä voi lisätä, poistaa ja muokata kuljetuskalustoja ja satamanostureita, tarkastella virtuaalista karttaa satama-alueesta, suorittaa reittihakua sataman sisällä ja muuttaa tiestön ominaisuuksia.

1.3 MÄÄRITELMÄT, TERMIT JA LYHENTEET

Sovellus	Tässä dokumentissa kuvattu ohjelmisto kokonaisuudessaan. Kattaa kaikki portletit (ml. niiden instanssit), servletit ja muun WAR-paketoidun sisällön.
WAR-paketti	Sovelluksen asennuspaketti. Sisältää kokonaisuudessaan kaikki asetustiedostot, binäärit ja muut ajonaikaisesti tarvittavat tiedostot. Voidaan asentaa Liferayn ohjauspaneelista.

1.4 VIITTEET JA LIITTYVÄT DOKUMENTIT

Tämä määrittelydokumentti on toimitettu Kymenlaakson ammattikorkeakoulussa tehdyn opinnäytetyön liitteenä. Opinnäytetyössä on kuvattu sovelluksen taustalla olevat teoriaosuudet ja perustelut valituille tekniikoille.

1.5 YLEISKATSAUS DOKUMENTTIIN

Tässä luvussa kerrotaan sovelluksesta lähtökohdista yleisellä tasolla. Luvussa 2 kuvataan sovelluksen käyttöympäristö, rajoitteet ja toiminta yleisellä tasolla. Luvussa 3 on tarkat kuvaukset sovelluksen käyttöliittymästä ja toiminnoista. Luku 4 kuvaa sovelluksen tietokannan rakenteen, mutta ei yksityiskohtaisia kuvauksia tietokannan tauluista. Yksityiskohtaiset kuvaukset ja tallennettavat tiedot määritellään teknisessä määrittelydokumentissa luvun 3 pohjalta. Luku 5 ja 6 keskittyvät sovelluksen erilaisiin suunnittelunäkökulmiin.

2 YLEISKUVAUS

Sovellus tulee osaksi mobiilisataman informaatiokeskusta. Informaatiokeskus on web-pohjainen portaali, joka toimii Liferay-alustalla. Sovellus integroidaan osaksi järjestelmään servleteinä ja portleteina.

2.1 YMPÄRISTÖ

Informaatiokeskus on keskitetty lennonjohtomainen keskus satamatoimijoiden välillä. Sovellus tulee osaksi informaatiokeskusta ja se on kehitetty eritoten tehostamaan logistisia erikoiskuljetuksia ja auttamaan niiden suunnittelussa satama-alueen sisällä.

2.2 TOIMINTA

- Virtuaalinen karttanäkymä
    - Koon muuttaminen
    - Satama-alueen esittäminen

- Tiestön ja reittien esittäminen
- Satamanostureiden esittäminen

- Reittihaku
  - Lyhin reitti kahden pisteen välillä
  - Rajoitteiden huomioiminen
    - Kuljetettavan kappaleen tiedot
    - Kuljetuskaluston ominaisuudet
    - Tiestön rajoitteet
- Kuljetuskalustojen hallinta
  - Uusien lisääminen järjestelmään
  - Olemassa olevien kalustojen poistaminen järjestelmästä
  - Olemassa olevan kaluston ominaisuuksien muokkaaminen
- Satamanostureiden hallinta
  - Uusien lisääminen järjestelmään
  - Olemassa olevien nostureiden poistaminen järjestelmästä
  - Olemassa olevan nosturin ominaisuuksien muokkaaminen
  - Nostureiden sijainnin muuttaminen kartalla

2.3 KÄYTTÄJÄT

Käyttäjinä ovat mobiilisatamatoimijat ja käyttöympäristönä normaali toimistotila. Käyttäjä on normaaliin toimistotyöhön tottunut suomen kieltä osaava henkilö, jolla on kokemusta WWW-selaimen ja eri sovellusten käytöstä.

2.4 YLEISET RAJOITTEET

Sovellus on Javapohjainen web-sovellus. Sovelluksella on pääsy Liferayn palveluihin, kuten tietokantaan ja pääsynhallintaan. Sovelluksen tulee noudattaa Liferayn määrittelemää tapaa toteuttaa sovelluksia, jotta siirrettävyys ja muut portaalistandardissa JSR 286 määritellyt alustariippumattomuuden ehdot täyttyvät. Käytännössä tämä tarkoittaa esi-

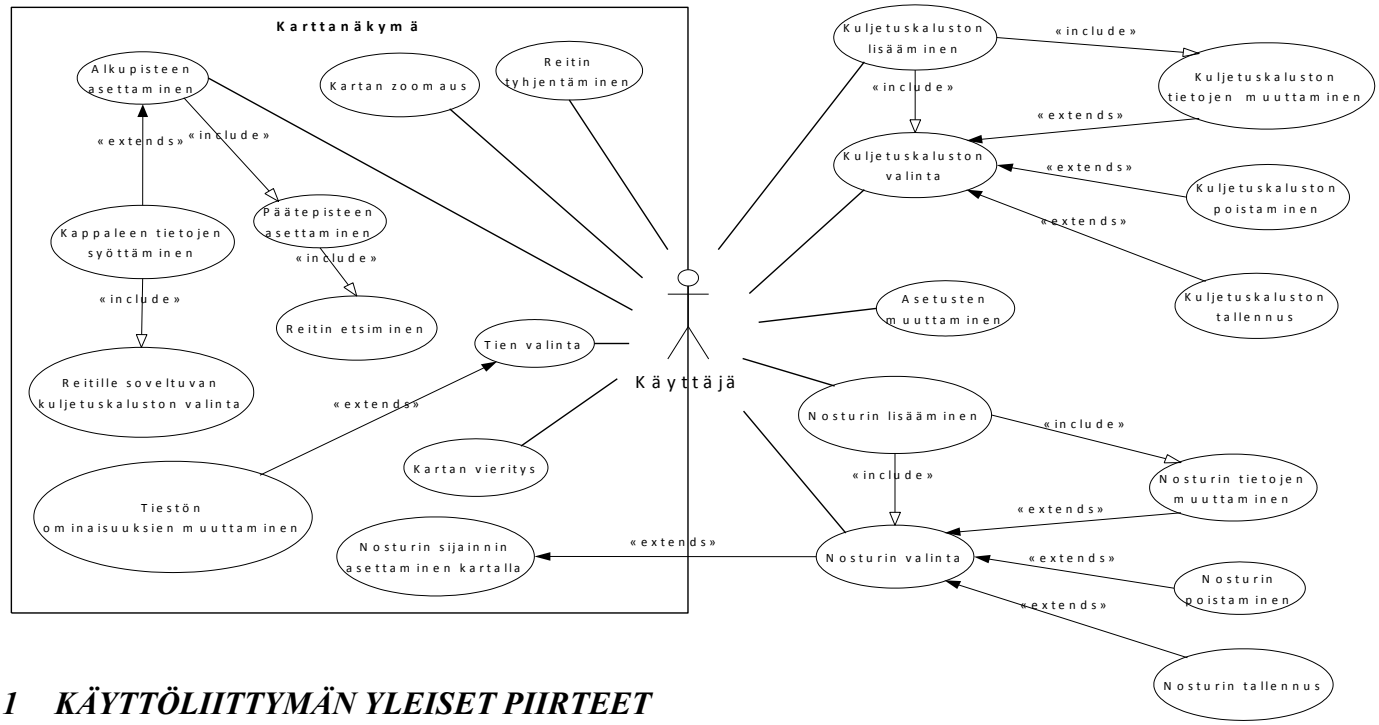
merkiksi, että sovelluksen ei tule käyttää tietokantaa matalalla tasolla (SQL-kyselyt); tietokannan käyttöön Liferayssä on korkeatason rajapinnat.

2.5 OLETUKSET JA RIIPPUVUUDET

Sovelluksen kehityksessä oletetaan, että palvelinkone on vähintään keskitasoa, jossa Liferay 6.0 tai uudempi asennettuna ja konfiguroituna. Asiakaspuolella oletetaan, että loppukäyttäjällä on käytössä moderni WWW-selain, jossa JavaScript-tuki päällä. Moderneja WWW-selaimia ovat muun muassa Internet Explorer 6 tai uudempi, Firefox 3.0 tai uudempi, Safari 3.2 tai uudempi, Opera 9.6 tai uudempi ja Google Chrome.

Sovellus toimitetaan WAR-paketoituna JSR 286 -portaalistandardin täyttävänä sovelluksena. Asennuksessa oletetaan, että palvelimelle on ylläpitäjän oikeudet ja osaaminen asentaa WAR-paketoituja portaalisovelluksia Liferay-alustalle.

3 TOIMINNOT JA KÄYTTÖLIITTYMÄ

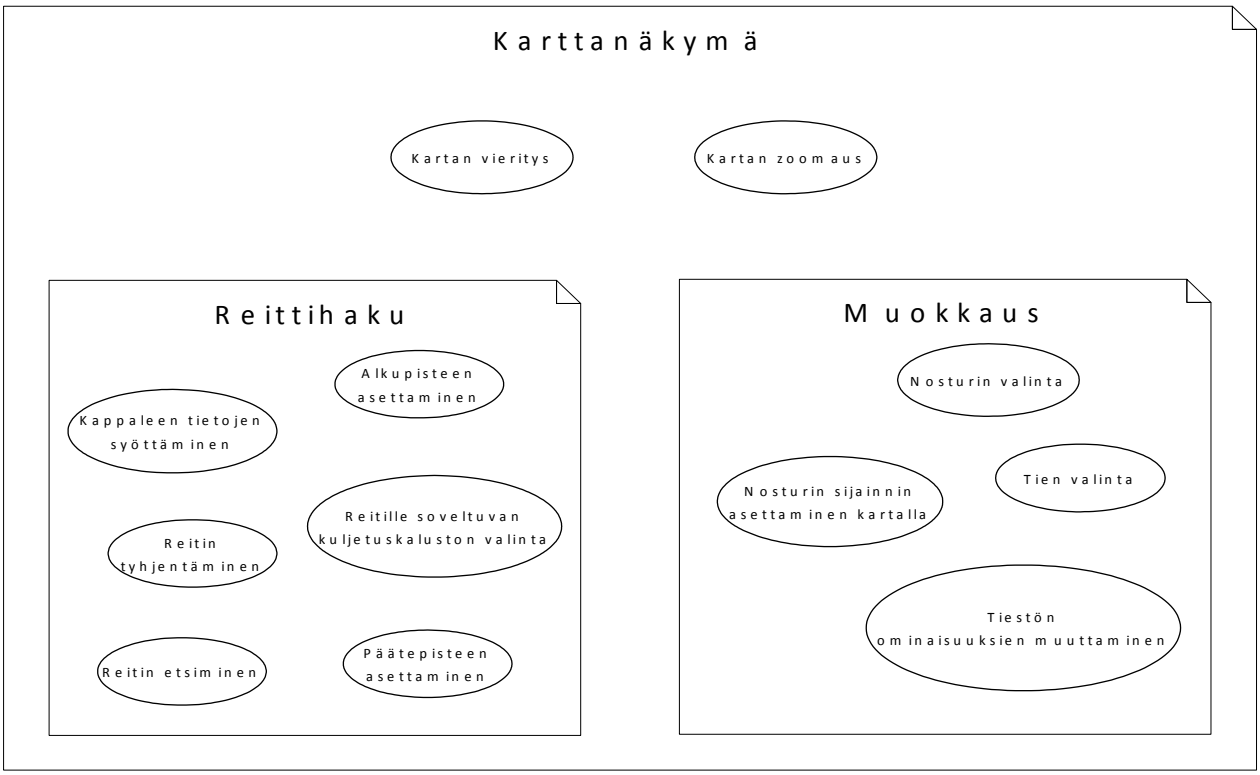


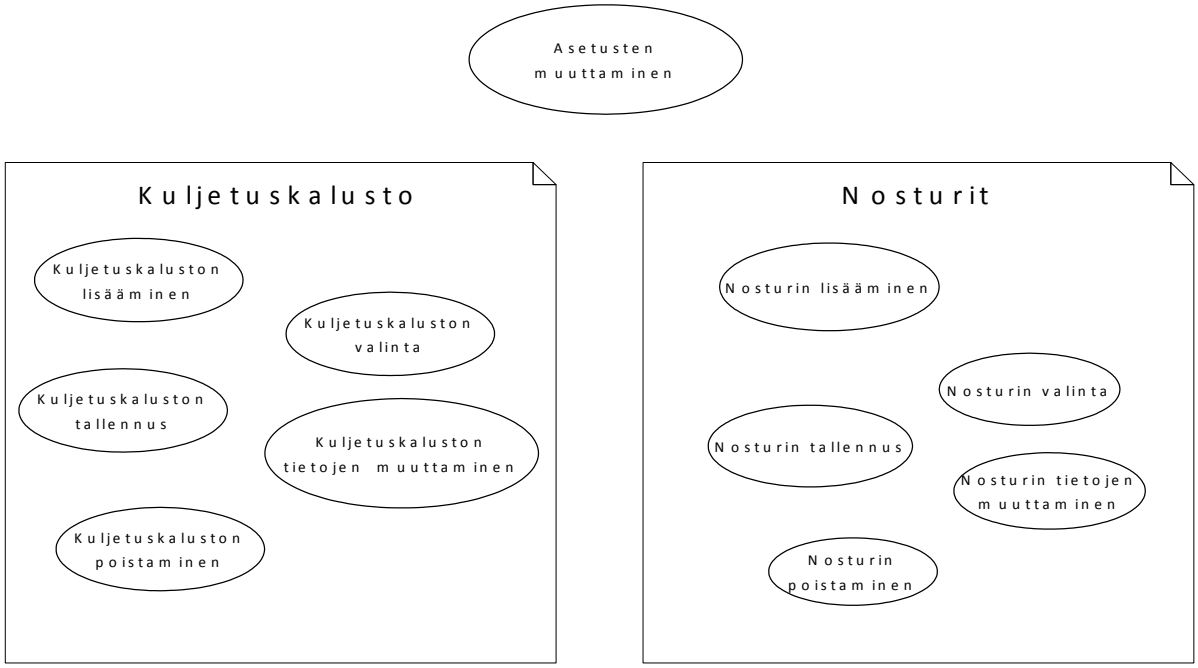
3.1 KÄYTTÖLIITTYMÄN YLEISET PIIRTEET

Sovellus on web-pohjainen, joten se noudattaa modernille WWW:lle tyypillisiä piirteitä. Toisaalta sovelluksen luonteen takia, se pyrkii olemaan lähellä tavanomaisia työpöytäsovelluksia. Esimerkiksi näppäinkomentoja ei käytetä, koska käyttäjä voisi hämääntyä normaalista WWW:stä poikkeavasta käyttäytymisestä. Tiedonvaihto tapahtuu palvelimen ja asiakkaan (tässä: selaimen) välillä saumattomasti. Lomakkeiden ja muiden tietojen oikeellisuus tarkistetaan heti, kun se on mahdollista ja ilmoitetaan käyttäjälle mahdollisesta virheestä. Erilaisten tietojen tallennus tapahtuu koko ajan automaattisesti taustalla, mutta käyttäjälle annetaan mahdollisuus myös manuaaliseen tallennukseen.

3.2 NÄYTÖT

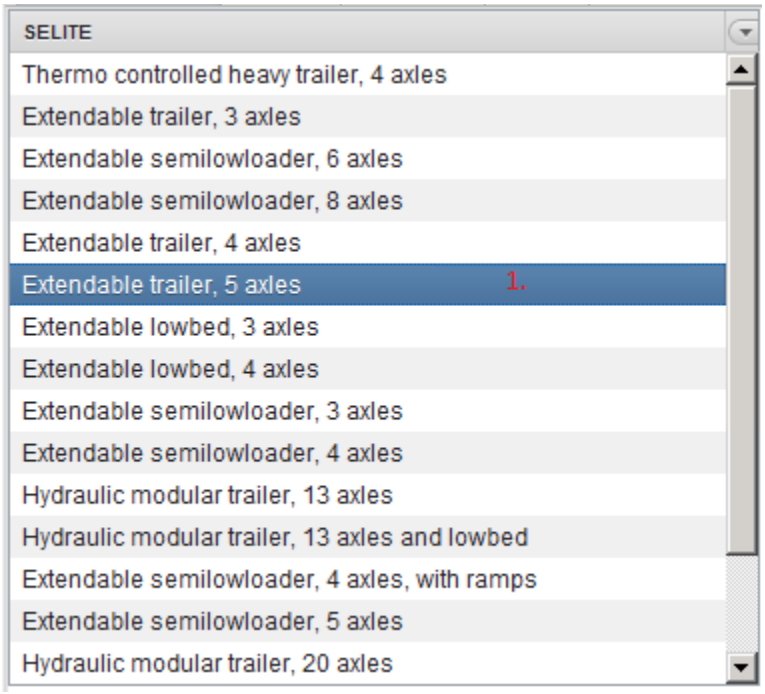
Sovellus on kaksiosainen, jossa toinen osa koostuu tietojen lisäämisestä, muokkaamisesta ja poistamisesta. Toinen osa käsittää karttanäkymän, reittihaun ja muut karttaa vaativat toimet. Koska sovellus on web-pohjainen, näytöt ovat eri sivuja, joille siirtyminen tapahtuu linkkien avulla. Alla on esitelty sovelluksen näytöt ja toiminnot, jotka ovat kussakin näytössä.





3.3 KULJETUSKALUSTON VALINTA

- Valinta tapahtuu taulukosta, joka listaa kaikki kuljetuskalustot.
    - Jos kuljetuskalustoja ei ole, näytetään tyhjä taulukko.
- Valinta tapahtuu klikkaamalla haluttua kuljetuskalustoa (1).
- Valittu kuljetuskalusto on aktiivinen ja sen tiedot tulevat muokattavaksi lomakkeeseen (ks. 3.7).
- Jos jokin kuljetuskalusto on valittuna ja käyttäjä yrittää valita toista tai poistaa valinnan, tallennetaan muokattavan kaluston tiedot tietokantaan ja valitaan uusi.
    - Jos muokkauslomakkeessa on virheellisiä tietoja, tietoja ei tallenneta tietokantaan, eikä uutta kuljetuskalustoa valita.
- Kun mitään kuljetuskalustoa ei ole valittuna, piilotetaan myös muokkauslomake.
- Karttanäkymässä:
    - Kuljetuskaluston muokkauslomake on aina piilotettu.



3.4 KULJETUSKALUSTON LISÄÄMINEN

- Lisäys tapahtuu painamalla ”*Lisää*”-painiketta.
- Uusi kuljetuskalusto lisätään taulukon ensimmäiseksi ja se tulee valituksi (ks. 3.3).
- Uuden kuljetuskaluston tiedot ovat oletuksina.
    1. Numeerisille tiedoille 0.
    2. Tekstitiedoille tyhjä merkkijono.
    3. Jos edellisiä vaatimuksia ei voida täyttää, käytetään oletustietojen määrittämisessä seuraavaa sääntöä: kuljetuskalusto täytyy pystyä tallettamaan tietokantaan oletustiedoilla ja tietojen pitää olla kyseisessä kentässä sallittuja.

3.5 KULJETUSKALUSTON POISTAMINEN

- Poisto tapahtuu painamalla ”*Poista*”-painiketta.
- Poistettu kuljetuskalusto poistuu taulukosta ja mikään kuljetuskalusto ei ole valittuna poiston jälkeen (ks. 3.3).

- Jos mitään kuljetuskalustoa ei ole valittuna, painike on näkyvillä, mutta poissa käytöstä.

3.6 KULJETUSKALUSTON TALLENNUS

- Tietojen automaattinen tallennus tapahtuu koko ajan taustalla, eikä käyttäjältä vaadita tietojen syöttämisen lisäksi muita toimia.
- Manuaalinen tallennus tapahtuu painamalla ”*Tallenna*”-painiketta.
- Manuaalisen tallennuksen jälkeen muokattavan kuljetuskaluston nykyiset tiedot tallennettaan välittömästi tietokantaan ja mikään kuljetuskalusto ei ole valittuna tallennuksen jälkeen (ks. 3.3).
    - Jos muokkauslomakkeen kentissä on virheitä, kuljetuskaluston tietoja ei tallenneta tietokantaan, eikä valintaa poisteta.
- Jos mitään kuljetuskalustoa ei ole valittuna, painike on näkyvillä, mutta poissa käytöstä.

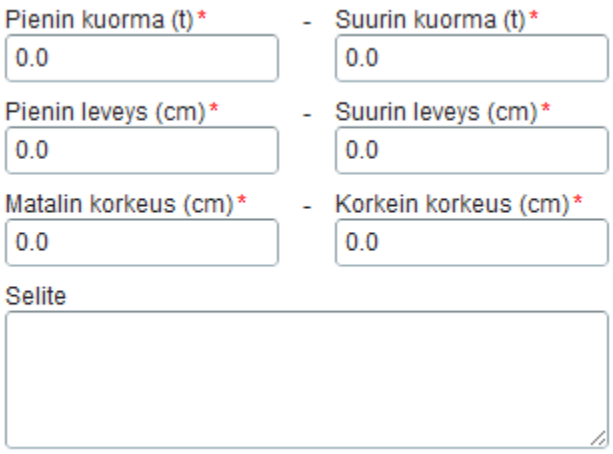
3.7 KULJETUSKALUSTON TIETOJEN MUUTTAMINEN

- Pienin kuorma
    - Pienin lastauskuorma, jota tällä kuljetuskalustolla kannattaa kuljettaa.
    - Syötteenä hyväksytään positiiviset desimaaliluvut.
        - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
    - Syötteen on oltava pienempi tai yhtä suuri kuin *Suurin kuorma.*
    - Syöte tonneissa.
    - Pakollinen tieto.
    - Tieto tallennetaan tietokannan entiteettiin *Transport*.
- Suurin kuorma
    - Suurin lastauskuorma, jota tällä kuljetuskalustolla voi kuljettaa.
    - Syötteenä hyväksytään positiiviset desimaaliluvut.

        - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
    - Syötteen on oltava suurempi tai yhtä suuri kuin *Pienin kuorma*.
    - Syöte tonneissa.
    - Pakollinen tieto.
    - Tieto tallennetaan tietokannan entiteettiin *Transport*.
- Pienin leveys
    - Kuorman pienin leveys, jota tällä kuljetuskalustolla kannattaa kuljettaa.
    - Syötteenä hyväksytään positiiviset desimaaliluvut.
        - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
    - Syötteen on oltava pienempi tai yhtä suuri kuin *Kuorman maksimileveys*.
    - Syöte senttimetreissä.
    - Pakollinen tieto.
    - Tieto tallennetaan tietokannan entiteettiin *Transport*.
- Suurin leveys
    - Kuorman suurin leveys, jota tällä kuljetuskalustolla voi kuljettaa.
    - Syötteenä hyväksytään positiiviset desimaaliluvut.
        - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
    - Syötteen on oltava suurempi tai yhtä suuri kuin *Kuorman minimileveys*.
    - Syöte senttimetreissä.
    - Pakollinen tieto.
    - Tieto tallennetaan tietokannan entiteettiin *Transport*.
- Matalin korkeus
    - Kuorman matalin korkeus, jota tällä kuljetuskalustolla kannattaa kuljettaa.
    - Syötteenä hyväksytään positiiviset desimaaliluvut.
        - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.

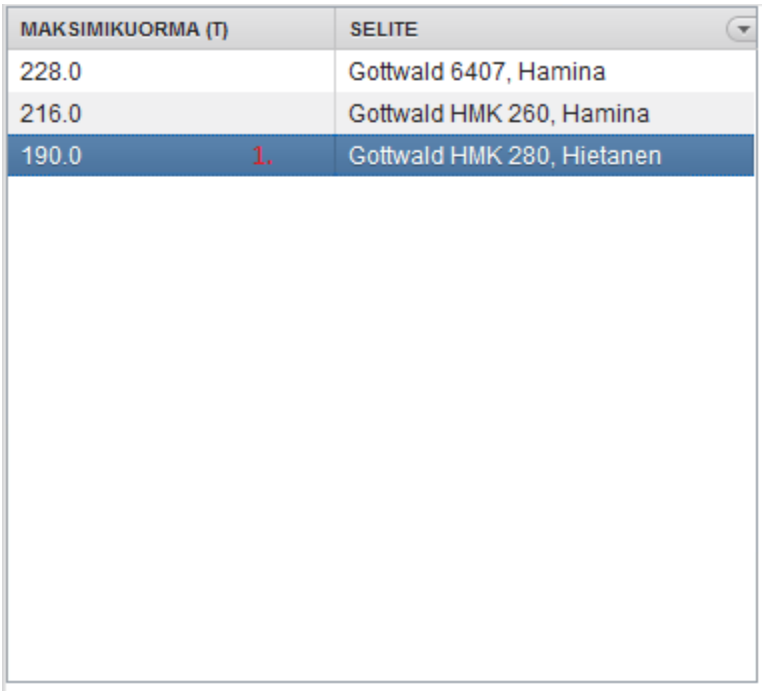
- Syötteen on oltava pienempi tai yhtä suuri kuin *Kuorman maksimikorkeus*.
- Syöte senttimetreissä.
- Pakollinen tieto.
- Tieto tallennetaan tietokannan entiteettiin *Transport*.

- Korkein korkeus
  - Kuorman korkein korkeus, jota tällä kuljetuskalustolla voi kuljettaa.
  - Syötteenä hyväksytään positiiviset desimaaliluvut.
    - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
  - Syötteen on oltava suurempi tai yhtä suuri kuin *Kuorman minimikorkeus*.
  - Syöte senttimetreissä.
  - Pakollinen tieto.
  - Tieto tallennetaan tietokannan entiteettiin *Transport*.
- Selite
  - Kuljetuskalustoa kuvaava selite.
  - Syötteen ensimmäistä riviä käytetään kuljetuskaluston tulostuksissa.
  - Vapaavalintainen tieto.



3.8 NOSTURIN VALINTA

- Valinta tapahtuu taulukosta, joka listaa kaikki nosturit.
    - Jos nostureita ei ole, näytetään tyhjä taulukko.
- Valinta tapahtuu klikkaamalla haluttua nosturia (1).
- Valittu nosturi on aktiivinen ja sen tiedot tulevat muokattavaksi lomakkeeseen (ks. 0).
- Jos jokin nosturi on valittuna ja käyttäjä yrittää valita toista tai poistaa valinnan, tallennetaan muokattavan nosturin tiedot tietokantaan ja valitaan uusi.
    - Jos muokkauslomakkeessa on virheellisiä tietoja, tietoja ei tallenneta tietokantaan, eikä uutta nosturia valita.
- Kun mitään nosturia ei ole valittuna, piilotetaan myös muokkauslomake.
- Karttanäkymässä:
    - Nosturia ei voi valita, jos jokin tie on valittuna.
    - Nosturin muokkauslomake on aina piilotettu.



3.9 NOSTURIN LISÄÄMINEN

- Lisäys tapahtuu painamalla ”*Lisää*”-painiketta.
- Uusi nosturi lisätään taulukon ensimmäiseksi ja se tulee valituksi (ks. 0).
- Uuden nosturin tiedot ovat oletuksina.
    1. Numeerisille tiedoille 0.
    2. Tekstitiedoille tyhjä merkkijono.
    3. Jos edellisiä vaatimuksia ei voida täyttää, käytetään oletustietojen määrittämisessä seuraavaa sääntöä: nosturi täytyy pystyä tallettamaan tietokantaan oletustiedoilla ja tietojen pitää olla kyseisessä kentässä sallittuja.

3.10 NOSTURIN POISTAMINEN

- Poisto tapahtuu painamalla ”*Poista*”-painiketta.
- Poistettu nosturi poistuu taulukosta ja mikään nosturi ei ole valittuna poiston jälkeen (ks. 0).
- Jos mitään nosturia ei ole valittuna, painike on näkyvillä, mutta poissa käytöstä.

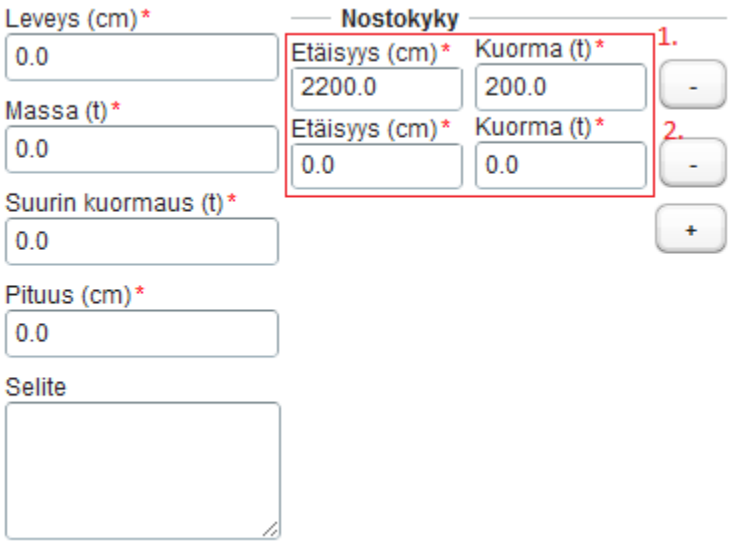
3.11 NOSTURIN TALLENNUS

- Tietojen automaattinen tallennus tapahtuu koko ajan taustalla, eikä käyttäjältä vaadita tietojen syöttämisen lisäksi muita toimia.
- Manuaalinen tallennus tapahtuu painamalla ”*Tallenna*”-painiketta.
- Manuaalisen tallennuksen jälkeen muokattavan nosturin nykyiset tiedot tallennettaan välittömästi tietokantaan ja mikään nosturi ei ole valittuna tallennuksen jälkeen (ks. 0).
    - Jos muokkauslomakkeen kentissä on virheitä, nosturin tietoja ei tallenneta tietokantaan, eikä valintaa poisteta.
- Jos mitään kuljetuskalustoa ei ole valittuna, painike on näkyvillä, mutta poissa käytöstä.

3.12 NOSTURIN TIETOJEN MUUTTAMINEN

- Leveys
    - Nosturin fyysinen leveys.
    - Syötteenä hyväksytään positiiviset desimaaliluvut.
        - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
    - Syöte senttimetreissä.
    - Pakollinen tieto.
    - Tieto tallennetaan tietokannan entiteettiin *Hoister*.
- Massa
    - Nosturin fyysinen massa.
    - Syötteenä hyväksytään positiiviset desimaaliluvut.
        - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
    - Syöte tonneissa.
    - Pakollinen tieto.
    - Tieto tallennetaan tietokannan entiteettiin *Hoister*.
- Suurin kuormaus
    - Suurin mahdollinen kuorma, jonka tällä nosturilla kannattaa tai voi nostaa.
    - Syötteenä hyväksytään positiiviset desimaaliluvut.
        - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
    - Syöte tonneissa.
    - Pakollinen tieto.
    - Tieto tallennetaan tietokannan entiteettiin *Hoister*.
- Pituus
    - Nosturin fyysinen pituus.
    - Syötteenä hyväksytään positiiviset desimaaliluvut.

        - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
    - Syöte senttimetreissä.
    - Pakollinen tieto.
    - Tieto tallennetaan tietokannan entiteettiin *Hoister*.
- Selite
    - Nosturia kuvaava selite.
    - Syötteen ensimmäistä riviä käytetään nosturin tulostuksissa.
    - Vapaavalintainen tieto.
- Nostokyky (ks. 3.12.1)
    - Nosturin nostokyky voi vaihdella riippuen korkeudesta ja muista tekijöistä.
    - Kaikki nosturin nostokyvyt listataan allekkain ja kaikkien nostoparametrien muuttaminen on mahdollista (1).
    - Uuden nostokyvyn lisääminen nosturiin tapahtuu painamalla ”+”-painiketta.
        - Uusi nostokyky lisätään oletustiedoin listan alimmaiseksi (2).
    - Nostokyvyn poistaminen nosturista tapahtuu painamalla halutun nostokyvyn vieressä olevaa ”-”-painiketta.



3.12.1 NOSTOKYKY

- Etäisyys
  - Nostoetäisyys.
  - Syötteenä hyväksytään positiiviset desimaaliluvut.
    - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
  - Syöte senttimetreissä.
  - Pakollinen tieto.
  - Tieto tallennetaan tietokannan entiteettiin *HoisterParam*.
- Kuorma
  - Nostokuorma kyseisellä korkeudella.
  - Syötteenä hyväksytään positiiviset desimaaliluvut.
    - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
  - Syöte tonneissa.
  - Pakollinen tieto.
  - Tieto tallennetaan tietokannan entiteettiin *HoisterParam*.

3.13 ASETUSTEN MUUTTAMINEN

- Sovelluksen asetuksia pystyy muuttamaan vain ylläpitäjän valtuuttamat henkilöt.
- Asetukset ovat instanssikohtaisia. Jos sivulla on kaksi sovellusinstanssia (esim. portlettia), portleteilla voi olla eri asetukset.
- Ihanteellinen tien alikulkukorkeus
  - Suositeltu tien alikulkukorkeus. Reittihaku suosii teitä, jolla vähintään tämä alikulkukorkeus.
  - Ottaa huomioon vain siirrettävän tavaran, ei kuljetuskalustoa.
  - Syötteenä hyväksytään positiiviset desimaaliluvut, jotka ovat suurempia kuin nolla.

        - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
    - Syöte senttimetreissä.
    - Pakollinen tieto.
    - Oletustieto:
        - 380.
    - Tieto tallennetaan tietokannan entiteettiin *Setting*.
- Ihanteellinen tien kuorma
    - Suositeltu tien kuormaus. Reittihaku suosii teitä, jolla vähintään tämä kuormaus.
    - Ottaa huomioon vain siirrettävän tavaran, ei kuljetuskalustoa.
    - Syötteenä hyväksytään positiiviset desimaaliluvut, jotka ovat suurempia kuin nolla.
        - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
    - Syöte tonneissa.
    - Pakollinen tieto.
    - Oletustieto:
        - 120.
    - Tieto tallennetaan tietokannan entiteettiin *Setting*.
- Ihanteellinen tien leveys
    - Suositeltu tien leveys. Reittihaku suosii teitä, jolla vähintään tämä leveys.
    - Ottaa huomioon vain siirrettävän tavaran, ei kuljetuskalustoa.
    - Syötteenä hyväksytään positiiviset desimaaliluvut, jotka ovat suurempia kuin nolla.
        - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
    - Syöte senttimetreissä.
    - Pakollinen tieto.

- Oletustieto:
  - 1 200.
- Tieto tallennetaan tietokannan entiteettiin *Setting*.

- Asetusten tallentaminen tapahtuu painamalla ”*Tallenna*”-painiketta.
  - Jos asetuksissa on virheitä, näytetään virheviestit ja jätetään painallus muuten huomioimatta.
- Muutettujen asetusten hylkääminen tapahtuu painamalla ”*Hylkää*”-painiketta.
  - Tyhjentää vain lomakkeen viimeisiin tallennettuihin tietoihin.
  - Ei poista mitään tietokannasta.
- Kaikkien asetusten poistaminen tapahtuu painamalla ”*Palauta asetukset alkutilaan*”.
  - Tämä palauttaa koko sovelluksen alkutilaansa.
  - Poistaa vain sovelluskohtaiset tiedot, ei siis tietokantaan tallennettuja muita tietoja.

3.14 KARTTANÄKYMÄ

Karttanäkymä on kaksipalstainen, jossa karttakuva vie noin 2/3 tilasta. Loput tilasta jakautuu karttakontrolleiden kesken. Kartan alapuolella on zoomaus-kontrollit ja oikealla

puolella on reittihakuun liittyvät kontrollit. Karttanäkymä täyttää aina koko selaimen ikkunan ja on aina yhtä suuri kuin ikkunan koko. Lisäksi karttanäkymässä on erillinen muokkaustila, jossa voidaan asettaa nostureita kartalle ja muuttaa tiestön ominaisuuksia. Muokkaustilassa ei voi suorittaa reittihakua.

Reittihaku voidaan suorittaa heti, kun päätepiste asetetaan, jolloin sovellus tekee triviaalin reittihaun ilman rajoitteita. Kaikki muuta reittihaun tiedot ovat vapaavalintaisia, joten sovellus ottaa huomioon vain annetut tiedot ja muita tietoja käsitellään tyhjänä.

Karttanäkymän alkutila on seuraava:

- Kaikki tiet ja nosturit ovat näkyvissä kartalla.
- Mitään ei ole valittuna.
- Käyttäjää pyydetään asettamaan reittihaun alkupiste.

3.14.1 NOSTURIN SIJAINNIN ASETTAMINEN KARTALLA

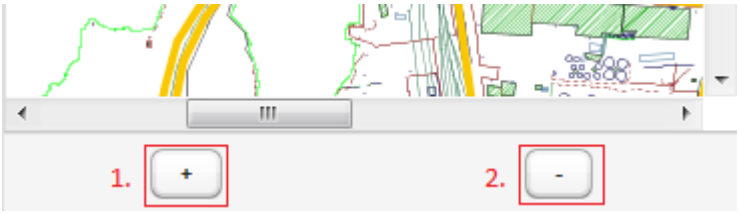
- Nosturi valitaan taulukosta (ks. 0).
- Karttanäkymässä näkyy vain valitun nosturin ikoni, jos nosturin paikka on asetettu.
- Jos paikkaa ei ole asetettu ennalta, nosturi asetetaan kartalle klikkaamalla hiirellä haluttua kohtaa. Nosturin ikoni seuraa kursoria koko ajan karttanäkymän päällä.
    - Klikkauksen jälkeen nosturi ja ikoni asetetaan kartalle kursorin kohtaa ja ikoni lopettaa kursorin seuraamisen.
- Klikkaamalla jo asetetun nosturin ikonia karttanäkymässä, poistuu nosturin sijaintitieto ja nosturi käyttäytyy kuin sen paikkaa ei olisi koskaan ollut asetettuna.
- Nosturin sijaintitiedot tallennetaan tietokannan entiteettiin *HoisterPOI*.

3.14.2 KARTAN VIERITYS

- Karttanäkymää on mahdollista vierittää eri suuntiin.
- Vierittäminen kartan reunojen yli ei ole mahdollista.
- Vierittäminen tapahtuu hiirellä raahaamalla karttakuvaa (kuten esimerkiksi PDF-dokumentissa) tai painikkeilla.
    - Suositeltu tapa toteuttaa painikevieritys on WWW-sivuille tavanomaiset vierityspalkit (1 ja 2).
- Jos karttakuva on jo vieritetty ääripäähän, vieritysyritys jätetään huomioimatta.

3.14.3 KARTAN ZOOMAUS

- Skaalaus on välillä 1 - ∞, jossa oletusskaalaus on 1.
- Kartan lähennys tapahtuu painamalla kartan alapuolella olevaa ”+”-painiketta (1).
  - Lähentäminen kasvattaa kartan skaalausta yhdellä yksiköllä.
- Kartan loitonnus tapahtuu painamalla kartan alapuolella olevaa ”-”-painiketta (2).
  - Loitontaminen pienentää kartan skaalausta yhdellä yksiköllä.
  - Jos skaalaus on 1, painike on näkyvillä, mutta poissa käytöstä.



3.14.4 ALKUPISTEEN ASETTAMINEN

- Alkupisteen ikoni seuraa kursoria karttanäkymän päällä.
- Piste asetetaan klikkaamalla haluttua tien kohtaa kartalla, jonka jälkeen alkupiste asettuu kartalle klikattuun kohtaan ja lopettaa kursorin seuraamisen.
- Jos klikkauskohta ei ole välittömästi tien päällä, etsitään lähin tie 50 kartan yksikön alueella ja asetetaan alkupiste tien päälle.
  - Jos tietä ei löydy, jätetään klikkaus huomioimatta.

3.14.5 PÄÄTEPISTEEN ASETTAMINEN

- Päätepisteen ikoni seuraa kursoria karttanäkymän päällä.
- Piste asetetaan klikkaamalla haluttua tien kohtaa kartalla, jonka jälkeen päätepiste asettuu kartalle klikattuun kohtaan, lopettaa kursorin seuraamisen ja reitti etsitään pisteiden välille.
- Jos klikkauskohta ei ole välittömästi tien päällä, etsitään lähin tie 50 kartan yksikön alueella ja asetetaan päätepiste tien päälle.
  - Jos tietä ei löydy, jätetään klikkaus huomioimatta.

- Päätepistettä ei voi asettaa, jos alkupistettä ei ole asetettu.

3.14.6 KAPPALEEN TIETOJEN SYÖTTÄMINEN

- Massa
  - Siirrettävän kappaleen massa.
  - Syötteenä hyväksytään positiiviset desimaaliluvut.
    - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
  - Syöte tonneissa.
  - Vapaavalintainen tieto.
- Mitat
  - Siirrettävän kappaleen mitat muodossa *leveys* x *korkeus* x *syvyys*.
  - Jokainen ulottuvuus on omana kenttänä.
  - Kaikille kentille pätee seuraavat määritelmät:
    - Syötteenä hyväksytään positiiviset desimaaliluvut.
      - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
    - Syöte senttimetreissä.
    - Vapaavalintainen tieto.
- Kappaleen tiedot päivitetään reittihakuun painamalla ”*OK*”-painiketta.
  - Jos tiedoissa on virheitä, jätetään painallus huomioimatta.
- Jos alkupistettä ei ole asetettu, näytetään lomake, mutta se on poissa käytöstä.

3.14.7 REITILLE SOVELTUVAN KULJETUSKALUSTON VALINTA

- Kaluston soveltuvuus määräytyy kappaleen tietojen perusteella.
- Jos kappaleen tietoja ei ole annettu, kuljetuskalustoja ei voi valita.
- Reitille soveltuvat kuljetuskalustot listataan taulukkoon, josta niitä voidaan valita (ks. 3.3).
    - Jos soveltuvia kuljetuskalustoja ei löydy, näytetään tyhjä taulukko.
- Soveltuvia kuljetuskalustoja yritetään sovittaa kaluston minimi- ja maksimiominaisuuksien mukaan.
    - Jos yhtäkään soveltuvaa kuljetuskalustoa ei näin löydy, käytetään vain kaluston maksimirajoitteita.
- Kuljetuskaluston valinta aiheuttaa reitin uudelleenlaskemisen.
- Kuljetuskaluston valinnan poistaminen suorittaa reittihaun ilman kappalerajoitteita.
- Jos alkupistettä ei ole asetettu, näytetään taulukko, mutta se on poissa käytöstä.

3.14.8 REITIN TYHJENTÄMINEN

- Reitti voidaan tyhjentää painamalla ”*Tyhjennä reitti*”-painiketta.
- Painike on aina saatavilla.
- Jos reittiä ei ole, painikkeen painallus jätetään huomioimatta.
    - Reitiksi oletetaan kaikki muutokset alkutilasta.
- Reitin tyhjentämisen jälkeen reittihaku ja karttanäkymä palaavat alkutilaansa.

3.14.9 REITIN ETSIMINEN

- Reitti alkupisteen (1) ja päätepisteen (2) välillä korostetaan kartalla.
- Jos reittiä ei löydy, ilmoitetaan siitä virheviestillä ja tyhjennetään reitti (ks. 3.14.8).

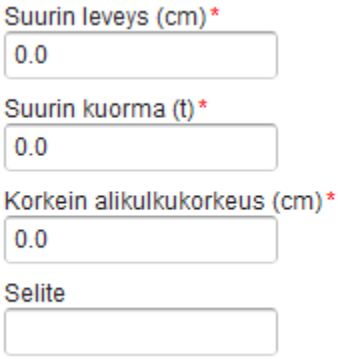
3.14.10 TIEN VALINTA

- Tie valitaan klikkaamalla sitä karttanäkymässä.
- Valittu tie on korostettuna kartalla (1).
    - Korostus mieluiten samalla tavalla kuin reittihaussa korostettu reitti.
- Jos klikkauskohta ei ole välittömästi tien päällä, etsitään lähin tie 50 kartan yksikön alueelta ja valitaan tämä tie.
    - Jos tietä ei löydy, poistetaan teiden valinnat.
- Valitun tien ominaisuudet tulevat muokattavaksi lomakkeelle (ks. 3.14.11).
- Jos tietä ei ole valittuna, lomake on piilotettuna.
- Tien valinta voidaan vaihtaa klikkaamalla jotakin muuta tietä kartasta.
- Jos jokin nostureista on valittuna, kun tie valitaan, tien valinta ylikirjoittaa nosturin valinnan.

3.14.11 TIESTÖN OMINAISUUKSIEN MUUTTAMINEN

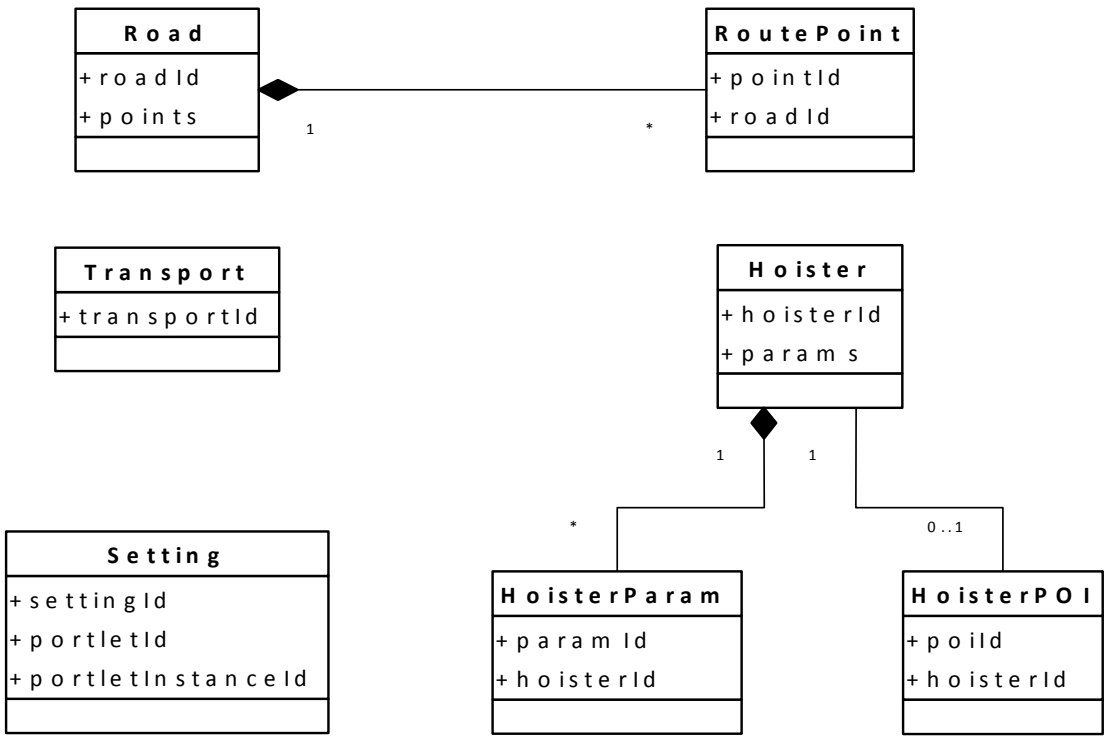
- Suurin leveys
    - Suurin kuorman leveys, jota tällä tiellä voi kuljettaa.
    - Syötteenä hyväksytään positiiviset desimaaliluvut.
        - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
    - Syöte senttimetreissä.
    - Pakollinen tieto.
        - Oletus on 0, joka tarkoittaa määrittelemätöntä leveyttä.
    - Tieto tallennetaan tietokannan entiteettiin *Road*.
- Suurin kuorma
    - Suurin kuorma, jota tällä tiellä voi kuljettaa.
    - Syötteenä hyväksytään positiiviset desimaaliluvut.
        - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
    - Syöte tonneissa.
    - Pakollinen tieto.
        - Oletus on 0, joka tarkoittaa määrittelemätöntä kuormaa.
    - Tieto tallennetaan tietokannan entiteettiin *Road*.
- Korkein alikulkukorkeus
    - Suurin tien alikulkukorkeus.
    - Syötteenä hyväksytään positiiviset desimaaliluvut.
        - Syötteen oikeellisuus tarkistetaan heti kun mahdollista ja ilmoitetaan mahdollisesta virheestä.
    - Syöte senttimetreissä.
    - Pakollinen tieto.
        - Oletus on 0, joka tarkoittaa määrittelemätöntä korkeutta.
    - Tieto tallennetaan tietokannan entiteettiin *Road*.
- Selite

- Tietä kuvaava yksirivinen selite.
- Vapaavalintainen tieto.



4 TIEDOT JA TIETOKANTA

Sovellus käyttää tietojen tallennukseen Liferayn tietokantaentiteettejä ja niitä hallitsevia palvelurakentajalla generoituja palveluita. Alla on kuvattu tietokantojen taulut ja niiden yhteydet.



- Road
    - Sisältää tiestön tiedot.
    - Tietoja käytetään reittihakuun, kun tiestön ominaisuuksia tarvitaan.
- RoutePoint
    - Kokoelma pisteistä kartalla, jotka muodostavat kaikki tiet.
    - Tietoja käytetään reittihakuun, kun liikutaan eteenpäin.
    - Tien pisteen perusteella on mahdollista selvittää, mille tielle piste kuuluu.
- Transport
    - Sisältää kuljetuskaluston tiedot.
- Hoister
    - Sisältää nosturin tiedot.
- HoisterParam
    - Nosturin nostokyvyn tiedot.
    - Yhdellä nosturilla voi olla useampia parametreja.
- HoisterPOI
    - Määrittelee nosturin sijainnin kartalla.
    - Nosturia ei välttämättä ole asetettu kartalle, joten tämä voi olla tyhjä joillekin nostureille.
- Setting
    - Sisältää kaikki sovelluksen asetukset.
    - Pääsy kaikkialta WARin sisältä.
    - Tuki monelle portletin instanssille, jolle jokaiselle omat asetukset.

5 MUUT OMINAISUUDET

5.1 SUORITUSKYKY

Sovelluksen staattiselle suorituskyvylle on asetettu seuraavat vaatimukset:

- 5 samanaikaista käyttäjää.

5.2 TURVALLISUUS JA SUOJAUKSET

Sovellus käyttää Liferayn pääsynhallintaa ja käyttäjäntodennusta. Portaalin ylläpitäjä voi säätää käyttöoikeuksia ja rajoittaa pääsyä sovellukseen ja sen osiin.

5.3 JOUSTAVUUS

Sovelluksen reittihakua voidaan mukauttaa muuttamalla tiestöjen parametreja. Lisäksi voidaan määrittää halutut tiestön ominaisuudet, jota reittihaussa suositaan.

5.4 YLLÄPIDETTÄVYYS

Karttapohjan pystyy vaihtamaan ja päivittämään tarvittaessa ilman suuria ponnisteluja. Lisäksi uusien tiestöjen lisääminen tai jo olemassa olevien poistaminen kartasta onnistuu helposti. Tiestöissä on otettu huomioon myös mahdollisuus, jossa käyttäjä voisi itse muuttaa teiden sijainteja ja muita fyysisiä ominaisuuksia.

5.5 SIIRRETTÄVYYS / KONVERSIO

Sovellus valmistetaan siten, että se noudattaa Liferayn kehitysperiaatteita, jotta siirto onnistuu WAR-paketoituna Liferayn tukemiin ympäristöihin.

6 SUUNNITTELURAJOITTEET JA -PAKOTTEET

Sataman informaatiokeskus on Liferay-alustalla toimiva Javapohjainen portaali. Sovellus täytyy toteuttaa portlettina ja/tai servletinä järjestelmään. Sen ohjelmoidaan käyttäen Java-ohjelmointikieltä ja sen tulee noudattaa JSR 286 -portaalistandardia. Lopullinen sovellus WAR-paketoidaan, joka voidaan sitten asentaa Liferayn ohjauspaneelista ylläpitäjän tunnuksin. Laitteistorajoitteet ja muut ohjelmistorajoitteet ovat samat kuin Liferayssä. Lisäksi sovelluksen pitää käyttää Liferayn tarjoamia palveluja esimerkiksi tietokannan käsittelyyn.