# Holoninen moniagenttijärjestelmä tietämyskannalla

Pro gradu-työ
Turun yliopisto
Tietotekniikan laitos
2022
Feridun Akpinar

Tarkastajat:
Filip Ginter
Timo Knuutila

Turun yliopiston laatujärjestelmän mukaisesti tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck-järjestelmällä.

TURUN YLIOPISTO
Tietotekniikan laitos

FERIDUN AKPINAR: Holoninen moniagenttijärjestelmä tietämyskannalla

Pro gradu-työ, 60 s.
Marraskuu 2022

---

Tietämysgraafiin perustuva holoninen moniagenttijärjestelmä on moniagenttijärjestelmän variantti, jossa agentti korvataan itsesimilaarisella holonilla. Holonin holarkisen rakenteen ansiosta järjestelmä sopii parhaiten monimutkaisille moniagenttiongelmille. Tietämysgraafi on matemaattiseen graafiin perustuva tietämyskanta, joka säilyttää ja tarjoaa tietämystä järjestelmälle ilman että tietämystä tarvittaisiin sulattaa järjestelmän logiikkaan. Tämän opinnäytetyön tarkoitus on toteuttaa esimerkki järjestelmä holonisella moniagenttijärjestelmällä ja tietämysgraafilla. Opinnäytetyössä arvioidaan tietämysgraafiin perustuvien holonisten moniagenttijärjestelmien tulevaisuutta, niiden tarjoamia etuja ja heikkouksia. Tutkimuskysymysten vastaamiseksi suoritettiin kolme eri koetta, joissa kerättiin sekä syventävää että yleistietoa järjestelmän laajennettavuudesta ja käytettävyydestä. Järjestelmän laajentamisessa tarkkailtiin uusien ominaisuuksien lisäysprosessia ja sen vaikutusta muihin järjestelmän osiin. Käytettävyyskokeissa testattiin käytettävyyttä ja etsittiin poikkeamia muista järjestelmistä. Tietämysgraaafiin perustuvaa holonista moniagenttijärjestelmää on ominaisuuksiltaan helppo laajentaa sekä älyllisesti että rakenteellisesti. Holarkia helpottaa laajojen monimutkaisten ongelmien suunnittelua. Tietämysgraafi eristää tietämyksen järjestelmän koodista ja integroi sen suoraan järjestelmään erillisenä itsenäisenä komponenttina. Tällaisilla järjestelmillä on käyttöä ja tulevaisuutta etenkin laskennallisesti hajautettavien ongelmien ja moniagenttiongelmien parissa. Tietämysgraafiin perustuvien holonisten moniagenttijärjestelmien uskotaan yleistyvän tulevaisuudessa.

Asiasanat: Tietämyskanta, Holoninen moniagenttijärjestelmä, Holoni, Resurssien allokointijärjestelmä

# Sisällys

# Kuvat

# Taulukot

# Termistö

**CQL** Cypher Query Language

**FIPAACL** Foundation for Intelligent Physical Agents Agent Communication Language

**GBH** Goal-based holon

**HMAS** Holonic multi-agent system

**KQML** Knowledge Query and Manipulation Language

**MBRH** Model-based Reflex Holon

**NP-hard** Non-deterministic polynomial-time hardness

**RDF** Resource Description Framework

**REST** Representational State Transfer

**SRH** Simple Reflex Holon

**UBH** Utility-based Holon

# 1 Johdanto

Älykkäiden järjestelmien kehittämiseen etsitään yhä enemmän uusia ratkaisuja. Tekoälyn kehittyminen tällä vuosituhannella on edistänyt älykkäiden järjestelmien kehitystä merkittävästi. Tekoälyn tarjoamien erilaisten metodien, kuten koneoppimis-, syväoppimis- ja moniagenttijärjestelmän ansiosta älykkäisiin toimintoihin perustuvia järjestelmiä voidaan suunnitella, toteuttaa ja kehittää entistä helpommin. Tekoälystä onkin tulossa älykkäiden järjestelmien perusta tänä päivänä.

Holoninen moniagenttijärjestelmä tietämysgraafilla tarjoaa uudenlaisen lähestymiskulman hajautettujen älykkäiden järjestelmien kehittämisessä. Holoni käsitteeseen perustuva holoninen moniagenttijärjestelmä on yksi moniagenttijärjestelmän varianteista, jota on käytetty usein monimutkaisten järjestelmien kehittämisessä sen tarjoaman kätevän holonirakenteen ansiosta. Holonit tarjoavat uudenlaisen ratkaisun älykkäiden järjestelmien suunnittelemisessa ja kehittämisessä.

Tietämysgraafi on matemaattiseen graafiin perustuva tietämyskanta, joka tarjoaa koneellisesti luettavaa informaatiota. Älykkäät järjestelmät, jossa säilytetään tietämystä ja jossa tietämys käytetään järjestelmän itsenäiseen päättelyyn käyttävät usein tietämysgraafeja. Tietämysgraafin yksi suurista eduista on sen kyky integroida suurta määrää tietämystä järjestelmään.

## 1.1 Tavoitteet

Tässä tutkielmassa suoritetaan kokeellinen tutkimus, jossa rakennetaan holoninen moniagenttijärjestelmä tietämysgraafilla. Tutkielmassa vastataan seuraaviin kysymyksiin:

1. Mitä etuja tällaiset järjestelmät tarjoavat?
2. Mitkä ovat tällaisen järjestelmän heikkoudet?
3. Millainen tulevaisuus tämän tyyppisillä järjestelmillä on?

## 1.2 Tutkielman rakenne

Tutkielman toisessa kappaleessa esitellään tämän tutkielman keskeisintä aihetta holonista moniagenttijärjestelmää. Kolmannessa kappaleessa esitellään tietämysgraafia ja neljännessä kappaleessa tutkimusmenetelmät, joita on käytetty tässä työssä. Viidennessä kappaleessa määritellään kokeellista tutkimusta varten valittua resurssien allokointijärjestelmä, joka perustuu holonisen moniagenttijärjestelmään ja tietämysgraafiin. Kuudennessa kappaleessa toteutetaan kyseistä järjestelmää, testataan sen kaikkia osia ja suoritetaan kokeellista tutkimusta. Viimeisessä kappaleessa kootaan kaikki tutkimuksen eri vaiheista saadut tiedot yhteen vastaamalla tutkimuskysymyksiin ja antamalla suppean johtopäätöksen järjestelmästä.

# 2 Holoninen moniagenttijärjestelmä

Joukko agenteista koostuva hajautettu järjestelmä, jossa jokaisella agentilla on oma itsemääräämisoikeus, kyky toimia itsenäisesti ja kyky havaita ympäristönsä kutsutaan moniagenttijärjestelmäksi [1]. Moniagenttijärjestelmässä agentit ovat jakamattomia yksikköjä, jotka kykenevät kommunikointia käyttäen saavuttamaan tavoitteitaan tekemällä yhteistyötä muiden agenttien kanssa.

Perinteisissä moniagenttijärjestelmissä on puutteita ja rajoituksia, kuten esimerkiksi se, että agentteja pidetään oletuksena jakamattomina yksikköinä. Tämä lähestymistapa on aiheuttanut moniagenttijärjestelmän kehittäjille merkittäviä arkkitehtuurillisia ongelmia, sillä se ei ota kantaa miten muodostaa organisaatiota eli agenttiryhmiä valmiina olevista agenteista niin että ne toimisivat yhdessä ikään kuin ne olisivat yksittäisiä agentteja.

Holoninen moniagenttijärjestelmä (engl. Holonic multi-agent system) pyrki vastaamaan tähän ongelmaan korvaamalla agentit holoneilla. Toisin kuin perinteisissä moniagenttijärjestelmissä, holonisessa moniagenttijärjestelmässä järjestelmän perusyksikkö holoni ei ole jakamaton entiteetti vaan holarkisesta rakenteesta koostuva kokonaisuus. Holonisessa moniagenttijärjestelmässä holonit kykenevät muodostamaan atomisia organisaatioita ja näin ollen synnyttämään korkeamman tason entiteettejä matalamman tason entiteeteistä [2, s. 3].

## 2.1 Holoni

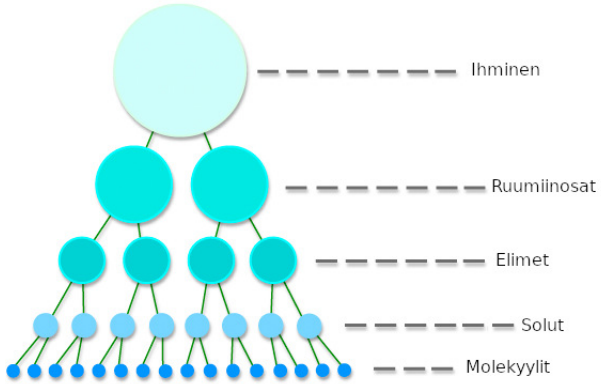
Holonisen moniagenttijärjestelmän keskeisin rakenneyksikkö on holoni. Termi holoni tulee kreikan kielen sanoista "holos"(kokonaan) ja "on"(osa). Sen alkuperäinen esittäjä on unkarilainen Arthur Koestler, joka käytti sitä kuvaamaan rekursiivisia itsesimilaarisia biologisia ja sosiaalisia rakenteita [3, s. 3]. Holoni on rakenteeltaan itsesimilaarinen rakenne, joka muodostuu muista holoneista [4, s. 3]. Holoni on siis ulkopuolelta katsottuna yksittäinen entiteetti, mutta todellisuudessa se voi olla muista entiteeteistä koostuva kokonaisuus.

Holonin ja agentin pääero pidetään siinä, että agentti on oletuksena jakamaton yksikkö, kun taas holoni voi olla muista holoneista koostuva kokonaisuus. Lisäksi holonilla on erilaisia asemia [5, s. 524]. Holoni voi esimerkiksi toimia edustajaholonin asemassa, jolloin se on vastuussa kaikista muista holoneista [6, s. 721]. Holoni voi kuulua myös useampaan holoniryhmään, jolloin sellaisesta käytetään nimitystä moniosainen holoni. Mulloin mikäli holoni kuuluu vain yhteen holoniryhmään, sitä kutsutaan yksiosaiseksi holoniksi.

## 2.2 Holonirakenteet

Holonit voivat muodostaa uusia holoneja lukuisilla eri holonirakenteilla. Holarkia on näistä kaikkein yleisin holonirakenne. Holarkia on hierarkkinen variantti, jossa ei ole välttämättä ylintä tai alinta entiteettiä. Holarkisen rakenteen suurimpiin etuihin kuuluu sen kyky sopeutua ympäristöön ja kesto ulko- ja sisäpuolisia häiriöitä vastaan. Holarkinen rakenne on myös tehokas tapa uudelleenkäyttää resursseja [7, s. 6].

Holarkisesta järjestäytymisestä on monenlaisia esimerkkejä, kuten esimerkiksi ihminen, joka koostuu erilaisia ruumiinosista ja ruumiinosat edelleen erilaisista soluista. Ulkopuolelta katsottuna ihminen on yksittäinen entiteetti, mutta todellisuudessa se on holarkisen rakenteen mukaan muodostunut kokonaisuus.




Kuva 2.1: Holarkia

## 2.3 Ominaisuudet

Holonisen moniagenttijärjestelmän yksi tärkeimmistä ominaisuuksista on sen holoneja yhdessä pitävät sitoutumissopimukset. Sitoutumissopimuksiin liittyy erilaiset tavoitteet, yhteiset intressit tai vaikkapa pelkästään hyödyllisyys, jolloin holonit ovat sitoutuneet yhteistyöhön ainoastaan hyötyäkseen toisista. Tähän voi liittyä lisäksi kuorman jakaminen tai suorastaan toisten holonien palvelujen käyttäminen. Mikä tahansa holonia voidaan määritellä ja esittää seuraavalla kaavalla, missä *Head* kuvaa holonin edustajaholonia, *Subholons* lapsiholoneja ja *Commitments* sitoutumissopimuksia.

h=(Head, Subholons, Commitments)

Holonisen moniagenttijärjestelmän toinen ominaisuus on sen holonien kyky muodostaa superholoneja [8, s. 6]. Superholoni on joukko holoneista muodostunut kokonaisuus, joka toimii ikään kuin se olisi yksittäinen holoni ja joka on ominaisuuksiltaan, palveluiltaan ja tavoitteiltaan ainutlaatuinen. Superholonia voi muodostaa monella eri organisoitumismallilla. Yksinkertaisin organisoitumismalli on muodostaa superholonia joukko autonomisilla holoneilla. Toinen organisoitumismalli on holonien yhdistyminen yhden superholonin alle, niin että ne luopuvat autonomiastaan.

Holonisen moniagenttijärjestelmän kolmas ominaisuus liittyy holonien dynaamiseen päättelyprosessiin. Päättelyprosessiin kuuluu yleisesti päättelyn aloittava osapuoli,

päättelyyn osallistuvat holonit ja adoptiomekanismi. Päättelyprosessi määrää miten holonit tekevät päätöksen ja millaisen päättelyjärjestyksen ne muodostavat. Yleisimmät adoptiomekanismit ovat monarkia, oligarkia, polyarkia ja apanarkia.

Monarkisessa päättelyprosessissa päättämisestä päättää ainoastaan yksi edustajaholoni. Koko päättelyprosessi on sidottu yhden edustajaholonin ympärille ja muilla holoneille ei ole jätetty minkäänlaista päätösoikeutta. Tällaisen päättelyprosessin ainoa hyvä puoli on tietoturva, sillä järjestelmän rakentajan ei tarvitse pitää huolta muusta kuin edustajaholonista.

Oligarkkinen päättelyprosessia eroaa monarkisesta päättelyprosessista sen edustajaholoniryhmällä. Joukko edustajaholoneille annetaan päätösoikeus niin että päättelyprosessi on ainoastaan näiden edustajaholonien vastuulla. Ero oligarkkisen päättelyprosessin ja monarkisen päättelyprosessin välillä on huomattava, sillä järjestelmän rakentaja joutuu luomaan päättelyalgoritmin, joka ottaa huomioon kaikkien äänioikeutettujen edustajaholonien äänet.

Polyarkisen päättelyprosessin pääidea on tarjota joustavaa päätöksentekoa. Polyarkisessa päättelyprosessissa päättämiseen osallistuu joukko ennaltamäärättyjä holoneja. Tämä menettelytapa eroaa edellisistä päättelyprosesseista sillä, että päättelyprosessiin voi osallistua myös muita kuin edustajaholoneja. Tämän päättelyprosessin toteuttaminen on järjestelmän kehittäjien kannalta kuitenkin hyvin hankalaa ja ongelmallista.

Viimeinen päättelyprosessi on apanarkia. Se on demokraattinen ja joustava päättelyprosessi, jossa päätöksen saa osallistua kaikki holonit riippumatta asemastaan. Tällöin jokainen holoni saa äänestysoikeuden vaikuttaa lopulliseen päätökseen päättelyprosessissa. Tämä päättelyprosessi suositaan sellaisten sovellusten kohdalla, joissa halutaan mahdollisimman tehokas päättelyprosessi.

# 3 Tietämysgraafi

Älykkäiden järjestelmien laajentuessa sekä arkkitehdit että muut kehitykseen osallistuvat kehittäjät alkavat törmätä erilaisiin ongelmiin. Näistä ongelmista yksi merkittävin on tietämysongelma, joka liittyy kasvavan määrän tietämyksen integroimiseen järjestelmään. Perinteinen tapa sulattaa tietämystä suoraan järjestelmään on todettu hyvin usein epätehokkaaksi ja monet järjestelmän kehittäjät ovatkin nykyään turvautuneet dynaamisiin tietämyksen integroitumismenetelmiin. Tietämysgraafi on matemaattiseen graafiin ja ontologiaan perustuva tietämyskanta, joka säilyttää ja tarjoaa sekä tietoa että tietämystä [9, s. 4]. Tietämysgraafi voidaan kuvata parhaiten "käytännöllisenä ja koneellisesti luettavana tapana esittää informaatiota"[10, s. 28].

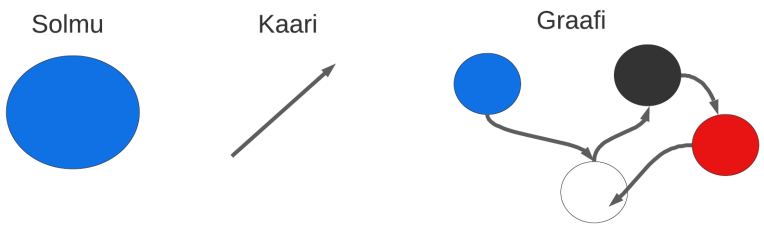
Data käsittelevissä järjestelmissä tietämystä käytetään tiedon etsimiseen, analysoimiseen ja oppimiseen. Kontekstuaalisen ymmärryksen tarve datasta on kiihdyttänyt tietämysgraafien tutkimista. Tietämysgraafeja on ollut markkinoilla vuodesta 1972, mutta niiden suosio alkoi kasvaa vasta sen jälkeen kun Googlen julkaisi Google Knowledge Graph-tietämyskannan vuonna 2011. Tietämysgraafien tärkein sovelluskohde on sellaiset tekoälyjärjestelmät, joissa tietämys on keskeisessä asemassa. Tällaiset järjestelmät tarvitsevat tietämystä datasta saadun informaation hyödyntämiseen. Toisin kuin perinteisissä järjestelmissä tietämysgraafeilla varustetut järjestelmät mahdollistavat poikkeuksellisen älykkäitä toiminnallisuuksia ja ominaisuuksia.

Tietämysgraafeja integroidaan järjestelmiin myös muihin tarkoituksiin kuin älykkäiden järjestelmien kehittämiseen. Tietämysgraafeja käytetään lääketieteessä [11],

lennonjohdoissa [12], neuroverkkojen kouluttamisessa [13] ja kyberturvallisuudessa [14]. Tietämysgraafit ovat tehokkaita ratkaisuja dynaamisten tietämysten integroitumismenetelmäksi.

## 3.1 Rakenne

Tietämysgraafissa pyritään hyödyntämään matemaattisen verkon työkaluja tietämyksen järjestämiseen, lukemiseen ja tallentamiseen. Tietämysgraafi koostuu solmuista ja kaarista. Se voi olla suuntamaton tai suunnattu, jolloin kaaret osoittavat yhdestä solmusta toiseen solmuun. Lisäksi tietämysgraafin ominaisuuksiin kuuluu nimellisyys ja nimeämättömyys [15, s. 5].

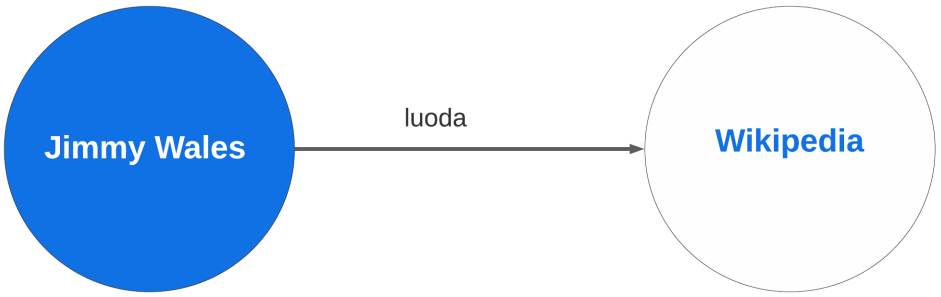



Kuva 3.1: Solmu, kaari ja graafi

Teoriassa tietämysgraafissa solmuja voi olla lukemattomia eri määriä, mutta käytännössä tietämysgraafin rajoituksiin kuuluu määrällisyysrajoite. Määrällisyysrajoite tarkoittaa että tietämysgraafissa tulee olla äärellinen määrä solmuja, jotta niitä pystytään digitaalisesti kirjoittamaan levylle.

Tietämysgraafissa solmut voivat edustaa mitä tahansa asiaa, mutta kaaret saavat edustaa ainoastaan solmujen välisiä suhteita. Jokainen kaari saa esittää ainoastaan kahden solmun välistä suhdetta antamalla suhteelle suunnan ja nimen. Kuvassa 3.2 esiintyy solmu nimeltä Jimmy Wales, josta lähtee kaari kohti Wikipedia-solmua. Kaaren nimellä *luoda* halutaan tarkoittaa, että Jimmy Wales on luonut Wikipedian.

Tietämysgraafit esiintyvät matemaattisesti suunnattujen graafien muodossa [16].




Kuva 3.2: Tietämysgraafin esimerkkisolmu ja kaari

Suunnatut graafit ovat tyypiltään sellaisia graafeja, joissa jokaisella kaarella on oma suuntansa ja kohteensa. Lisäksi tietämysgraafit ovat nimettyjä graafeja eli solmuilla tulee olla ainutlaatuiset tunnisteet tai nimet. Tietämysgraafin perusyksikkö on triple ja se esitetään subjekti, predikaatti ja objekti-kolmiona. Yllä olevassa kuvassa 3.2 esiintyy triple subjektilla Jimmy Wales, predikaatilla luoda ja objektilla Wikipedia.

## 3.2 Variantit

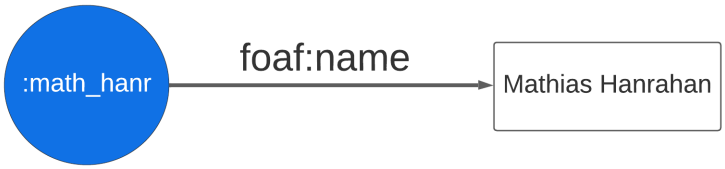
Tietämysgraafeja voidaan rakentaa käyttämällä erilaisia malleja. Yleisimmät mallit perustuvat RDF, Wikidata ja hyödyllisyysgraafi malleihin. Näistä kaikista RDF-malli on yleisin ja laajalle levinnein variantti, vaikka alun perin RDF kehitettiin ainoastaan *kehykseksi* kuvaamaan informaatiota internetissä. Johtuen sen laajalle levinneisyydestä ja ylellisyydestä monet tietämysgraafit ovat tänä päivänä mallinnettuja sen ympärille.

### 3.2.1 RDF

RDF (engl. Resource Description Framework) on World Wide Web Consortiumin (W3C) standardoima datamalli, joita käytetään julkaisemaan ja jakamaan jäsenneltyä dataa Webissä [17]. RDF-datamallin juuret ovat vuodessa 1996, jolloin haluttiin käyttöjärjestelmä riippumatonta ja toimittajaneutraalia metatietojärjestelmää. Myöhemmin 1999-luvulla W3C julkaisi ensimmäisen dokumentaation RDF-datamallista ja siitä lähtien se

on ollut runsaan kehityksen alla.

RDF on ylivoimaisesti suosituin malli tietämysgraafiyhteisöjen keskuudessa. Se on rakenteeltaan poikkeuksellisen yksinkertainen ja helposti ymmärrettävää malli, jonka käyttöönottaminen on hyvin helppoa sen runsaan dokumentaation ansiosta. RDF-mallin ydin on suunniteltu kolmikon nimisen lausekkeen ympärille. Kolmikko on RDF-graafin perusyksikkö, joka muodostuu subjektista, predikaatista ja objektista. Subjekti on solmu, josta lähtee kaari ja joka päättyy objektiin. Objekti on subjektin kohde ja kaari kolmikon predikaatti. Joukko kolmikoita kutsutaan RDF-graafiksi.




Kuva 3.3: RDF kolmikko

Kuvassa 3.3 on havainnollistettu esimerkki kolmikko. Kolmikon subjekti *:math_hanr* on käyttäjänimi, joka kuuluu käyttäjälle *Mathias Hanrahan*. Kolmikon objekti *Mathias Hanrahan* on literaali ja predikaatti *foaf:name* tulee lyhenteestä *a friend of friend* ja *name*. Foaf on standardoitu ontologia kuvaamaan ihmisiä ja heidän välisiä suhteita [18]. *Foaf:name*-predikaatti tarkoittaa, että objektissa on subjektin oikea nimi.

### 3.2.2 Wikidata

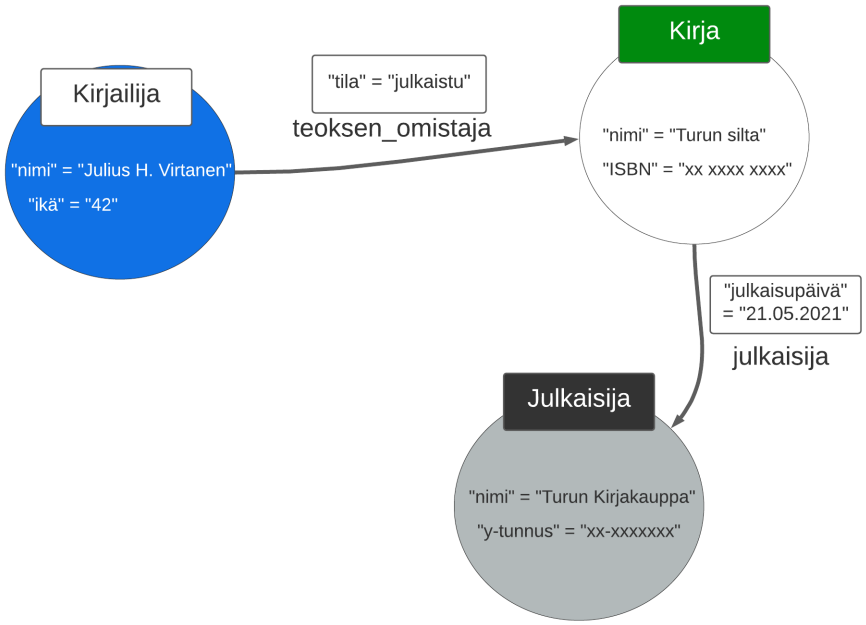
RDF-mallin tietämysgraafit eivät välttämättä ole paras ratkaisu kaikille tietämysgraafeille. Esimerkiksi Wikipedian tarpeisiin kehitetty Wikidata-tietämysgraafi on rakenteeltaan ja ominaisuuksiltaan täysin erilainen kuin RDF-mallin graafit. RDF-mallin tietämysgraafin ja Wikidata-mallin tietämysgraafin merkittävimmät erot ovat luokissa, literaaleissa, perimisessä, jäsenneltyyn datan esittämisessä, lähteiden merkitsemisessä ja informaation yksiköissä.

Wikidata käyttää luokkahierarkiaa, jossa ylin luokka on *class (Q16889133)*. Kaikki muut luokat ovat tämän class-luokan perijöitä. Literaalit ovat monipuolisempia ja ne mahdollistavat muun muassa intervallien ja monipuolisten yksiköiden syöttämistä järjestelmälle. Wikidatan lähteet sisällytetään predikaatteihin, joille on annettu oma erikoinen nimitys: lausekkeet. Lausekkeet sisältävät kaiken olennaisen tiedon subjektista ja predikaattien kohteista. Wikidatan ja RDF-mallin suurin ero on jäsennetyn datan esittämisessä. RDF-kolmikko vaatii ainoastaan subjektin, predikaatin ja objektin kahden solmun välisen suhteen esittämiseen, kun taas Wikidatalle tämä ei ole riittävä, sillä on olemassa sellaisia suhteita, joissa on enemmän kuin kaksi solmua. Yksi esimerkki tästä on näyttelijä, joka esiintyy jossakin elokuvassa tietyssä roolissa. RDF-kolmikko ei olisi riittävä tällaisen datan esittämiseen, sillä tiedon esittämiseen tarvittaisiin yhteensä kolme solmua: näyttelijä, elokuva ja rooli.

### 3.2.3 Hyödyllisyysgraafi

Kolmas merkittävä malli jota käytetään tietämysgraafien rakentamiseen on hyödyllisyysgraafi. Hyödyllisyysgraafi on matemaattisesti suunnattu ja nimetty moniverkko, jossa jokaisella verkon entiteetillä, mukaan lukien kaarilla, on avain-arvo ominaisuudet [19]. Hyödyllisyysgraafissa solmuihin ja kaariin kiinnitetään avain-arvo pareja rikastuttamaan graafin semantiikkaa. Tällainen graafi on yleensä joustava, helposti ylläpidettävä ja ominaisuuksiltaan tarpeeksi monimuotoinen tietämysgraafin pohjaksi.

Hyödyllisyysgraafin rakenneosat ovat solmu, kaari ja ominaisuus. Näistä rakenneosista erikoisin rakenneosa on ominaisuus, jota lisätään solmuihin ja kaariin avain-arvo muodossa. Se on tarkoitettu ohjaamaan ja syventämään graafiin järjestäytymisperiaatetta. Järjestäytymisperiaate on graafin ydinperiaate, jonka mukaan graafin sisältö järjestetään tiettyjen attribuuttien mukaan. Nämä attribuutit määräävät millainen graafi tulee syntymään, miten graafin solmut järjestetään ja millainen ontologia tulee muodostumaan.




Kuva 3.4: Esimerkki ominaisuugraafi

Kuvassa 3.4 esitetään hyödyllisyysgraafi kirjasta *Turun silta*. Tässä tietämysgraafissa esiintyvällä solmulla *Kirjailija* on ominaisuudet *nimi* ja *ikä*. Solmusta lähtee kaari solmuun *kirja* nimellä *teoksen_omistaja* ja ominaisuudella *tila*. Tässä hyödyllisyysgraafissa nähdään miten monimuotoisen informaation esittäminen muuttuu helpommaksi käyttämällä avain-arvo ominaisuuksia. Mikäli sama graafi toteutettaisiin RDF-mallilla, syntyisi lopputuloksena suuri määrä solmuja ja triple-kolmikoita.

## 3.3 Ominaisuudet

Sanalle tietämysgraafi ei löydy tarkkaa määritelmää. Yleisesti ottaen tietämysgraafia pidetään tietämyskantana, joka hyödyntää matemaattisen verkon ominaisuuksia. Matemaattisella verkolla ja sen ominaisuuksilla voidaan käsitteellistää, kuvata ja esittää monimutkaista ja epätäydellistä informaatiota helposti. Yksi tietämysgraafin merkittävistä ominaisuuksista on sen kyky tarjota datapainotteista tietämystä. Datapainotteisessa tietämyksessä tietämys lisätään järjestelmän dataan eikä itse koodiin.

Toinen merkittävä ominaisuus on järjestäytymisperiaate. Järjestäytymisperiaate on menetelmä, jolla luokitellaan ja järjestetään tietämysgraafin sisältöä. Järjestäytymisperiaate toimii sopimuksena tietämysgraafin rakentajan ja käyttäjän välillä, ja sillä määritellään myös miten tietämysgraafi on suunniteltu ja millainen rakenne sillä tulee olemaan. Järjestäytymisperiaate auttaa tietämysgraafin loppukäyttäjiä sen kontekstuaalisessa ymmärtämisessä.

Viimeisin merkittävä tietämysgraafin ominaisuus on sen tarjoama kyselykieli. Tietämysgraafin kyselykieliä käyttämällä voidaan syntaksikyselyjen lisäksi tehdä myös semanttisia kyselyjä. Semanttisissa kyselyissä hyödynnetään semanttista ja rakenteellista tietoa. Tämän lisäksi kyselyihin voidaan sisällyttää ontologista tietoa.

# 4 Tutkimusmenetelmät

Kappaleessa 1.1 on esitelty kaikki tutkimuksen tavoitteet. Työn olennaisin alkupainopiste pysyy esimerkkijärjestelmän kehittämisessä ja sen kvalitatiivisessa analysoinnissa. Järjestelmän analysoinnissa käytetään apuna laajennettavuutta ja käytettävyyttä. Laajennettavuudella tarkoitetaan järjestelmän kykyä laajentua mahdollisimman vähäisellä järjestelmän sisäisen rakenteen muutoksilla, silloin kun järjestelmään lisätään uusia toiminnallisuuksia [20]. Käytettävyys kertoo miten käyttäjä oppii käyttämään järjestelmää, valmistelemaan syötteitä ja ymmärtämään järjestelmän antamia viestejä [21]. Käytettävyydellä pyritään arvioimaan järjestelmän käytettävyyttä yleisesti ja tietämysgraafin käyttäjälle ja muille järjestelmän osille tuomia etuja ja haittoja. Lopullinen pääpaino pysyy siinä, että kerätään tietoa ja syvennetään tietämystä tällaisen järjestelmän käytettävyydestä.

Tämän työn lopullinen tavoite on vastata seuraaviin kysymyksiin:

- *Mitä etuja tällaiset järjestelmät tarjoavat?* - Kysymyksellä halutaan tuoda esiin hyödyt, kun järjestelmä perustuu holoniseen moniagenttijärjestelmään ja tietämysgraafiin.
- *Mitkä ovat tällaisen järjestelmän heikkoudet?* - Kysymykseen vastataan listaamalla rajoitteita, ongelmia ja yleisiä poikkeavuuksia.
- *Millainen tulevaisuus tämän tyyppisillä järjestelmillä on?* - Annetaan karkea arvio tällaisen järjestelmän tulevaisuudesta.

# 5 Resurssien allokointijärjestelmä

Tässä tutkielmassa rakennetaan HMAS-pohjainen resurssien allokointijärjestelmä tietämysgraafilla. Järjestelmä tulee olemaan suppea, mutta vaativia toimintoja salliva kokonaisuus. Kehityksen painopiste pidetään sellaisissa toiminnallisuuksissa missä korostuu holonisen moniagenttijärjestelmän ja tietämysgraafin ainutlaatuiset ominaisuudet. Muiden järjestelmän osien kehitysprioriteetti pidetään matalana.

Tässä luvussa käydään läpi rakennettavan järjestelmän yleinen määrittely, arkkitehtuuri, toiminnallisuudet ja ominaisuudet. Aliluvussa 5.1 määritellään resurssien allokointijärjestelmä ja sen keskeiset rakenneosat. Sitä seuraavassa aliluvussa 5.2 esitetään tehtäväkehys, jota käytetään tehtävien syöttämiseen järjestelmään. Aliluvussa 5.3 listataan holonien analysointikäyrät ja aliluvussa 5.4 järjestelmässä käytettävät algoritmit. Järjestelmän hallintapaneelista ja sen sisällöstä kerrotaan aliluvussa 5.5.

## 5.1 Yleinen määrittely

Resurssien allokointi on yksi keski- ja suurorganisaatioiden merkittävimpiä ongelmia. Ongelman tekee hankalaksi sen hajanainen luonne, jossa jokainen resurssi voi olla strategisessa asemassa ajamassa omia intressejä [22]. Ongelmalle ei yleensä löydy algoritmeja, joita voidaan todeta parhaaksi, sillä allokointiongelma on stokastinen ja siihen kuuluu paljon erilaisia inhimillisiä tekijöitä. Stokastisen luonteensa lisäksi jopa vaativammat optimointialgoritmit, kuten kolmiulotteiset pakkausongelma-algoritmit, joita voidaan hyödyntää allokoinnissa ovat NP-vaikeita (engl. non-deterministic polynomial-

time hardness) [23].

Tämän tutkielman resurssien allokointijärjestelmä on työvoimaresurssien jakamiseen tarkoitettu järjestelmä, jossa on holoneja, algoritmeja, tehtäviä, käyttäjiä ja allokointipyyntöjä. Järjestelmä tehostaa työvoimaresurssien allokointia, etujen maksimointia ja riskien vähentämistä. Järjestelmään voidaan lisätä tehtäviä ja määrätä ne työvoimaresursseille eri etujen mukaisesti. Resurssien allokointiongelma on yleinen moniagenttiongelma, sillä allokoinnissa resurssit toimivat kuin autonomiset etujaan ajavat agentit. Tässä tutkielmassa allokointijärjestelmä toteutetaan HMAS-pohjaisena moniagenttijärjestelmänä tietämyskannalla. Tietämysgraafia käytetään tietämyksen ja tietojen integroimiseen.

### 5.1.1 Resurssit

Tämän tutkielman HMAS-pohjaisessa resurssien allokointijärjestelmässä resurssilla tarkoitetaan työvoimaresurssia ja sellainen rekisteröidään järjestelmään holonimuodossa. Rekisteröitäessä työvoimaresurssia sille annetaan tyyppi, joka kertoo resurssin kategoriasta. Tämän jälkeen tietämysgraafiin kartoitetaan työvoimaresurssin osaamiset ja heikkoudet. Resurssin rekisteröinnin yhteydessä järjestelmään syötetään taulukon 5.1 mukaiset tiedot. Näihin tietoihin kuuluu tunnus, tyyppi, nimi, sukupuoli, työtunnit, ulkoisen rajapinnan osoite, ulkoisen rajapinnan avain, saatavuustieto, kuormitustieto, stressitieto, kustannustieto, ikä, kokemus vuodet ja saatavuus.

ID on yksilöllinen tunnus, jota käytetään työvoimaresurssin tunnistamiseen. Tunnus koostuu pelkästään numeroista ja eikä siinä saa olla mitään muita merkkejä. Tunnus on myös ensisijainen kommunikointitunniste yhden tai kahden työvoimaresurssin välisessä tiedonjaossa. Mikäli työvoimaresurssi poistetaan tai muutetaan, sen tunnus ei saisi enää määrätä toiselle holonille.

Tyyppi kertoo työvoimaresurssin tyypistä. Työvoimaresurssi voi olla tyypiltään työntekijä, ryhmä, organisaatio ja jopa ulkoinen palvelu. Allokointijärjestelmä käyttää työ-

| **Nimi** | **Kuvaus** |
|---|---|
| ID | Työvoimaresurssin yksilöity tunnus. |
| Type | Työvoimaresurssin tyyppi (esim. työntekijä, ulkoistettu_resurssi jne.) |
| Name | Työvoimaresurssin nimi. |
| Gender | Työvoimaresurssin sukupuoli mikäli kyseessä on työntekijä. |
| Working hours | Työvoimaresurssin keskimääräiset työtunnit päivässä. |
| Remote address | Holonin rajapinta-osoite mikäli se sijaitsee muualla. |
| API token | Muualla sijaitsevan holonin rajapinta-avain. |
| Availability data | Tietoa työvoimaresurssin saatavuudesta. |
| Load data | Tietoa työvoimaresurssin kuormittumisesta. |
| Stress data | Tietoa työvoimaresurssin stressistä. |
| Cost data | Tietoa työvoimaresurssin kustannuksesta. |
| Age | Työvoimaresurssin ikä mikäli kyseessä on työntekijä. |
| Experience Years | Työvoimaresurssin kokemus yhteensä vuosina. |
| Is available | Tietoa siitä että onko työvoimaresurssi saatavilla allokointia varten. |

Taulukko 5.1: Holonin tiedot

voimaresurssin tyyppiä allokoinnin suorittamisessa. Mikäli työvoimaresurssilla ei ole tyyppiä tiedossa, sille voidaan antaa arvoksi NA (engl. Not Available), joka tarkoittaa, että työvoimaresurssilla ei ole tyyppiä.

Työvoimaresurssin ikä ja sukupuoli ovat tarkoitettu sellaisille resursseille, joiden tyypiksi on ilmoitettu *työntekijä*. Nämä tiedot käytetään pääosittain helpottamaan suodatusta ja allokoinnin suorittamista. Järjestelmässä saattaa olla myös mukautettuja algoritmeja, jotka käyttävät ikää ja sukupuolta optimaalisten arvojen laskemiseen.

Jokaisella työvoimaresurssilla on sille määrättyjä päivittäisiä työntekoaikoja. Rekisteröitäessä työvoimaresurssia sille määrätään *daily_work_hours* niminen tietokenttä, johon syötetään työtuntien määärä ja jota lasketaan jakamalla viikoittaisten työtuntien määrä seitsemällä. Järjestelmän algoritmit hyödyntävät työtunteja allokoinnissa, resurssin saatavuuden laskemisessa ja tehtävien valinnassa. Työtuntien määrä tulee ilmoittaa kokonaislukuna eikä desimaalilukuna.

Luotaessa työvoimaresurssia sille tehdään automaattisesti tilaa tietämysgraafiin. Tietämysgraafiin voidaan lisätä tietoa työvoimaresurssin työkokemuksista, heikkouksista ja muista allokointiin vaikuttavista asioista. Työvoimaresurssin graafi näkyy sekä järjestelmän käyttäjille että muille työvoimaresursseille. Allokointijärjestelmä käyttää tietämysgraafia työvoimaresurssiin liittyvien tietojen ja osaamisen kartoittamiseen. Taulukossa 5.2 on listattu tietämysgraafin yleiset ontologialuokat.

Class-luokka on tietämysgraafin ontologiassa ylin mahdollinen luokka, josta periytyy kaikki muut luokat. Kaikki class-luokasta periytyvät solmut kuuluvat järjestelmän tietämystoimialueeseen. Mikäli tietämysgraafissa on muita kuin class-luokasta periytyviä solmuja, ne eivät yleensä liity järjestelmän tietämykseen vaan ovat jotain muita tietoja. Esimerkiksi työvoimaresursseihin liittyvät tiedot käyttävät solmuja, jotka eivät periydy class-luokasta.

Holon-luokka edustaa työvoimaresurssia ja Knowledge-luokka on kokemusta varten luotu luokka, jota käytetään osaamisen, teknologian, kirjaston tai muiden asioiden

| **Luokka** | **Kuvaus** |
|---|---|
| Class | Tietämysgraafin ontologian ylin luokka. |
| Holon | Työvoimaresurssia edustava luokka. |
| Knowledge | Osaamisen kohdetta edustava luokka. |
| Number | Luokka edustamaan numeerisia arvoja. |
| Issue | Sairaus, stressi tai jokin muu työntekoon vaikuttava asia. |
| Task | Tehtävää edustava luokka. |

Taulukko 5.2: Työvoimaresurssin ontologialuokat

esittämiseen. Number-luokka on kyselykielellä luotujen kyselyjen helpottamiseksi luotu numeriisia arvoja edustava luokka.

Issue-luokka on sairautta, estettä tai jokin muu työntekoa vaikeuttavan asian edustamiseksi luotu luokka. Issue eli ongelma on monen algoritmin kannalta välttämätön tieto riskin laskemisessa. Mikäli tehtävässä jätettäisiin tämä tieto huomioimatta, saattaa tehtävään pahimmillaan ilmestyä tavallista useammin stressaantuvia tai sairastuvia työvoimaresursseja.

Task-luokka on tehtävää, projektia tai mikä tahansa työvoimaresurssia vaativaa asiaa edustava luokka. Task-luokan solmut eroavat muista solmuista pakollisilla avain-arvo ominaisuuksilla. Niitä voidaan syöttää järjestelmään parantamaan tietämysgraafin ontologiaa. Pakolliset avain-arvot ovat seuraavat: task_type, task_priority. Ensimmäinen liittyy tehtävän tyyppiin ja jälkimmäinen tehtävän prioriteettiin.

### 5.1.2 Holonit

Resurssien allokointijärjestelmä on luonteeltaan holoninen moniagenttijärjestelmä. Järjestelmässä holoni edustaa työvoimaresurssia, sen ominaisuuksia ja toiminnallisuuksia. Järjestelmään on mahdollista luoda neljä erilaista holonityyppiä. Nämä holonityypit eroavat toisistaan merkittävällä tavalla. Eroavaisuudet ovat holonien rakenteissa, tavoit-

teissa, toiminnallisuuksissa ja käyttäytymisessä. Taulukossa 5.3 on listattu erityyppiset holonit kuvauksineen.

| **Luokka** | **Lyhyt Kuvaus** |
|---|---|
| Yksinkertainen refleksiholoni (SRH) | Refleksiseen toiminnallisuuteen perustuva holoni. |
| Mallipohjainen refleksiholoni (MBRH) | Holoni, jolla on sisäinen tila ja malli ympäristöstään. |
| Tavoitepohjainen holoni (GBH) | Holoni, jolla on sekä tavoite että malli ympäristöstään. |
| Hyödyllisyyspohjainen holoni (UBH) | Maksimaalista hyötyä tavoitteleva holoni. |

Taulukko 5.3: Eri tyyppiset holonit

Yksinkertaisella refleksiholonilla (engl. simple reflex holon) on yksinkertainen agenttiohjelma, joka perustuu nimensä mukaisesti refleksimalliin. Refleksimallissa on joukko ehto-toiminta pareja ja vertailija, jonka tehtävänä on vertailla reaalihavaintoja ehto-toiminta parien ehtoihin. Mikäli havainto toteuttaa jokin ehto-toiminta parin ehdon, SRH-agenttiohjelma toteuttaa sille parille kuuluvan toiminnan. Yksinkertaisen agenttiohjelmansa ansiosta SRH on erittäin tehokas holoni suorittamaan sille annettuja tehtäviä. Ehto-toiminta pareihin perustuvalla agenttiohjelmalla ei ole kuitenkaan sisäistä tilaa, ja tämä taas aiheuttaa sen että holoni ei kykene suorittamaan vaativia tehtäviä.

Mallipohjainen refleksiholoni (engl. model-based reflex holon) on refleksiholonia edistyneempi holoni, jolla on sisäinen tila. MBRH-agenttiohjelmalla on tarkka malli ympäristöstään ja se ylläpitää tilansa kahdella mallilla: siirtymämallilla ja anturimallilla. Siirtymämalli sisältää tietoa holonin ympäristöstä ja toimintojen vaikutuksista siihen. Anturimalli taas tarjoaa syvällisempää tietoa ja tietämystä havaintojen tulkkaamisesta ja käsittelemisestä. Anturimalli kykenee käsittelemään havaintosarjoja ja käyttämään ne johtopäätösten tekemiseen. MBRH-agenttiohjelman sisäinen tila mahdollistaa ha-

vaintojen tallentamisen ja uudelleenhyödyntämisen.

Tavoitepohjainen holoni (engl. goal-based holon) on yksi yleisimmistä holonityypeistä holonisissa moniagenttijärjestelmissä. GBH-agenttiohjelman ydin perustuu kirjaimellisesti päämäärään, jota se yrittää tavoittaa. Tällaisen holonin agenttiohjelma on suurin piirtein samanlainen kuin mallipohjaisella refleksiagentilla. Ero löytyy siinä, että jokaisen holonin valitsemaa toimenpide on vietävä holonia kohti sen päämäärää. Toisiin sanoen, jokainen toimenpide, jota holoni valitsee on edistettävä sen tavoitetta. Toimeenpiteen valitseminen saattaa olla parhaimmillaan triviaalia tai pahimmillaan paljon resursseja vievää. Agentti saattaa suunnitella tai etsiä oikeaa toimenpidettä hyvinkin pitkään. GBH-agenttiohjelmassa korostuu myös enemmän dynaamisuutta ja tämä näkyy siinä, että agenttiohjelma kykenee vaihtamaan tavoitteensa ajon aikana.

Hyödyllisyyspohjainen holoni (engl. utility-based agent) on GBH kaltainen holoni, jolla on myös hyödyllisyysfunktio. GBH-agenttiohjelmalla on yksi merkittävä heikkous mikä tekee siitä lähes käyttökelvottoman monimutkaisten tavoitteiden suorituksessa: sokea tavoitteeseen vievien toimenpiteiden valitseminen. Toisiin sanoen se saattaa päästää tavoitteeseen, mutta se saattaa myös kuluttaa liian paljon resursseja tai aiheuttamalla liikaa haittaa. UBH, joka on kaikista holoneista yleisin ja kehittynein malli ratkaisee tämän ongelman hyödyllisyyfunktiollaan. Hyödyllisyysfunktio maksimoi holonin intressejä vuorokauden jokaisena hetkenä valitsemalla ainoastaan ne toiminnot, jotka ovat optimaalisia. Holonin joutuessa tilanteeseen, jolloin sillä on useampaa toimintoa valittavana, hyödyllisyysfunktio punnitsee ja valitsee ainoastaan sen toiminnon, joka maksimoi sen edut.

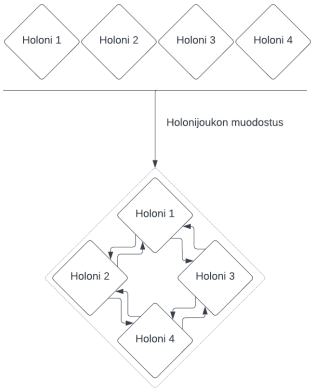
**Holonirakenteet**

Holoneilla on päättelyprosessin adoptiomekanismien lisäksi useampia erilaisia organisoitumismenetelmiä ja nämä eroavat toisistaan erittäin radikaalisti. Taulukossa 5.4 on listattu eri organisoitumismenetelmät.

| **Holonirakenne** | **Järjestäytymisperiaate** |
|---|---|
| Holonijoukko | Joukko holoneja yhdistyy yhden holonikaton alle säilyttämällä itsemääräämisoikeutensa. |
| Yhtenäistyneet holonit | Holonit yhdistyvät yhdeksi holoniksi luopumalla kokonaan itsemääräämisoikeudestaan. |
| Moderoitunut liitto | Holonit yhdistyvät yhdeksi holoniksi luovuttamalla osan itsemääräämisoikeudestaan yhdelle superholonille. |

Taulukko 5.4: Holonirakenteet

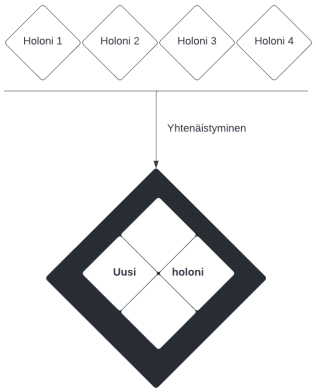
Joukko holoneja voivat yhdistyä yhdeksi holoniksi säilyttämällä autonomiansa ja tarjoamalla uusia toiminnallisuuksia ja ominaisuuksia. Tällöin uuden holonin toiminnallisuudet ja ominaisuudet ovat näiden holonien yhteystyöliitosta syntyviä kokonaisuuksia. Tällaista holonia, joka syntyy yhdistyneiden holonien liitosta kutsutaan konseptuaaliseksi holoniksi, sillä holonia ei itsessään ole kuin konseptuaalisesti. Kuvassa 5.1 on havainnollistettu holonijoukon muodostus.




Kuva 5.1: Holonijoukko

Yhtenäistyneet holonit ovat holonijoukkojen vastapää. Yhtenäistyneissä holoneissa holonit muodostuvat yhden uuden holonin luovuttaen sille kaikki autonomiansa. Tällöin syntyy täysin uusi holoni, joka peittää kaikki holonit yhden katon alle viemällä

heiltä itsemääräämisoikeutta. Uuden holonin liitto voidaan esittää matemaattisena kaavana $h = (\{A\}, \{A\}, C_{merge})$. Kaavan ensimmäinen A edustaa holonien liitosta syntynyttä uutta holonia. Toinen {A} edustaa lapsiholoneja, ja koska tällaisessa holonirakenteessa ei synny lapsiholoneja, myös lapsiholonin paikalle on laitettu uusi vasta syntynyt holoni. $C_{merge}$ kuvaa holonien välisiä sitoumuksia. Sitoumuksilla holonit sopivat ehdoista, säännöistä ja sopimuksista, jotka pitävät ne yhdessä. Kuvassa 5.2 havainnollistetaan holonien yhdistyminen yhdeksi holonoksi.

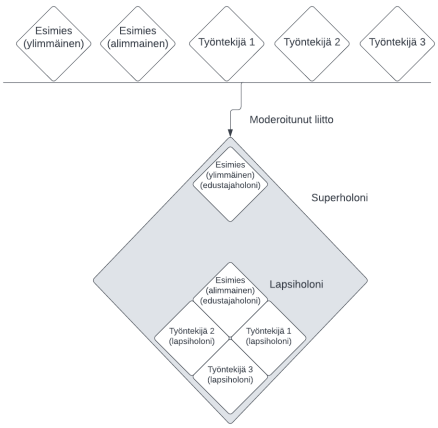



Kuva 5.2: Yhtenäistyneet holonit

Holonijoukko ja yhtenäistyneet holonit ovat holoniorganisoitumisen kaksi ääripäätä. Holoneja rakentaessa tulee usein tarve muodostaa räätälöityjä rakenteita. Tämä pätee varsinkin silloin, kun sitoutumiset muuttuvat monimutkaisemmaksi ja sitoutumisjoukkoon $C_{merge}$ ilmestyy uusia intressejä. Tällainen tapaus voidaan havainnollistaa esimies-esimies-työntekijät tapauksella, missä sekä ylimmällä että alimmalla esimiehellä on omat erilliset intressit. Työntekijöiden intressejä lisättäessä sitoutumisjoukko muuttuu entistä dynaamisemmaksi intressijoukoksi. Sekä holonijoukko että yhtenäistyneet holonit eivät enää pystyisi palvelemaan tällaista tapausta, sillä ensimmäinen vaihtoehto jättäisi esimiesten intressit huomioimatta, kun taas jälkimmäinen työntekijöiden intressejä.

Ainoa toimiva ratkaisu esimies-esimies-työntekijät tapaukseen olisi moderoitunut

liitto. Moderoituneessa liitossa holonit muodostavat liiton valitsemalla yhden edustajaholonin ja asettamalla kaikki muut holonit lapsiholoneiksi. Edustajaholoni on kaikkien lapsiholonien yhteinen edustaja, joka vastaa kommunikoinnista ja määräyksistä. Tällöin syntyy superholoni, jonka edustajaholonina toimisi ylimmäinen esimies ja lapsiholoneina alimmainen esimies ja kaikki työntekijät. Kuvassa 6.3 on havainnollistettu moderioituneesta liitosta syntyvä superholoni.




Kuva 5.3: Moderioitunut liitto

Moderoituneen liiton yhdistyneet holonit luovuttavat ainoastaan osan autonomiastaan edustajaholonille. Tämä johtaa siihen, että tarvittaessa holonit voivat vaihtaa tyypikseen moniosaiseksi holoniksi ja liittyä myös toisiin holoneihin. Lisäksi moderoituneessa liitossa holonit saavat pitää kommunikointinsa auki antamalla ulkopuolisten holonien hyödyntää niiden toiminnallisuuksia, palveluja ja ominaisuuksia.

### Holonien välinen kommunikointi

Moniagenttijärjestelmäyhteisö on tähän asti tuottanut kaksi tärkeää kommunikointikieltä holonien tai agenttien väliseen kommunikointiin. Nämä molemmat kommunikointikielet ovat KQML (engl. Knowledge Query and Manipulation Language) ja FIPA ACL (engl. Foundation for Intelligent Physical Agents Agent Communication Language ). Molemmissa kommunikointikielissä on sekä hyviä että huonoja puolia. Esimerkiksi KQML

on protokollasta ja sisällöstä riippumaton vaihtoehto, kun taas FIPA ACL on helppokäyttöinen ja helposti luettava kommunikointikieli. Molempien kommunikointikielten huonot puolet ovat semantiikkaan ja keskusteluihin liittyvissä puutteissa ja ongelmissa [24]. Tässä järjestelmässä käytetään FIPA ACL kaltaista räätälöityä kommunikointikieltä, johon sisällytetään taulukossa 5.5 esiteltävät parametrit.

| **Parametri** | **Kuvaus** |
|---|---|
| Sender | Viestin lähettäjäholonin yksilöllinen tunnus. |
| Type | Viestin tyyppi. |
| Content | Tähän parametriin sisällytetään viesti. |
| Ontology | Ontologia, jota viesti koskee. Mikäli ontologialuokkia on useampia, niitä täytyy ilmoittaa taulukossa. |
| Receiver | Viestin saajan yksilöllinen tunnus. Mikäli saajia on useampia, ilmoitetaan tunnukset taulukossa. |
| Conversation ID | Järjestelmän generoima tunnus keskustelulle. |

Taulukko 5.5: Holoniviesti

Sender-parametriin sijoitetaan lähettäjäholonin tunnus. Type-parametriin taas voidaan kirjoittaa seuraavat tyypit: default, reply-to-holon ja reply-to-conversation. Default arvolla viestin tyypiksi ilmoitetaan olevan tavallinen viesti. Reply-to-holon-tyypin viesti on vastaus toiselle holonille ja reply-to-conversation on vastaus keskustelulle, jossa voi olla mukana useampia holoneja. Viestin content-parametri sisältää viestin sisällön ja ontology-parametri määrää mitä ontologialuokkaa viesti koskee. Receiver-parametri voi saada arvokseen *all* tai saajaholonin tunnuksen. Mikäli arvoksi valitaan *all*, viesti lähtee kaikille holoneille. Tämä ominaisuus on varattu ainoastaan erilaisten ilmoitusten näyttämiselle. Conversation ID-parametri on järjestelmän keskustelulle varamaa tunnus, jota ilmoitetaan kun viestin tyypiksi valitaan reply-to-conversation. Tämä ominaisuus on hyvin kätevä ryhmäkeskusteluille, sillä holonin ei enää tarvitse säilyttää holonien

tunnukset keskustelun aikana.

## 5.2 Tehtävät

Epäsäännölliset tehtävät ovat resurssien allokointijärjestelmän yksi suurimpia ongelmia. Allokointijärjestelmään saattaa ilmestyä kaikentyyppisiä tehtäviä, jotka vaihtelevat tunteja kestävistä tehtävistä kokonaisiin projekteihin. Osa tehtävistä saattaa sisältää myös erikoisvaatimuksia, kuten esimerkiksi sellaista tietoa, että ainoastaan tiettyjä vaatimuksia täyttävä työvoimaresurssi eli holoni saa osallistua tehtävän suorittamiseen. Algoritmit eivät kuitenkaan kykene käsittelemään tällaisia tehtäviä ilman standardeja.

Tehtävien esikäsittelyprosessissa tehtävät muunnetaan yhteensopivaksi syöttämällä ne ensin järjestelmän lomakkeelle. Esikäsittelyprosessissa tehtävistä tehdään lukukelpoisia ja yhteensopivia sekä itse järjestelmälle että järjestelmän algoritmeille. Tehtävien esikäsittelyprosessiin kuuluu neljä eri vaihetta: tehtävän tarkistus, tehtävän puhdistus, tehtävän syöttäminen lomakkeeseen ja tehtävän korjaaminen. Tehtävän tarkistukseen kuuluu tehtävän oikeellisuuden tarkistus. Siinä varmistetaan, että tehtävä kelpaa työvoimaresursseille eli holoneille. Tehtävän puhdistusvaiheessa tehtävän virheelliset parametrit korjataan ja kelpaamattomat parametrit korvataan annettujen ohjeiden mukaisesti. Kolmannessa vaiheessa tehtävä syötetään järjestelmän lomakkeelle ja viimeisessä vaiheessa täydennetään puuttuvat tiedot.

### 5.2.1 Parametrit

Resurssien allokointijärjestelmän tehtävät koostuvat taulukon 5.6 mukaisesti seitsemästä eri parametrista. Jokaisen järjestelmään syötettävän tehtävän on sisältävä nämä parametrit järjestelmän ohjeiden mukaisesti, muussa tapauksessa niistä tulee kelvottomia allokointioperaatioiden suorituksissa. Osa parametreista koskee tehtävän yleistietoja ja loput on varattu laskennallisuudelle. Tehtäviin voidaan lisätä myös mukautettuja

parametreja, mutta algoritmit eivät välttämättä kuitenkaan kykene hyödyntämään ne.

| **Parametri** | **Kuvaus** | **Tyyppi** |
|---|---|---|
| *Id* | Tehtävän tunnus. | Teksti |
| *Type* | Tehtävän tyyppi. | Teksti |
| *Is completed* | Tehtävän valmius. | Teksti |
| *Name* | Tehtävän nimi. | Teksti |
| *Description* | Tehtävän kuvaus. | Teksti |
| *Estimated time* | Tehtävän arvioitu valmistusaika tunteina. | Teksti |
| **Knowledge tags** | Tehtävän suorittamiseen vaadittava osaaminen, teknologia ja kirjasto. | Tägit. |
| **Resource Demand Table** | Tehtävän suorittamiseen vaadittavat resurssit vaatimuksineen: resurssin tyyppi, resursin kokemusvuosien vähimmäismäärä ja resurssin osaamiset. | Taulu |
| *Priority* | Tehtävän prioriteetti. | 1-5 (2 on matala ja 5 korkea) |
| *Start date* | Päivä, jolloin tehtävä alkaa. | dd.mm.YYYY |
| *Due date* | Tehtävän eräpäivä eli päivä, jolloin tehtävän on oltava viimeistään valmis. | dd.mm.YYYY |
| **Assigned to** | Tehtävän suorittamiseen valitut holonit eli työvoimaresurssit. | Luonnollinen numero |

Taulukko 5.6: Tehtävän parametrit

Tehtävän nimi tulee kuvata tehtävää mahdollisimman selkeästi ja yksinkertaisesti. Pitkiä ja epäselviä nimiä tulee välttää, jotta käyttäjät eivät haaskaisi aikaa tehtävän selvittämiseen. Tehtävän aloitus- ja eräpäivät koskevat tehtävän aikaisinta aloituspäivää ja viimeisintä päivää, jolloin tehtävä on oltava valmis. Tehtävän kuvauksessa tehtävä kuvataan mahdollisimman selkeästi kaikkine vaiheineen. Tehtävän kuvaus tulee olla sellainen, että lukija ymmärtää mitä se koskee ja mitä sen suorittaminen vaatii.

Tehtävän prioriteetti kuvaa tehtävän tärkeysastetta. Tärkeysaste on yhdestä viiteen annettava numero, jolla kuvataan tehtävän tärkeyttä. Tärkeysasteet ovat none (1), low (2), middle (3), high (4) ja important (5). None-tärkeysaste edustaa sellaisia tehtäviä, joiden tärkeysaste ei ole määritetty. Tärkeysaste low edustaa matalaa tärkeysastetta. Tärkeysasteella high on toiseksi korkein prioriteetti important-tärkeysasteen jälkeen. Important-tärkeysasteen saaneita tehtäviä on suoritettava ennen muita tehtäviä.

Jokaista tehtävää lisättäessä on pystyttävä arvioimaan tehtävään kulutettavaa aika. Tehtävään kulutettava aika on tehtävän kokonaiseen suorittamiseen arvioitu aika, jota ilmoitetaan tunteina ilman erikoismerkkejä. Mikäli tehtävä on tyypiltään sellainen, että se on toistaiseksi voimassa oleva tehtävä, sen kenttä on jätettävä tyhjäksi.

Resurssien kysyntätaulukko on kaikkein tärkein parametri. Se on taulu, jossa ilmoitetaan minimaaliset resurssit, joita on valittava tehtävälle. Taulussa on kolme kolumnia: resurssin tyyppi (type), resurssin kokemusvuosien vähimmäismäärä (experience_years) ja resurssin osaaminen (knowledge_tags). Esimerkiksi rivi *employee, 2, [sql, rest]* tarkoittaisi, että tehtävälle on valittava ainakin yksi työntekijä, jolla on vähintään kahden vuoden kokemus ja osaaminen sql-kielestä ja rest-rajapinnasta.

## 5.3 Holonien analysointikäyrät

Honeille määrätään erilaisia käyriä, joita käytetään niiden tarkkailuun ja analysointiin. Käyrät liittyvät holonin saatavuuteen, kuormitukseen, stressiin ja kustannukseen. Sekä käyttäjät itse että holonit käyttävät näitä käyriä analysointiin ja johtopäätösten tekemiseen. Analysointikäyrät kertovat missä tilassa holonit ovat kyseisellä hetkellä.

Saatavuuskäyrällä tarkkaillaan holonin saatavuutta. Saatavuuskäyrä näyttää holonin saatavuuden annetulla hetkellä. Se on reaaliaikainen jatkuva funktio, jota voi hallita ainoastaan holoni itse. Saatavuus ilmoitetaan tunteina per päivä, niin että jos käyrältä poimitaan nykyhetken y-arvo, sen pitäisi palauttaa nykyhetken saatavuuden prosentteina. Saatavuuskäyrän x-akseli taas edustaa vuoden päiviä, ja x-akselin 0-arvo edustaa holonin rekisteröintipäivää.

Kuormituskäyrän on holonien kuormituksen analysointiin luotu käyrä, jota käytetään valvomaan tehtävien aiheuttamia kuormia holonille. Kuormituskäyrän funktion y-arvon ollessa 0 (nolla) kuormittuminen tulkataan olevan mitätön eli holoni suorittaa tehtäviä alkuperäisen suunnitelman mukaisesti. Mikäli kuormitusarvo alkaa nousta, silloin tehtävät alkavat kuormittaa holonia, jolloin tehtävät vaativat alkuperäissuunni-

telmaa enemmän resursseja. Kuormituskäyrän saavutettaessa maksimaalisen arvon 1 (yksi) kuormitus aiheuttaa holonin jäätymisen, mikä johtaa siihen, että holoni ei enää kykene suorittamaan muita tehtäviä. Kuormituskäyrän voi säätää ainoastaan käyttäjä itse.

Stressikäyrä määrätään niille holoneille, joiden tyyppi on työntekijä. Käyrällä pyritään valvomaan holonin vointia ja kykyä suoriutua sille määrätyistä tehtävistä. Stressikäyrän arvon ollessa 0 (nolla) holoni kykenee jatkamaan alkuperäissuunnitelman mukaisesti ilman että tehtävien suorittamisesta aiheutuisi sille ongelmia. Stressikäyrän arvon noustessa yhteen (1) holoni jäätyy totaalisesti aiheuttaen sen, että holoni ei enää kykene suorittamaan uusia tehtäviä. Stressikäyrä tuottaa erittäin tärkeää dataa työvoimaresurssien analysoinnissa.

Kustannuskäyrä on kustannuksen arviointiin ja analysointiin varten kehitetty käyrä. Kustannuskäyrä käytetään mittaamaan holonin sopivuutta tehtävien suorittamiseen. Kustannuskäyrä lasketaan yhdistämällä kuormituskäyrä ja stressikäyrää ja laskemalle näille keskiarvon. Se on funktio, jonka matemaattinen kaava on $k(x) = ((\frac{s(x)+ku(x)}{2}))$, jos stressikäyrän s(x) ja kuormituskäyrän ku(x) arvot pysyvät nollan (0) ja yhden (1) välissä.

$$k(x) = \begin{cases} k(x) = \frac{s(x)+ku(x)}{2}, & if \;\; 0 \leq s(x) \leq 1 \;\; and \;\; 0 \leq ku(x) \leq 1 \\ 1, & \text{otherwise} \end{cases} \tag{5.1}$$

Kustannuskäyrän alin arvo 0 (nolla) vastaa kustannuksen merkitsemättömyyttä, kun taas arvo 1 (yksi) tarkoittaa holonin olevan todella huono vaihtoehto mille tahansa tehtävälle. Algoritmit käyttävät kustannuskäyrää holonin kustannuksen selvittämiseen allokointioperaation aikana. Järjestelmän ylläpitäjät taas käyttävät sen ennakoimaan epäonnistuvia holoneja, tehtäviä ja projekteja. Esimerkiksi superholonin kustannuskäyrän kallistuessa jatkuvasti riittävän pitkällä aikavälillä, järjestelmä hälyttää asiasta ylläpitäjälle. Järjestelmä laskee ja päivittää kustannuskäyrän automaattisesti allokointioperaatioiden aikana. Mikäli holoni haluaa käyttää mukautettua kustannuskäyrää,

holonin ylläpitäjän on ilmoitettava asiasta järjestelmän ylläpitäjälle.

## 5.4 Algoritmit

Resurssien allokointijärjestelmä käyttää allokointiholonia resurssien allokointiin. Allokointiholoni on rakenteeltaan holonijoukko ja tyypiltään yksinkertainen refleksiholoni. Se koostuu rajapintaholonista, ydinholonista, allokointialgoritmiholonista ja piippuholonista. Sen tehtävänä on vastaa järjestelmän resurssien allokoinnista ja tehtävien ja allokointialgoritmien tarjoamisesta.

Rajapintaholoni on allokointiholonin kaikkien lapsiholonien edustajaholoni. Rajapintaholonin tehtäviin kuuluu ulkopuolelta tulevien viestin vastaanottaminen ja tulkkaaminen, lapsiholonien hallitseminen ja allokointioperaatioiden suorittaminen. Ydinholoni vastaa allokointioperaatioiden vastaanottamisesta, ajamisesta ja tulosten palauttamisesta rajapintaholonille. Piippuholonin tehtävänä on noutaa tehtävät, pitää lajiteltua listaa tehtävistä ja välittää ne ytimelle allokointia varten.

Allokointialgoritmiholoni on tyypiltään hyödyllisyyspohjainen holoni, jonka tehtäviin kuuluu allokointialgoritmien rekisteröiminen, testaaminen, välittäminen ja poistaminen. Järjestelmään saa lisätä allokointialgoritmeja ainoastaan rekisteröimällä ne ensin allokointialgoritmiholoniin. Tämä holoni testaa vasta rekisteröityjä allokointialgoritmeja suorittamalla rajapintatestejä. Sen jälkeen se merkitsee testejä läpäisseet allokointialgoritmit valmiiksi käytettäväksi.

### 5.4.1 Allokointialgoritmit

Allokointialgoritmi on resurssien allokointia suorittava algoritmi. Järjestelmä tarjoaa oletuksena kolme eri allokointialgoritmia: find-optimal-algoritmi, find-maximum-execution-algoritmi ja find-best-match-algoritmi. Allokointijärjestelmään on erikseen mahdollista rekisteröidä myös mukautettuja allokointialgoritmeja.

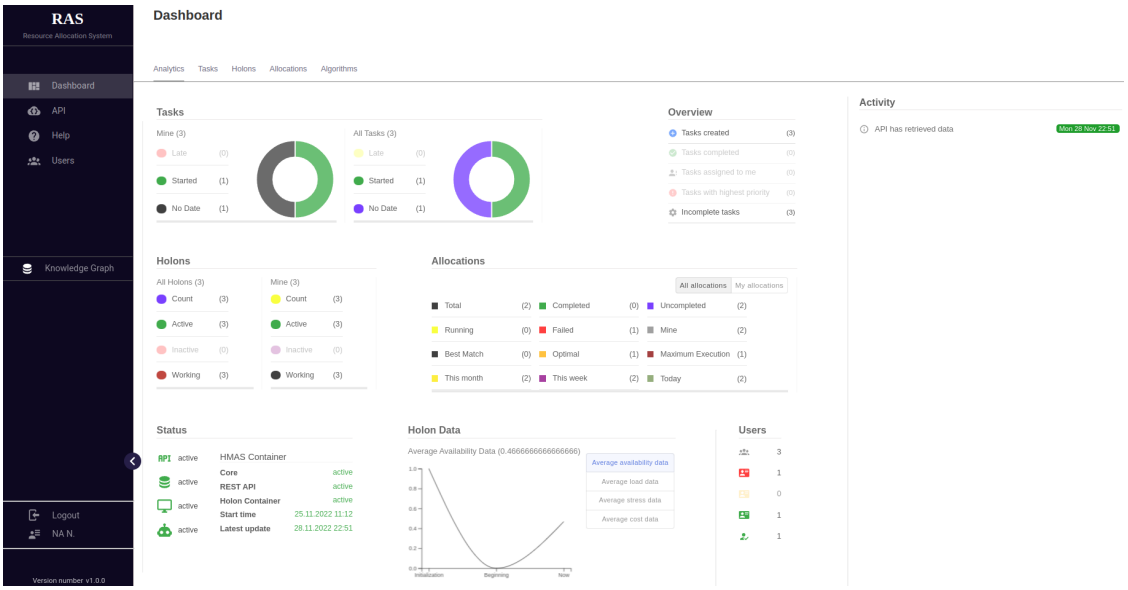
Ensimmäinen ja kaikkein vaativin allokointialgoritmi on find-optimal-algoritmi. Se on optimaalisimman allokoinnin löytämistä varten kehitetty ahne algoritmi, joka etsii jokaiselle tehtävälle parhaat holonit ottamalla huomioon tehtävien ja resurssien kaikki parametrit. Algoritmin toiminta on abstraktiotasolla hyvin yksinkertainen, se etsii jokaisella kierroksella parhaan mahdollisen holonin kyseiselle tehtävälle, kunnes tehtävän kaikki vaatimukset täyttyvät. Algoritmin tavoitteena on löytää oikeita ja sopivia holoneja jokaiselle tehtävälle.

Toinen allokointialgoritmi on find-maximum-execution-algoritmi. Se on tyypiltään ahne allokointialgoritmi, mutta optimoinnin sijaan sen päämääränä on maksimoida suoritettavien tehtävien määrää. Find-maximum-execution-algoritmi priorisoi suoritettavien tehtävien määrää keskittymällä pieniin ja lyhytkestoisiin tehtäviin ja näille holonien etsimiseen ensisijaisesti. Tällainen toimintatapa ei löydä optimaalisinta ratkaisua, mutta sillä pystytään maksimoimaan suoritettavien tehtävien määrää.

Kolmas ja viimeinen järjestelmän tarjoama allokointialgoritmi on find-best-match-algoritmi. Tämä allokointialgoritmi on tyypiltään ahne algoritmi, joka etsii kaikille tehtäville parhaat mahdolliset holonit. Algoritmi jakaa ensiksi holonit korkeamman prioriteetin tehtäville, jonka jälkeen se siirtyy alemman prioriteetin tehtäviin. Tämä allokointialgoritmi on teholtaan erittäin epätehokas ja sen ainoa hyöty on sen tarjoamassa laadussa.

## 5.5 Hallintapaneeli

Järjestelmä tarjoaa lähes kaikki toiminnallisuudet hallintapaneelin kautta. Hallintapaneelissa voidaan selata ja hallita käyttäjiä, tehtäviä, holoneja ja algoritmeja. Näiden perustoiminnallisuuksien lisäksi hallintapaneeli tarjoaa työkaluja holonien saatavuus-, kuormitus-, stressi- ja kustannustietojen päivittämiseen. Analyysiä varten hallintapaneelissa on analysointinäkymä, joita seuraamalla käyttäjä voi lukea analyysitietoja käyttäjistä, holoneista ja tehtävistä.




Kuva 5.4: Hallintapaneeli

### 5.5.1 Hallintapaneelin rakenne

Hallintapaneeli koostuu sivuista Dashboard, API, Help, Users ja Account. Dashboard on hallintapaneelin kojelautasivu ja sen tehtäväalueeseen kuuluu Analytics, Tasks, Holons, Allocations ja Algorithms näkymien tarjoaminen. Analytics-näkymä tarjoaa tietoa käyttäjän tehtävistä, holoneista ja toiminnasta. Näkymässä on myös analyysitietoa holonien analysointikäyristä, käyttäjistä ja järjestelmän komponenttien tiloista. Tasks-näymässä voidaan esikatsella ja hallita tehtäviä ja allocations-näkymässä käyttäjä voi hallita allokaatioita ja katsella vanhojen allokaatiopyyntöjen tuloksia. Algorithms-näkymä listaa järjestelmään rekisteröityjä algoritmeja.

Holons-näkymässä käyttäjä voi hallita holoneja ja niiden tiloja. Näkymä tarjoaa työkaluja analysointikäyrien päivittämiseen ja holonien yleistietojen muokkaamiseen, poistamiseen ja jäädyttämiseen. API-sivussa on järjestelmän REST-rajapinnan dokumentaatio. Sivussa on tietoa järjestelmän rajapinnasta ja sen päätepisteistä mikäli käyttäjä haluaa käyttää järjestelmää REST-rajapinnalla. Help-sivussa annetaan ohjeita järjestelmästä yleisesti ja sen käyttämisestä. Users-sivu on varattu käyttäjien hallintapaneeliksi

ja Account-sivu oman käyttäjätietojen esikatselemiseen ja päivittämiseen.

# 6 Kokeellinen tutkimus

Tämän työn kokeellisen tutkimuksen piiriin kuuluu resurssien allokointijärjestelmän toteuttaminen, testaaminen ja analysoiminen. Resurssien allokointijärjestelmän toteuttaminen alkaa arkkitehtuurin kehittämisellä. Arkkitehtuurin kehitysprosessi on jaettu neljään eri osavaiheeseen: arkkitehtoniset vaatimukset, arkkitehtuurin suunnittelu, arkkitehtuurin dokumentointi ja arkkitehtuurin evaluointi. Työn toisessa vaiheessa luodaan REST-rajapinta osavaiheilla standardointi ja toteutus.

Kolmannessa vaiheessa syvennytään järjestelmän Backend-osion suunnitteluun ja toteuttamiseen. Postgresql tietokannan suunnittelu ja toteutus tapahtuu neljännessa vaiheessa. Tietämyskanta suunnitellaan viidennessä vaiheessa ja sen toteutukseen käytetään Neo4j-graafitietokantaa. Kuudennessa vaiheessa luodaan allokointialgoritmit. Järjestelmän toteutuksen viimeisessä vaiheessa luodaan järjestelmän hallintapaneeli käyttämällä React-kirjastoa.

Järjestelmä testataan yksikkö- ja integraatiotesteillä. Testejä kirjoitetaan REST rajapinnalle, hallintapaneelille, Backend-osiolle ja järjestelmän allokointialgoritmeille. Näiden jälkeen aloitetaan laajennettavuuskokeet, jotka koskevat arkkitehtuuria, holoneja ja tietämyskantaa. Laajennettavuuskokeita seuraa käytettävyyskokeet, joissa testaan järjestelmän käytettävyyttä ottamalla huomioon tietämyskantaa.

## 6.1 Suunnittelu ja toteutus

### 6.1.1 Arkkitehtuuri

#### Arkkitehtoniset vaatimukset

Ensimmäiseksi arkkitehtoninen vaatimus on järjestelmän hajautettavuus. Vaatimus perustellaan sillä, että osa algoritmeista tai holoneista saattaa sijaita muualla kuin alkuperäisessä ympäristössä. Hajautettu järjestelmä on merkittävä arkkitehtoninen ajuri, sillä se yleensä määrää arkkitehtuurin lopullisen rakenteen.

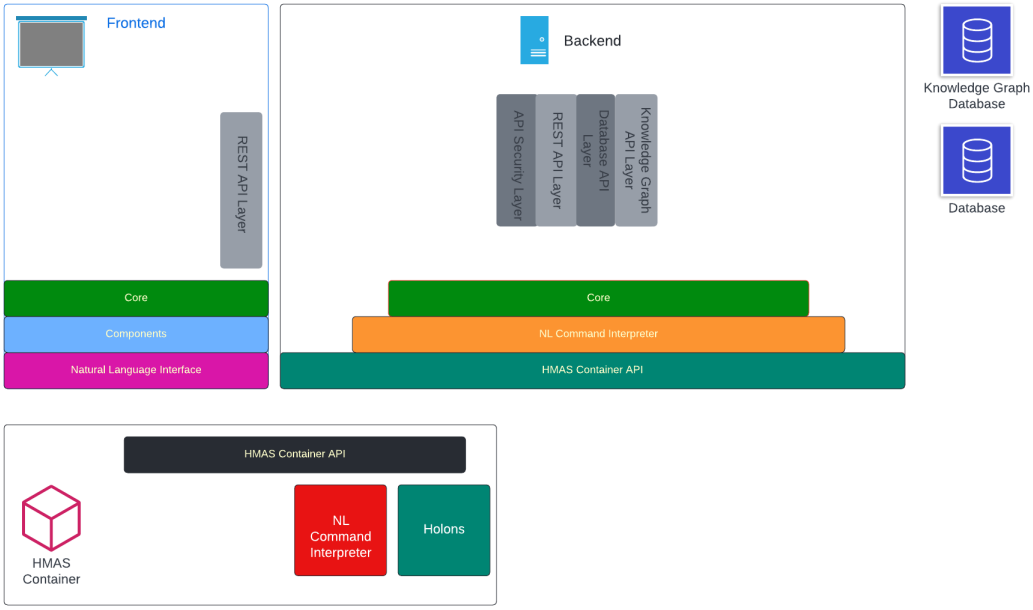
Toinen arkkitehtoninen vaatimus on tilattomuus, joka perustuu siihen, että järjestelmä käyttää REST-rajapintaa ja tilattomuus kuuluu REST (engl. representational state transfer) arkkitehtuurin rajoituksiin [25]. Vaatimus luo arkkitehtonista ajuria, joka panostaa tilattomuuden arkkitehtuurin suunnittelussa.

Kolmas arkkitehtoninen vaatimus koskee laajennettavuutta. Tällä pyritään korostamaan laajennettavuuden merkitystä arkkitehtuurin suunnittelussa. Järjestelmä saattaa laajentua sekä sisäisesti funktiotasolla että ulkoisesti komponenttitasolla, jonka takia arkkitehtuurin olisi hyvä taata mahdollisimman joustava järjestelmä.

#### Arkkitehtuurin suunnittelu, dokumentointi ja evaluointi

Kuvassa 6.1 esiintyvä järjestelmän arkkitehtuuridiagrammi sisältää seuraavia arkkitehtuurin komponentteja: Frontend, Backend, Knowledge Graph Database, Database ja HMAS Container. Frontend-komponentti on React-kirjastoon perustuva hallintapaneeli, jota toteutetaan web-sovelluksena. Sovelluksen kuuluu core (ydin), React-komponentit ja REST-rajapinta kirjasto, jolla se ottaa yhteyttä palvelimeen. Frontend-komponentti on suunniteltu niin että laajennukset voi toteuttaa suoraan sisäisinä React-komponentteina.

HMAS Container-komponentti on holoninen moniagenttijärjestelmä, jossa sijaitsee kaikki järjestelmän holonit. Komponentin vastuuseen kuuluu holonisen moniagenttijär-




Kuva 6.1: Resurssien allokointijärjestelmän arkkitehtuuri

jestelmän lisäksi allokointioperaatioiden toteuttaminen. Knowledge Graph Database-komponentti edustaa Neo4j-tietämyskantaa ja Database-komponentti järjestelmän PostgreSQL-relaatiotietokantaa.

Backend-komponenttiin piiriin kuuluu järjestelmän ohjauslogiikkaa ja REST-rajapinta. Komponentin REST-rajapinta tarjoaa kaikki järjestelmän toiminnallisuudet erilaisilla päätepisteillä. Backend-komponentti on suunniteltu mahdollisimman joustavaksi ja sen ydintä on mahdollista laajentaa vaivattomasti ilman arkkitehtuurisia muutoksia. Komponentin suunnittelussa on otettu huomioon REST-rajapinnan tilattomuus.

### 6.1.2 Yleinen REST-rajapinta

#### Suunnittelu

REST-rajapinta on toteutettu REST-käytäntöjen mukaisesti ottamalla huomioon seuraavat REST-standardin rajoitukset: yhtenäinen rajapinta, tilattomuus, välimuistiin tallentavuus, asiakas-palvelin erotus ja kerrostettu rakenne. Päätepisteiden nimet on

pidetty yksinkertaisina ja itse kuvailevina. Rajapinnan palauttamat viestit saapuvat rajapinnan käyttäjälle JSON:API-standardien mukaisesti [26].

#### Standardointi ja toteutus

Päätepisteiden nimeämisessä on noudatettu yhtenäinen rajapinta-rajoituksen ohjeita. Ensin järjestelmän toiminnallisuudet on listattu ja lajiteltu erilaisten konseptien alle ja sen jälkeen on luotu pääpisteet, joita on nimetty näiden konseptien mukaan. Päätepisteiden nimet ovat mahdollisimman itse kuvailevia ja ytimekkäitä. Taulukossa 6.1 on rajapinnan päätepisteet ja niiden kuvaukset. Liitteestä A löytyy REST-rajapinnan dokumentaatio.

Päätepisteitä on yhteensä kymmenen kappaletta. Ensimmäinen päätepiste /holons pitää sisällään holoneihin liittyvät toiminnallisuudet. Päätepiste /allocations on varattu allokointioperaatioiden hallintaan ja kyselyihin liittyviin operaatioihin. Päätepiste /tasks tarjoaa operaatioita, jotka liittyvät tehtäviin ja niiden hallintaan. Päätepiste /users tarjoaa käyttäjiin liittyviä operaatioita ja algorithm-päätepiste on vastaavasti varattu algoritmien hallintaan ja ajamiseen.

Settings-päätepiste on varattu hallintapaneelin yleisiä asetuksia varten. Päätepiste /auth tarjoaa todennusoperaatioita, joita käyttäjä ja järjestelmän HMAS Container-komponentti tarvitsee sisään kirjautumiseen. Status-päätepiste palauttaa tietoa järjestelmän ja sen komponenttien tiloista. Viimeinen päätepiste /search on luotu monimutkaisia päivitysoperaatiota ja hakukyselyjä varten.

### 6.1.3 Backend

Resurssien allokointijärjestelmän Backend-osion vastuuseen kuuluu järjestelmän yleisen REST-rajapinnan palvelujen tarjoaminen. Järjestelmän keskeisimpänä komponenttina sen käyttäjät ovat HMAS Container-komponentti, REST API-käyttäjät ja hallintapaneeli. Se on toteutettu Node.js alustalla ja Express-kirjastolla. Backend-osio on suorassa

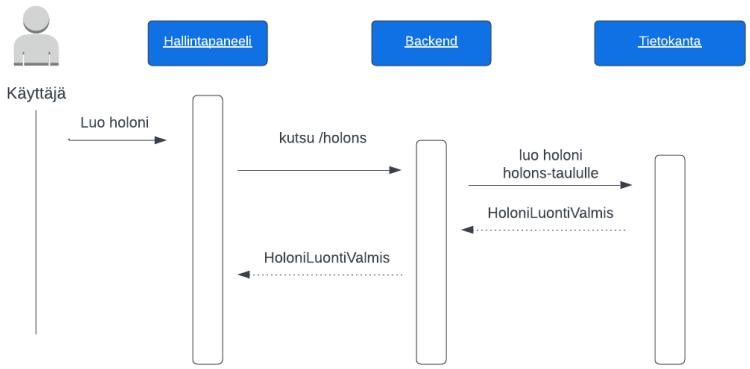
| **Päätepiste** | **Kuvaus** |
|---|---|
| /holons | Holonien hallintaan ja kyselyihin liittyvät kyselyt. |
| /allocations | Allokointioperaatioiden hallintaan ja kyselyihin liittyvät kyselyt. |
| /tasks | Tehtävien hallintaan ja kyselyihin liittyvät kyselyt. |
| /users | Käyttäjien hallintaan ja kyselyihin liittyvät kyselyt. |
| /algorithms | Algoritmien hallintaan ja kyselyihin liittyvät kyselyt. |
| /settings | Järjestelmän asetuksien hallintaa varten tehty päätepiste. |
| /auth | Todennuspalveluiden tarjoamiseen tehty päätepiste. |
| /status | Tarjoaa tietoa järjestelmän ja sen osien tiloista. |
| /search | Monimutkaisten hakukyselyjen tekemistä varten tehty päätepiste. |

Taulukko 6.1: REST-rajapinnan päätepisteviittaustaulu

yhteydessä relaatiotietokantaan, tietämysgraafiin ja HMAS Container-komponenttiin.

Backend-osion tarjoama yleinen REST-rajapinta toimii ohjelmistosiltana järjestelmän kaikkien käyttäjien ja osien välillä. Hallintapaneeli käyttää relaatiotetokantaa ja HMAS Container-komponenttia Backend-osion kautta. HMAS Container-komponentti taas hakee jonossa olevia allokointioperaatioita Backend-osion REST-rajapinnan avulla. Myös hallintapaneeli perustuu Backend-osion REST-rajapintaan.

Kuvassa 6.2 on esimerkki sekvenssidiagrammi holonin luontioperaatiosta ja sen eri vaiheista järjestelmässä. Holonin luonti alkaa hallintapaneelissa, mikä etenee sen jälkeen Backend-osiolle. Backend-osio tarkistaa holonin tiedot ja luo uuden holonin tietokantaan. Tämän jälkeen Backend-osio palauttaa viestin hallintapaneelille, jossa ilmoitetaan että holoni on luotu onnistuneesti tai virhe on tapahtunut luontioperaatiossa. Sekvenssidiagrammista näkee, että Backend-osio toimii siltana hallintapaneelin ja tietokannan välillä.




Kuva 6.2: Holonin luonti - sekvenssidiagrammi

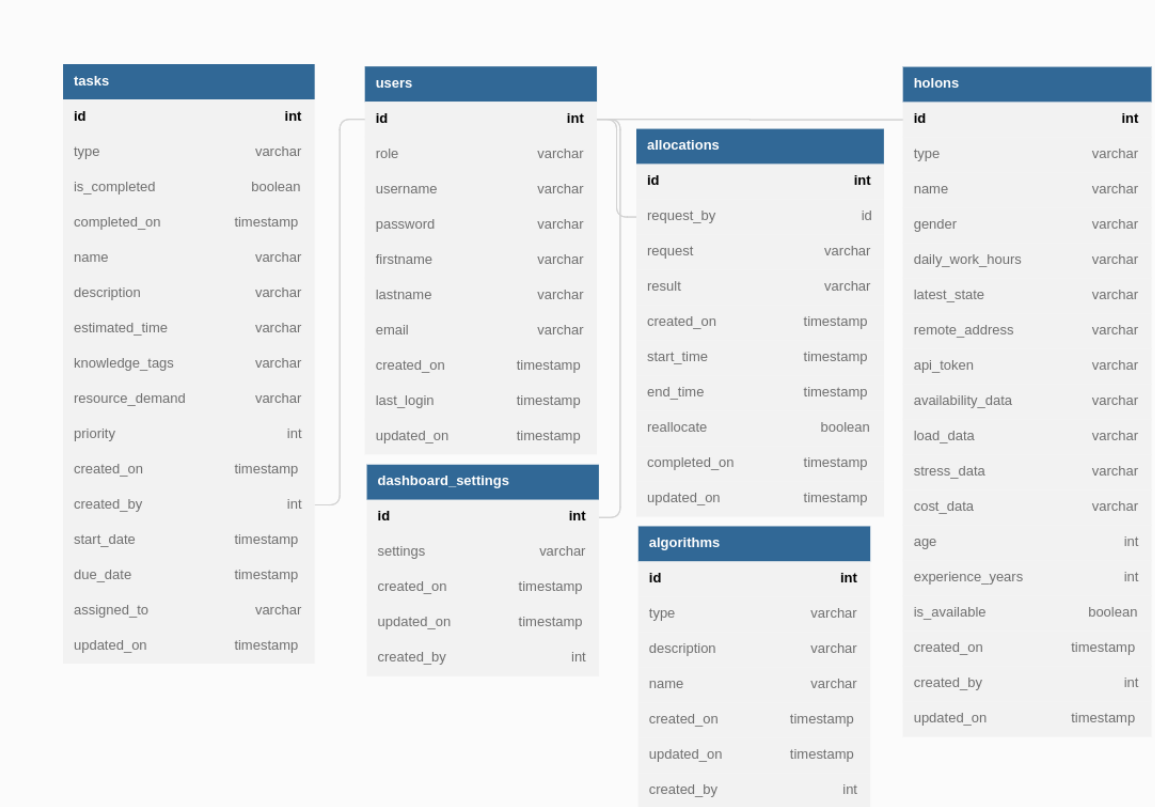
### 6.1.4 Tietokanta

Relaatiotietokanta on rakennettu palvelemaan kaikki järjestelmän vaatimuksia ja tarpeita. Tietokannassa on tauluja algoritmeille, allokointioperaatioille, asetuksille, holoneille, algoritmeille, tehtäville ja käyttäjille. Tietokannan toteuttamisessa on käytetty PostgreSQL-tietokantaa ja sillä on yhteensä kuusi eri taulua: tasks, users, holons, algorithms, dashboard_settings ja allocations. Kuvassa 6.3 esitetään resurssien allokointijärjestelmän lopullinen tietokantakaavio.

### 6.1.5 Tietämysgraafi

Resurssien allokointijärjestelmän tietämysgraafin rakentamisessa käytetään Neo4j-graafitietokantaa. Neo4j on yleinen graafitietokantaa, jolla voidaan luoda myös tietämysgraafeja. Tietämysgraafin luonnissa on käytetty taulun 6.2 ontologialuokkia ja yleisiä tietämysgraafin kehittämisessä käytettäviä käytäntöjä. Luotaessa uutta käyttäjää Backend-osio luo käyttäjän sekä relaatiotietokantaan että Neo4j-graafitietokantan hallintajärjestelmään.

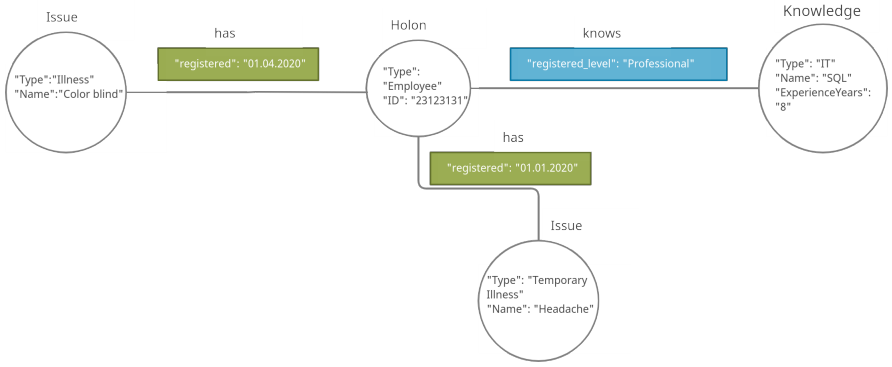
Hallintapaneelissa on linkki, jolla pääsee tietämysgraafin hallintajärjestelmään. Hallintajärjestelmään pääsee kirjautumaan sisään omalla käyttäjänimellä ja salasanalla. Hallintajärjestelmässä käyttäjä voi luoda sekä uusia solmuja että laajentaa olemassa




Kuva 6.3: Tietokantakaaviot

olevia graafeja. Kuvan 6.4 esimerkki näyttää holonia ja sen suhteita. Kuvassa esiintyvä holoni on rekisteröity ontologialuokalle Holon kahdella avain-arvo parilla. Holonilla on kolme eri suhdetta muihin solmuihin, joista ensimmäinen liittyy tietämykseen ja kaksi jälkimmäistä sairauteen. Tietämykseen liittyvällä solmulla on kolme ja sairaussolmuilla kaksi avain-arvo paria.

Toisin kuin relaatiotietokannassa tietämysgraafilla ei ole tietokantakaaviota ja tietämysgraafissa esiintyvät solmut eivät välttämättä säilytä samoja avain-arvo pareja, vaikka kuuluisivat samalle ontologialuokalle. Rekisteröitymättömiä ja ennestään havaitsemattomia ontologialuokkia saattaa myös esiintyä graafissa. Toisiin sanoen tietämysgraafin kasvattamiselle ei ole ennestään määrättyjä ehtoja ja rajoituksia.



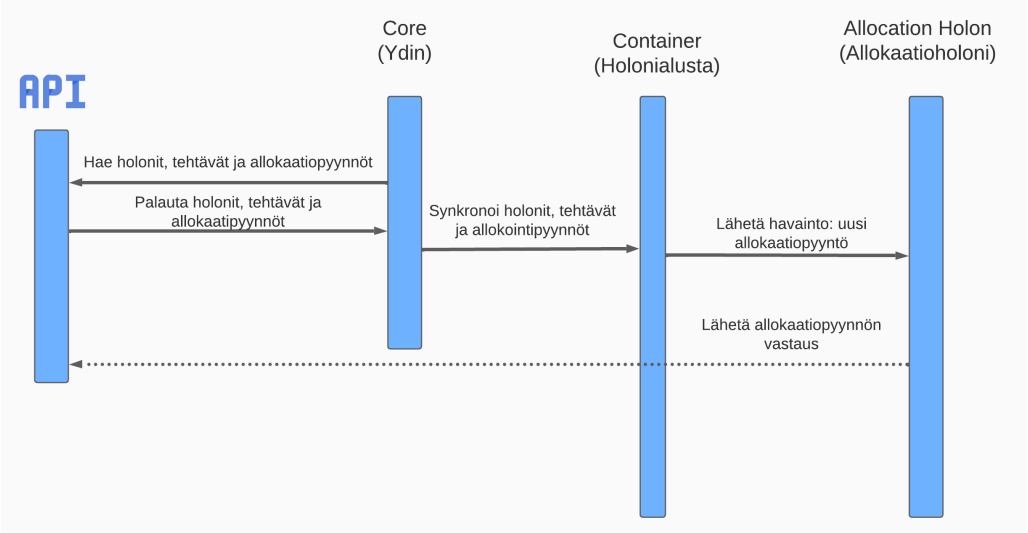

Kuva 6.4: Tietämysgraafi

### 6.1.6 HMAS Container

HMAS Container on muista järjestelmän osioista eristetty erillinen komponentti, jossa on toteutettu holoninen moniagenttijärjestelmä. Komponentti on jaettu Core, Container ja API moduuleihin. Core-moduulin tehtävänä on hakea holonit, tehtävät ja allokointioperaatiot Backend-osiosta ja jakaa ne Container-moduulille. API-moduuli tarjoaa REST-rajapinnan, jossa on status-päätepiste komponentin tilan selvittämiseen.

Container-moduuli on JavaScript-kielellä kirjoitettu holonialusta, jonka tehtäviin kuuluu holonien luominen, ajaminen ja näille ympäristön tarjoaminen. Moduuli tarjoaa kaikki välttämättömät ominaisuudet ja toiminnallisuudet moniagenttijärjestelmän simuloimiseen. Moduuli on suoraan kiinni API ja Core moduuleissa ja siinä sijaitsee myös allokointiholoni, joka suorittaa allokointioperaatioita ja palauttaa ne Backend-osiolle.

HMAS Container-komponentti on riippuvainen järjestelmän tietämysgraafista ja Backend-osiosta. Komponentin Container-moduulin holonit hakevat tietämysgraafista itseensä liittyviä graafeja tulevia allokaatiooperaatioita varten. Komponentin Core-moduuli käyttää Backend-osiota holonien, tehtävien ja allokointioperaatioden hakemiseen. Komponentti on rakennettu itsenäiseksi ja riippumattomaksi järjestelmän osioksi ajamaan holoneja ja suorittamaan allokointioperaatioita.




Kuva 6.5: Allokaatiopyynnön sekvenssikaavio HMAS Container-komponentissa

Kuvassa 6.5 on havainnollistettu HMAS Container-komponentin sisäinen allokaatioprosessi. Core-moduuli hakee allokaation, holonit ja tehtävät API-moduulin avulla Backend-osiosta. Tämän jälkeen se välittää ne Container-moduulissa sijaitsevalle holonialustalle. Holonialusta välittää allokaatiopyynnön allokaatioholonille ja tämä suorittaa allokaation annetuilla parametreilla. Allokaation valmistuessa allokaatioholoni lähettää allokaatiotuloksen suoraan takaisin Backend-osiolle API-moduulin kautta.

### 6.1.7 Algoritmit

Järjestelmän allokointiholonilla tarjoaa oletuksena kolme allokointialgoritmia: find-optimal-algoritmi, find-maximum-execution-algoritmi ja find-best-match-algoritmi. Nämä allokointialgoritmien toteutukset perustuvat aliluvussa 5.4.1 kuvattuihin algoritmeihin. Mikäli käyttäjä haluaa lisätä omia allokointialgoritmeja hänen on päivitettävä allokointiholonia. Allokointiholonin päivittäminen taas vaatii HMAS Container-komponentin uudelleenkäynnistämistä.

Allokointiholonin toiminta on erittäin yksinkertainen. Sen lapsiholoni nimeltä rajapintaholoni vastaanottaa Core-moduulista allokointipyyntöjä, jonka jälkeen se vä-

littää ne toiselle lapsiholonille nimeltä ydinholoni. Ydinholoni suorittaa allokoinnin annetuilla holoneilla, tehtävillä ja algoritmilla, jonka jälkeen se luovuttaa tuloksen takaisin rajapintaholonille. Rajapintaholoni taas lähettää allokointioperaation tuloksen Backend-osiolle. Backend-osio tallentaa allokointioperaation tuloksen tietokantaan ja merkitsee allokaatiopyynnön suoritetuksi.

### 6.1.8 Web-sovellus

Järjestelmän web-sovellus on toteutettu React-kirjastolla komponenttipohjaisella strategialla. Sovelluksessa on alikappaleen 5.5.1 mukaan neljä sivua: Dashboard, API, Help, Users ja Account. Kaikki sivut on toteutettu itsenäisinä React-komponentteina. Järjestelmän perustoiminnallisuudet sijaitsevat Dashboard-sivussa, jossa käyttäjä voi seurata järjestelmän analyyttisiä tietoja ja hallita holoneja, tehtäviä ja allokointipyyntöjä. Käyttäjä voi samassa sivussa listata HMAS Container-komponentiin rekisteröityjä algoritmeja ja lukea niiden kuvauksia.

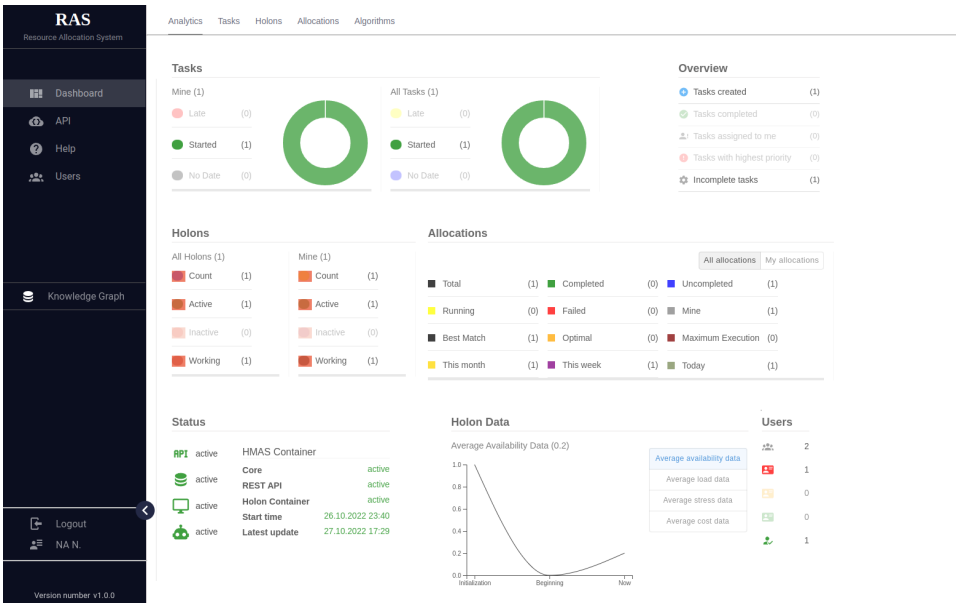
#### Käyttäjät

Web-sovelluksen käyttäjät on jaettu kolmeen eri kategoriaan: user, moderator ja admin. User-käyttäjät ovat järjestelmän käyttäjiä eli työntekijöitä, joilla ei ole lupaa hallita muita kuin omia holoneja ja tietoja. Moderator-käyttäjät ovat oikeutettuja hallitsemaan sekä omia että user-kategorian käyttäjien tietoja ja holoneja. Admin-kategoria on varattu järjestelmän ylläpitäjille. Admin-kategorian käyttäjät ovat oikeutettuja hallitsemaan kaikkia käyttäjiä.

#### Dashboard

Dashboard-sivu on kojelauta, jossa on Analytics, Holon, Tasks, Algorithms ja Allocations näkymät. Analytics-näkymä tarjoaa analyyttisiä tietoja tehtävistä, holoneista, allokointipyynnöistä, järjestelmän komponenttien tiloista ja holonien analysointikäyristä.

Näkymässä voi myös lukea luotujen käyttäjien määrää ja niiden tyypt. Kuvassa 6.6 on kuvakaappaus Analytics-näkymästä.

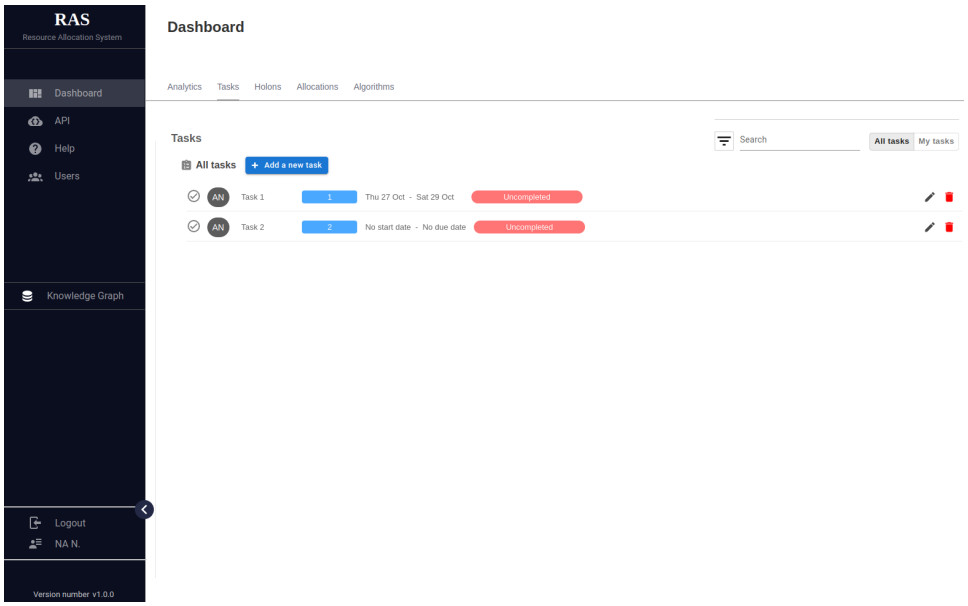



Kuva 6.6: Analytics-näkymä

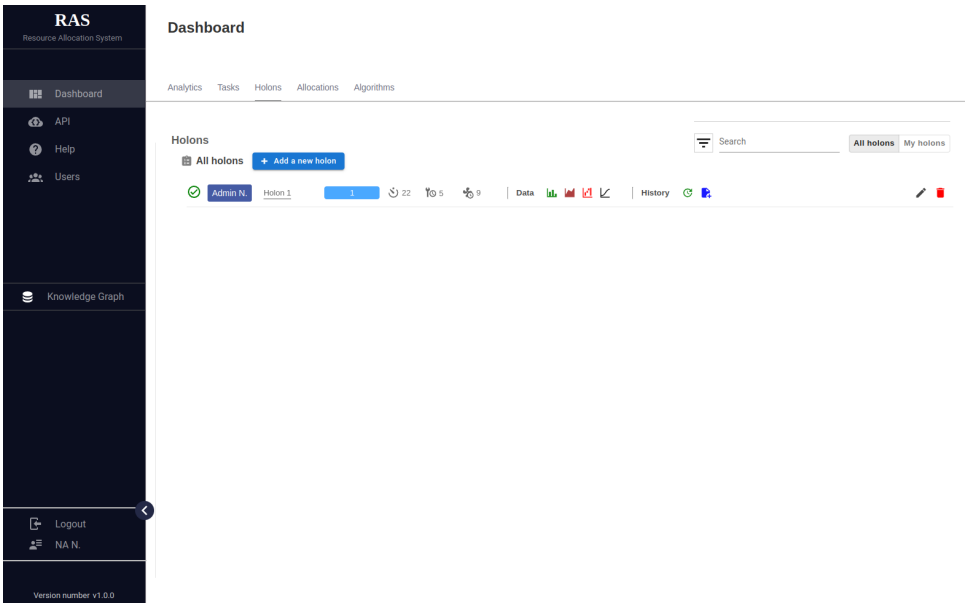
Tasks-näkymä on tehtävien hallintapaneeli, jossa voi luoda, muokata, esikatsella ja poistaa tehtäviä. Tehtävien hallintapaneelissa on myös suodatin ja hakukenttä, jolla voi etsiä ja suodattaa tehtäviä. Tehtävien muokkaaja mahdollistaa kaikkien tehtävien tietojen muokkaamista ja esikatselua. Kuvassa 6.7 on kuvakaappaus Tasks-näkymästä.

Holons-näkymä on holoneja varten luotu hallintapaneeli, jolla on samanlainen rakenne kuin Tasks-näkymä. Näkymässä on hallintapaneeli, jolla voi luoda, muokata, esikatsella ja poistaa holoneja. Näiden lisäksi näkymällä on vastaavanlainen hakukenttä ja suodatin. Näkymässä voi myös katsella holoneihin liittyviä historiatietoja. Kuvassa 6.8 näkyy kuvankaappaus Holons-näkymästä.

Allocations-näkymässä on työkaluja allokointipyyntöjen luomiseen, muokkaamiseen ja hallitsemiseen. Näkymässä voi selata, suodattaa ja hakea allokointipyyntöjä erilaisilla hauilla. Näkymän allokointipyyntömuokkaaja tarjoaa allokointipyyntöjen tietojen muokkaamisen lisäksi allokointipyyntöjen vahvistamista. Vahvistettujen allokointipyyntöjen tehtävät siirtyvät valituille holoneille, niin että tehtävän tiedoista pystyy



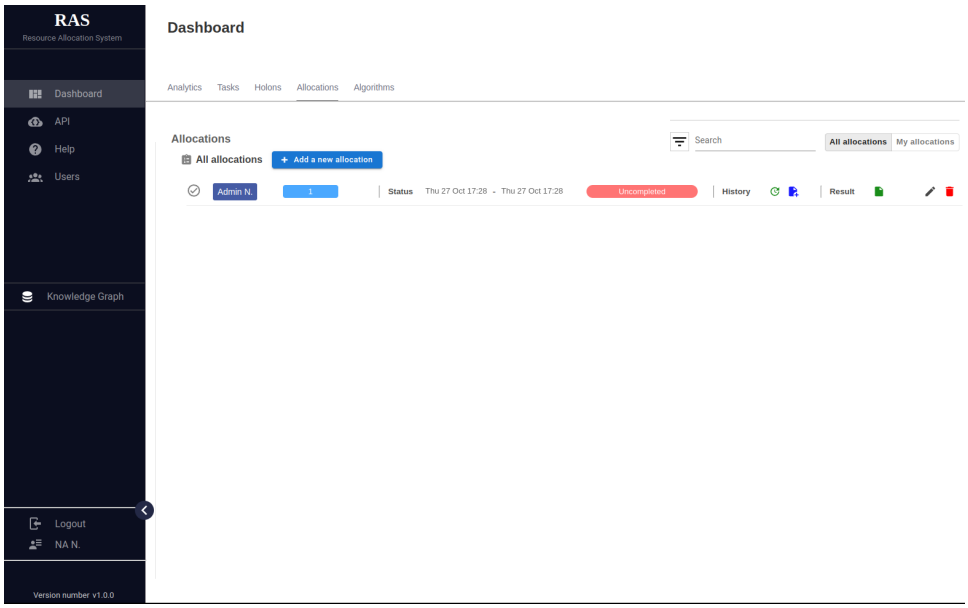

Kuva 6.7: Tasks-näkymä




Kuva 6.8: Holons-näkymä

näkemään mitkä holonit kuuluvat millekin tehtävälle. Kuvassa 6.9 on kuvakaappaus Allocations-näkymästä.

Viimeinen näkymä Algorithms on järjestelmään rekisteröityjen algoritmien selaamiselle tarkoitettu näkymä. Näkymässä on algoritmien nimet, kuvaukset, tyypit ja määrä.




Kuva 6.9: Allocations-näkymä

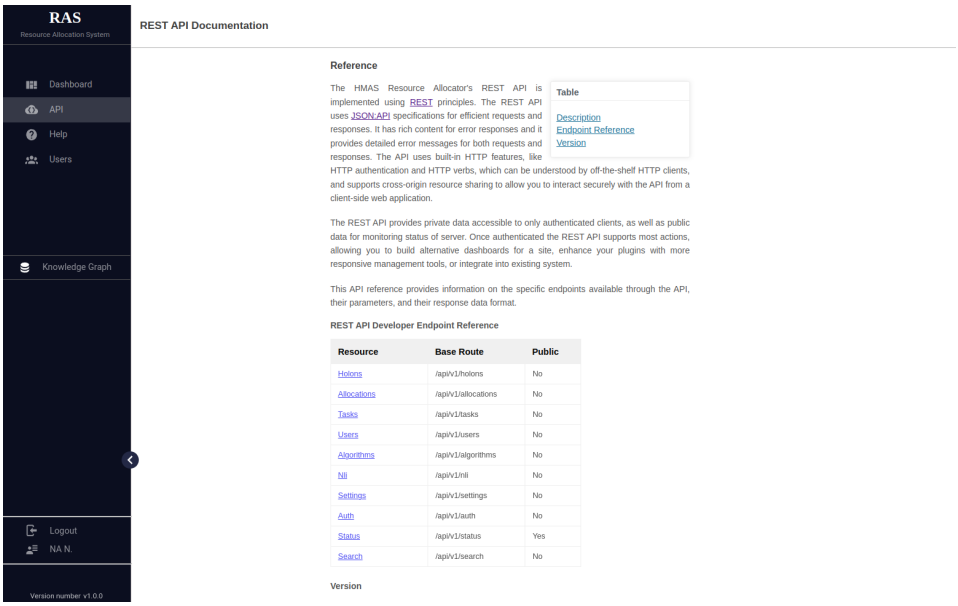
Näkymä ei tarjoa työkaluja algoritmien poistamiseen tai muokkaamiseen, sillä algoritmeja on tarkoitus muokata ainoastaan ohjelmallisesti.

### API

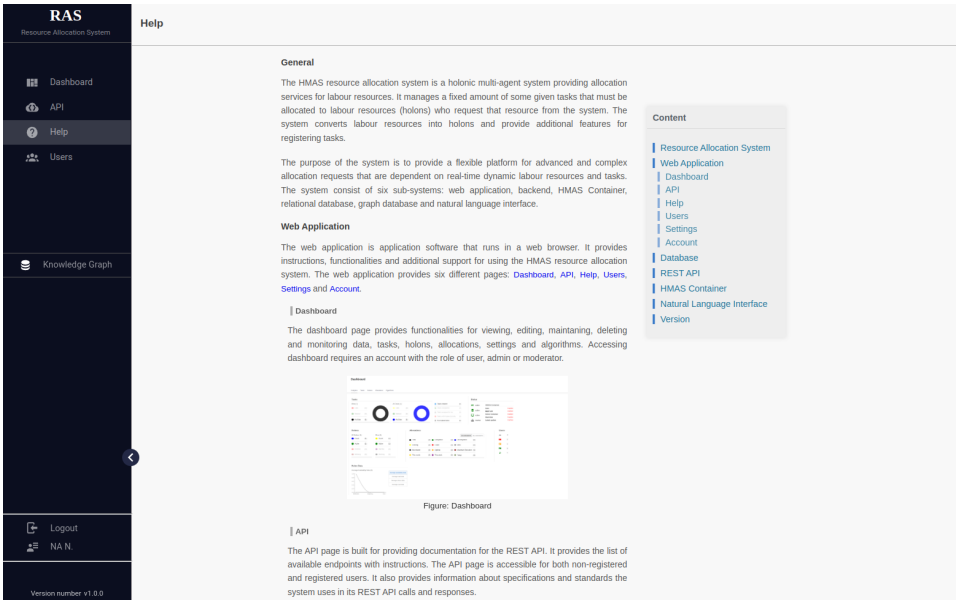
API-sivu on toteutettu alikappaleen 5.5.1 mukaisesti. Sivu tarjoaa REST-rajapinnan dokumentaation ja se sisältää ohjeet kutsujen luomiseen ja vastausten lukemiseen. Sivun on avoin myös rekisteröitymättömille käyttäjille ja sen tarkoituksena on toimia dokumentaationa sekä rajapinnan käyttäjille että ohjelmistokehittäjille. Kuvassa 6.10 on kuvakaappaus sivun sisällöstä.

### Help

Help-sivu tarjoaa ohjeita ja kuvauksia järjestelmästä. Sivussa on tietoa järjestelmästä, web-sovelluksesta, kojelaudasta ja muista sovelluksen sivuista. Sivu tarjoaa myös tietoa tietokannan rakenteesta ja muista järjestelmän osista. Sivun tarkoitus on alikappaleen 5.5.1 mukaisesti tarjota kaikille käyttäjille yleistä tietoa järjestelmästä. Kuvassa 6.11 on kuvakaappaus Help-sivusta.




Kuva 6.10: API-sivu

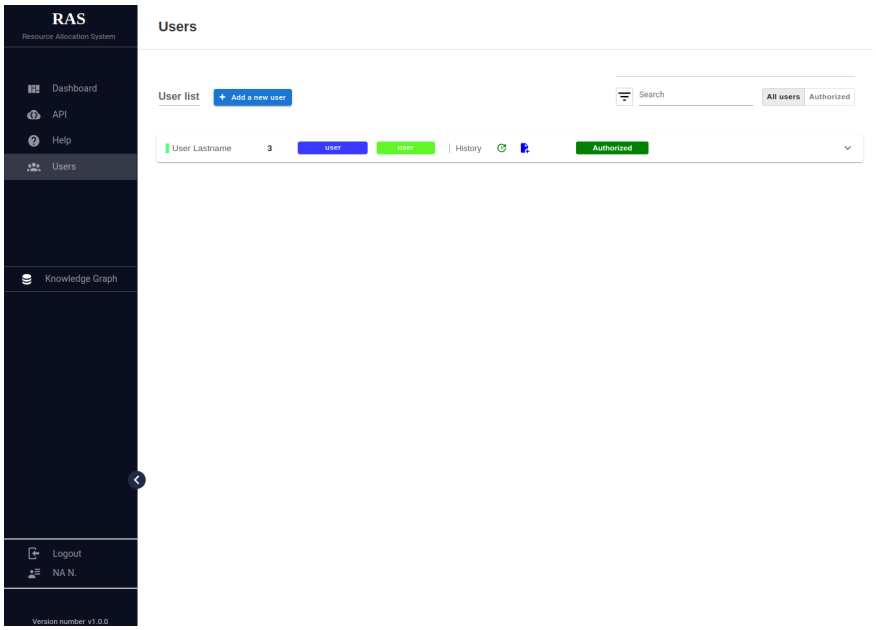



Kuva 6.11: Help-sivu

## Users

Users-sivu on käyttäjien hallintapaneeli, jossa voi luoda, muokata, etsiä ja selata käyttäjiä ja heidän tietojaan. Käyttäjät voivat user-sivusta etsiä käyttäjien tunnuksia, rooleja ja historiatietoja. Sivun käyttäminen vaatii käyttäjätunnuksen, jolla on user, admin tai

moderator rooli. User-roolin käyttäjät voivat muokata ainoastaan omia tietoja ja luoda user-käyttäjiä. Moderator-käyttäjät saavat muokata sekä omia että user-käyttäjien tietoja. Admin-roolin käyttäjät voivat luoda ja muokata sekä moderator että user roolin käyttäjiä. Kuva 6.12 näyttää kuvakaappauksen sivusta.




Kuva 6.12: Users-sivu

**Account**

Account-sivu on omien käyttäjätietojen hallintapaneeli. Sivussa voi muokata omia tietoja ja salasanaa. Sivussa on myös "Delete Account"niminen toiminnallisuus, jolla voi poistaa oman käyttäjätilin. Linkki sivuun sijaitsee "Logout-painikkeen alapuolella.

## 6.2 Testaus

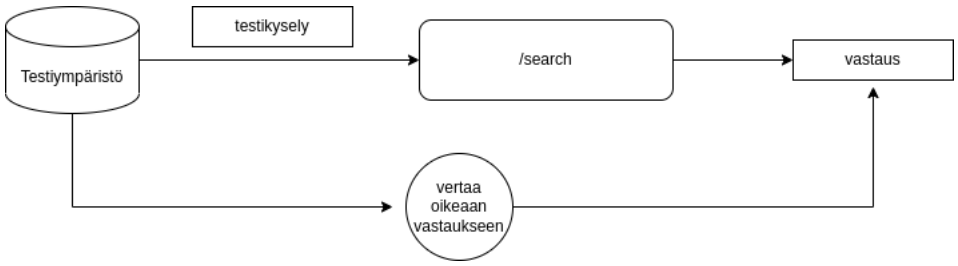
Jokaista ohjelmistoa voi testata ainakin neljällä eri testausstrategialla. Nämä testausstrategiat ovat yksikkötestaus, integraatiotestaus, validointitestaus ja järjestelmätestaus [27]. Näistä yksikkötestaus ja integraatiotestaus ovat kaksi kattavaa strategiaa, joilla resurs-

sien allokointijärjestelmä ja sen osia on testattu tässä työssä. Näiden molempien strategioiden lisäksi REST-rajapinnan testauksessa on käytetty räätälöityä testausstrategiaa. Räätälöidyllä testausstrategialla on testattu rajapinnan pääpisteet ja niiden palauttamat vastauspyynnöt.

### 6.2.1 Backend ja REST-rajapinta

Backend-komponentin ja REST-rajapinnan päätepisteet on testattu räätälöidyllä testausstrategialla, joka koostuu integraatio- ja yksikkötestausstrategiasta ja musta laatikko-menetelmästä. Testeissä on testattu pääpisteiden tarjoamat palvelut esimerkkisyötteillä ja verrattu näiden palauttamat viestit oikeisiin vastauksiin. Testien tarkoitus on varmistaa pääpisteiden toimivuutta, niiden palauttamien vastauksien oikeellisuutta ja rajapinnan yleisiä ominaisuuksia.

Pääpisteiden testauksissa testataan päätepisteen nimeä, parametreja, kyselyparametreja ja sen tukemaa HTTP-metodeja ja toiminnallisuutta. Kuvassa 6.13 on esimerkki testi, jossa päätepisteelle /search tehdään esimerkkikysely. Testin tuloksena päätepiste tuottaa vastauksen, jota verrataan oikeaan testikyselyn vastaukseen. Mikäli päätepisteen tuottama vastaus on oikein, testi menee läpi, muulloin testi epäonnistuu. Testeissä on pyritty varmistamaan etenkin päätepisteiden palautusviestien rakenteen oikeellisuutta vertaamalla ne JSON:API-standardien rakenteisiin.




Kuva 6.13: Päätepisteen testausprosessi

Backend-komponentin SQL kysely-generaattori on testattu yksikkötesteillä. Testit luovat esimerkkikyselyjä annetuilla esimerkkisyötteillä ja tarkistavat sql-kyselyjen

oikeellisuutta vertaamalle ne oikeisiin kyselyihin. Johtuen kyselygeneraattorin kriittisyydestä yksikkötestejä on kirjoitettu huolella erittäin kattavilla syötteillä. Viimeiset yksikkötestit koskevat graafitietokantaa, joissa testit luovat uusia graafitietokannan käyttäjiä testaamalle niiden toimivuutta.

### 6.2.2 Web-sovellus

Web-sovellus on testattu kolmella eri integraatiotestillä. Ensimmäinen integraatiotesti liittyy sovelluksen reitittimeen. Testissä testaan saatavilla olevat reitit ja niiden sisällöt vertaamalla reittien sisältöä odotettavaan sisältöön. Reiteistä dashboard, users, account, api ja help on testattu varmistamalla että selain avaa kyseiset sivut ja näyttää niiden oikeat sisällöt.

Toinen integraatiotesti koskee kojelautan Tasks-näkymää. Testissä tarkistetaan reitittimen toimivuutta, tehtävien poistamista, tehtävien lisäämistä ja tehtävien osia. Testi koostuu neljästä eri osatestistä, josta ensimmäinen poistaa kaikki tehtävät, toinen luo uutta tehtävää, kolmas muokkaa tehtävää ja neljäs tarkistaa tehtävän tietoja. Kolmas integraatiotesti testaa samoilla vaiheilla holoneja.

### 6.2.3 HMAS Container

HMAS Container-komponentin testit koostuvat yksikkötesteistä ja integraatiotesteistä. Yksikkötestit testaavat allokointialgoritmeja, komponentin kriittisiä metodeja ja sen tarjoama REST-rajapintaa. Allokointialgoritmien testeissä allokointialgoritmien oikeellisuutta varmistetaan muiden yksikkötestien tapaan syöttämällä siihen esimerkkisyötteitä ja vertaamalla sen tuottamia tuloksia oikeisiin tuloksiin. Rajapinnan status-päätepisteen yksikkötestaus varmistaa, että rajapinta palauttaa oikean tiedon komponentin tilasta ja sen osista.

Integraatiotestit testaavat komponenttia ja sen toiminnallisuuksia yhtenä suurena kokonaisuutena. Testit luovat testiallokointipyyntöjä ja lähettävät ne komponentille

REST-rajapinnan kautta. Testeissä HMAS Container-komponentin relaatiotietokantaan tuottamat tulokset verrataan odotettaviin tuloksiin. Testien sisältöön kuuluu allokointi-pyynnöt, utils-kirjaston yksikkötestit ja erilaiset kestävyystestit. Testien tarkoituksena on varmistaa tässä työssä laadittujen vaatimusten mukainen toiminnallisuus.

## 6.3 Laajennettavuuskokeet

Tämän työn tutkimuskysymysten vastaamisen helpottamiseksi suoritetaan kaksi laajennettavuuskoetta. Molemmissa kokeissa laajennetaan järjestelmää uusilla ominaisuuksilla ja analysoidaan laajentumisesta aiheutuvia muutoksia, töitä ja muita asioita. Muutosten kohdalla keskitytään arkkitehtuurillisiin muutoksiin ja järjestelmän sisäisiin komponentteihin. Kokeiden lopussa kirjoitetaan lyhyitä ytimekkäitä analyysejä kokeista ja niiden tuloksista.

### 6.3.1 Algoritmikokeet

#### Kuvaus

Järjestelmän kyky hyväksyä uusia algoritmeja testataan lisäämällä uutta algoritmia nimeltä FindHeavyTask. FindHeavyTask-algoritmi etsii ja poimii holoneja niille tehtäville joiden suorittaminen vaatii eniten resursseja. Tällä kokeella pyritään selvittämään algoritmien lisäysprosessia holonisessa moniagenttijärjestelmässä analysoimalla ja tarkkailemalla lisäysprosessin eri vaiheet.

#### Toteutus

HMAS Container-komponentin algorithms-kansiolle luodaan uusi tiedosto nimellä FindHeavyTask.js. Algorithms.js-tiedostoon lisätään algoritmin nimi, kuvaus, sijainti ja viittaus. FindHeavyTask-algoritmin rakenne koostuu seitsemästä eri vaiheesta: tehtävien lajittelu vaadittavan työmäärän mukaisesti, holonien lajittelu ja suodatus, holonien kloonaus, virhetilojen tarkistus, tehtävän valinta, algoritmin kutsu ja vastauksen palautus.

Tehtävien lajitteluvaiheessa tehtävät lajitellaan työmäärän eli *estimated_time* parametrin mukaisesti. Tehtävät, joiden työmäärä on suurin siirtyvätlistan alkuun ja tehtävät joiden aloituspäivämäärä ei ole vielä alkanut poistetaan listasta. Tuloksena syntyy lista,

jossa tehtävät on lajiteltu työmäärän eli *estimated_time* parametrin mukaisesti.

Holonien lajitteluvaiheessa holonien saatavuus tarkistetaan poistamalla ne holonit, joiden saatavuus parametri on poissa päältä. Sen jälkeen tarkistetaan holonien saatavuusdata ja päivittäiset työtunnit. Mikäli päivittäisiä työtunteja on saatavilla ja saatavuuskäyrän arvo on suurempi kuin 0 ja pienempi kuin 1, holoni merkitään käyttökelpoiseksi. Muulloin, holoni on käyttökelvoton ja se ei kelpaa allokointialgoritmille.

Holonien kloonausvaihe luo holoneista klooneja, jotta allokointialgoritmi ei tee ajonaikaisia muutoksia alkuperäisiin holoneihin. Kloonausvaihetta seuraa virhetilojen tarkistusvaihe, jossa tarkistetaan mahdollisten virheiden syntyä edellisissä vaiheissa. Virheen syntyminen tarkoittaa, että joko yksi tai useampi holoni tai tehtävä ovat virheellisiä ja siten algoritmille kelvottomia.

Tehtävän valintavaiheessa valitaan tehtävä, jonka arvioitu työmäärä on kaikkein suurin. Tämän jälkeen siirrytään seuraavalle vaiheelle, jossa kutsutaan itse algoritmia. Viimeisessä vaiheessa palautetaan algoritmin vastaus ja lopetetaan laskeminen. Algoritmin palauttamassa vastauksessa on tehtävät ja niille valitut holonit. Mikäli algoritmi joutuu virhetilanteeseen, vastauksena palautetaan virheviesti.

**Analyysi**

Algoritmi testattiin luomalla kaksi tehtävää ja kaksi holonia. Web sovelluksen Allocations-näkymässä luotiin uusi allokointipyyntö, jonka algoritmiksi valittiin FindHeavyTask-algoritmi. Molemmat vasta luodut tehtävät ja holonit valittiin allokointipyyntöön ja sen jälkeen palvelimelle lähetettiin pyyntö. Algoritmi onnistui tehtävässään etsimällä ensiksi sille tehtävälle holonin, jonka työmäärä oli suurin.

Algoritmin lisääminen oli hyvin helppo ja mutkaton. Lisäysoperaatiossa seurattiin edellisiä ohjeita, jossa pyydettiin rekisteröimään se Algorithms.js-tiedostoon, joka toimii rajapintana moniagenttijärjestelmän allokointiholonille. Sen jälkeen kirjoitettiin varsinainen koodi ohjeiden mukaisesti: algoritmi tarjoaa run nimisen julkisen funktion,

joka vastaanottaa holonit, tehtävät ja palauttaa vastauksen JSON-muodossa. Lisäämisprosessi ei vaatinut arkkitehtuurillisia muutoksia.

### 6.3.2 Tietämyskantakokeet

#### Kuvaus

Järjestelmän lisätään uusi algoritmi, joka hakee tietämyskannasta osaamisen ja sairauksien lisäksi *specialized* nimisen tiedon. *Specialized*-parametri kertoo minkä tyyppisiin tehtäviin holoni on erikoistunut ja pakottaa algoritmia valitsemaan pääosin niitä holoneja, jotka ovat erikoistuneita kyseiselle tehtävälle. Tällä kokeella pyritään selvittämään miten tietämyskanta pystyy auttamaan uudenlaisten älykkäiden algoritmien kehittämisessä.

#### Toteutus

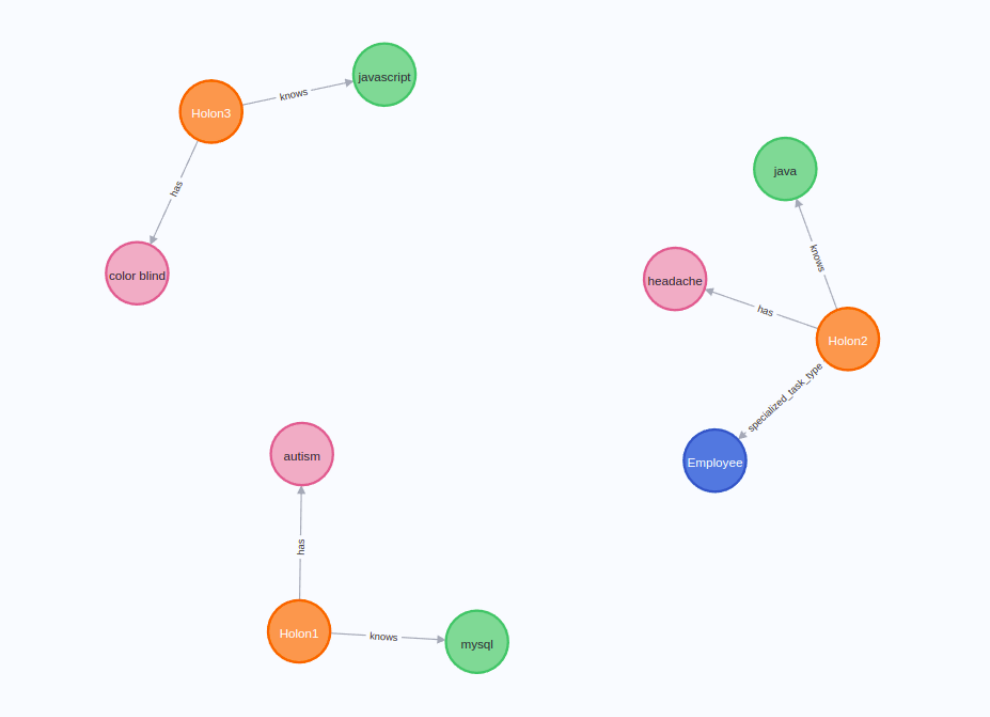
HMAS Container-komponentin algorithms kansiolle luodaan uusi tiedosto nimellä FindBySpecialization.js. Algorithms.js-tiedostoon lisätään algoritmin nimi, kuvaus, sijainti ja viittaus. FindBySpecialization-algoritmin rakenne koostuu yhdeksän eri vaiheesta: tehtävien lajittelu työmäärän mukaisesti, holonien lajittelu ja suodatus, holonien kloonaus, virhetilojen tarkistus, tehtävän valinta, algoritmin kutsu, holonien erikoistumisvertailu, vastauksen palautus ja tietämysgraafin lisääminen.

Ensimmäisessä vaiheessa lajitellaan tehtävät työmäärän mukaisesti. Tehtävät, joiden työmäärä on vähäinen, siirretään listan alkuun. Toisessa vaiheessa lajitellaan holonit ja poistetaan sellaiset holonit, jotka eivät ole käyttökelpoisia. Kolmannessa vaiheessa kloonataan holonit ja neljännessä vaiheessa tarkistetaan mahdollisia syntyneitä virheitä edellisissä vaiheissa. Jos virheitä ei tapahdu ohjelma jatkaa viidenteen vaiheeseen. Viidennessä vaiheessa valitaan tehtävä listan alusta ja se syötetään algoritmille, jolloin ohjelma siirtyy kuudenteen vaiheeseen.

Seitsemännessä vaiheessa valitaan tehtävälle sellainen holoni, jonka erikoistumisen

eli specialized-parametri vastaa tehtävän tyyppiä. Mikäli sellainen ei löydy, tehtävälle valitaan holoni listan kärjestä. Holonin erikoistuminen saadaan selville etsimällä tietämysgraafista sellaisen kolmikon, jonka suhteen nimi on *specialized_task_type*. Ohjelman löytäessä sellaisen kolmikon, objektista etsitään ominaisuus nimeltä *name*. Kahdeksannessa vaiheessa algoritmi palauttaa vastauksen JSON-muodossa.

Viimeisessä vaiheessa tietämysgraafiin lisätään holonin erikoistuminen. Tämä tapahtuu Neo4j-ohjelmiston tarjoamassa web-sovelluksessa. Tietämysgraafiin lisätään holoni, holonin tietämykset, sairaudet ja erikoistuminen. Erikoistumisen lisäyksessä noudatetaan sovittuja ohjeita, jotka ovat seuraavat: holoni-solmusta lähtee suhde nimellä *specialized_task_type* toiselle solmulle, jossa on ominaisuus nimeltä *name*. Kuvassa 6.14 näkyy graafi, jossa Holon2-solmulla on suhde nimeltä *specialized_task_type* toiseen solmuun, jolla on *name* niminen parametri arvoltaan "Management".




Kuva 6.14: Holonien graafi

#### Analyysi

Tietämyskantakokeen testaamiseksi luotiin kolme holonia ja kolme tehtävää. Ensimmäisen tehtävän tyypiksi nimettiin Management ja kahden muun tyypiksi kirjattiin IT. Graafiin lisättiin tietoa kolmesta holonista. Kolmannen holonin *specialized_task_type*-arvoksi tallennettiin Management ja algoritmi ajettiin syöttämällä siihen vasta luodut holonit ja tehtävät. Algoritmi laski tuloksen onnistuneesti ja määräsi kolmannen holonin ensimmäiselle tehtävälle.

Sekä algoritmin lisääminen että graafin laajentaminen onnistui ongelmitta ilman arkkitehtuurillisia muutoksia. Graafin laajentaminen oli hieman työlästä johtuen CQL (engl. Cypher Query Language) kielestä. Kielen syntaksi oli ainutlaatuinen ja sen oppiminen vaatii aikaa. Graafin laajentamisen jälkeen algoritmi palautti odotetun vastauksen JSON-muodossa.

## 6.4 Käytettävyyskokeet

### 6.4.1 Hallintapaneeli

#### Kuvaus

Kokeessa lisätään neljä holonia, viisi tehtävää ja uusi allokointipyyntö. Tämän jälkeen kirjaudutaan Neo4j-tietämyskantaan ja lisätään holoneille graafit. Tällä kokeella pyritään keräämään yleistä tietoa järjestelmän käytettävyydestä ja arvioimaan tietämyskannan järjestelmälle aiheuttamaa negatiivisia ja positiivisia käytettävyys kokemuksia. Koe auttaa myös ymmärtämään poikkeamia holonisen moniagenttijärjestelmän käytettävyydessä.

### Toteutus

Holonit lisättiin osoitteessa /dashboard/holons painamalla “Add a new holon “-painiketta. Holonien luonti-ikkunaan lisättiin luotavien holonien tiedot ja lopulta painettiin “Save changes “-painiketta. Järjestelmä loi holonit onnistuneesti ja näytti tulokset Holons-näkymässä. Samalla menetelmällä lisättiin tehtävät osoitteessa /dashboard/tasks. Tehtävien näkymä näytti myös vasta luodut tehtävät Tasks-näkymässä.

Allokointipyyntö tehtiin osoitteessa /dashboard/allocations painamalla “Add a new allocation“-painiketta. Painike avoi luontinäkymän, johon täytettiin uuden allokaatiopyynnön tiedot. Näkymään syötettiin algoritmi ja sen jälkeen valittiin holonit ja tehtävät. HMAS Container-komponentin laskettuaan allokaatiopyynnön tulos vahvistettiin siirtämällä “complete “-painikkeen on-tilaan ja painamalla “Save changes “-painiketta.

Kokeen viimeisessä vaiheessa lisättiin uusi allokaatiopyyntö. Sen jälkeen siirryttiin tietämyskantaan painamalla “Knowledge Graph “-painiketta ja syöttämällä tunnuksen ja salasanan kirjautumisikkunaan. Neo4j-tietäymskannan graafiin lisättiin holoneihin liittyviä tietämyksiä, sairauksia ja erikoistumisia satunnaisesti. Tämän jälkeen palattiin allokaationäkymään ja allokaatiopyyntö lähetettiin palvelimelle.

### Analyysi

Holonien, tehtävien ja allokaatiopyynnön lisääminen sujui ongelmitta ja prosessi oli yksinkertainen. Holoninen moniagenttijärjestelmä tai tietämyskanta eivät vaikuttaneet lisäämisprosessiin tai aiheuttaneet merkittäviä poikkeuksia verrattuna muihin järjestelmiin. Järjestelmän palauttamat viestit olivat selkeitä ja ymmärrettäviä. Tietämysgraafin lisäämiseksi piti selvittää holonien tunnukset ja tiedon lisäämisen tarvittavat CQL-komennot. CQL-komentojen luomisessa syntyi alussa kaksi virhettä, ensimmäinen liittyi syntaksiin ja toinen parametreihin. Virheiden korjausten jälkeen Neo4j hyväksyi komennot ja loi graafit holoneille. Kaiken kaikkiaan tietämyskannan käyttäminen oli kuitenkin vaativaa ja hankalaa.

# 7 Johtopäätökset

Tässä työssä rakennettiin resurssien allokointijärjestelmä, joka perustui holonisen moniagenttijärjestelmään ja tietämysgraafiin. Tämän työn motiivina oli selvittää mitä etuja tietämysgraafiin perustuvat holoniset moniagenttijärjestelmät tarjoavat, mitkä ovat tällaisten järjestelmien heikkoudet ja millainen tulevaisuus tällaisilla järjestelmillä on. Tutkimuskysymysten vastaamisen helpottamiseksi suoritettiin kolme eri koetta, joissa järjestelmästä kerättiin tuloksia analysointia varten.

Kolmessa suoritetussa kokeessa kerättiin tietoa järjestelmän laajentamisesta ja sen käytettävyydestä. Ensimmäisessä kokeessa lisättiin uusi algoritmi tarkkaillen lisäysprosessia ja sen järjestelmälle aiheuttamia muutoksia. Toisessa kokeessa laajennettiin tietämysgraafi ja tarkkailtiin lisäysprosessi alusta loppuun saakka. Molemmissa kokeissa lisäysprosessi ei aiheuttanut arkkitehtuurillisia tai rakenteellisia muutoksia järjestelmään. Moniagenttijärjestelmän hajautetun luonteen ansiosta lisäysprosessi onnistui vaivatta ja helposti.

Kolmannessa kokeessa testattiin käytettävyyttä ja kerättiin yleistä tietoa. Minkäänlaisia poikkeuksia ei havaittu käytettävyydessä holonisen moniagenttijärjestelmän kohdalla. Tietämysgraafi kuitenkin aiheutti merkittävän ongelman järjestelmän käytettävyydessä johtuen sen käyttövaikeudesta. Tietäymysgraafin käyttöön ei toistaiseksi löydy muuta keinoa kuin opetella siihen varten luotua kyselykieltä.

Holoninen moniagenttijärjestelmä tietämysgraafilla tarjoaa uudenlaisen lähestymiskulman uudenlaisten älykkäiden järjestelmien kehittämisessä. Holoninen monia-

genttijärjestelmä jakaa ongelma-alueen itsenäisiin holoneihin ja helpottaa laajan ja monimutkaisen järjestelmän kehittämistä. Tietämysgraafi tuottaa holoniseen moniagenttijärjestelmään uudenlaisen ominaisuuden eristämällä tietämyksen ja tiedon itsenäiseksi komponentiksi perinteisen sulatetun menetelmän sijaan. Näin ollen tietämys ja tieto eivät enää ole sulatettuja koodin seassa, vaan erillinen osa järjestelmässä. Tämän tyyppiset ratkaisut sopivat laskennallisesti hajautettuihin ongelmiin, joissa holarkia on sopiva ratkaisu ja joiden ratkaiseminen monoliittisella arkkitehtuurilla on vaikeaa.

Holonisen moniagenttijärjestelmän kehittäminen tietämysgraafilla on kuitenkin heikkouksia ja poikkeamia muista järjestelmistä. Järjestelmä sopii parhaiten laajoille monimutkaisille ongelmille helppojen yksinkertaisten ongelmien sijaan. Tietämysgraafille ei ole vielä kehitetty helpompaa käyttömenetelmää kuin graafitietokannan kyselykielen käyttäminen, joten tällaisen järjestelmän täyshyödyntäminen vaatii kyselykielen oppimista. Järjestelmä poikkeaa arkkitehtuuriltaan merkittävästi muista järjestelmistä ja tämä taas vaatii kehittäjiltä riittävää asiantuntemusta moniagenttijärjestelmistä.

Holoniset moniagenttijärjestelmät ovat 1990-luvulta lähtien yleistyneet erilaisin aloihin ja jatkavat leviämistä kovaa vauhtia. Niitä käytetään lääketieteessä [28], kasvitieteissä [29], mikropiireissä [30], liikenteenohjauksessa [31] ja jopa arkeologiassa [32]. Nimenomaan tietäymysgraafien perustuvien holonisten moniagenttijärjestelmien tulevaisuudesta on kuitenkin hankala lausua mitään johtuen aiheen vähäisestä tutkimuksesta ja sen suhteellisen vähäisestä käyttäjämäärästä. Tästä huolimatta tietämysgraafiin perustuvien holonisten moniagenttijärjestelmien ennakoidaan myös yleistymään, sillä tarve tietämyskantoihin perustuvien järjestelmien kehittämiselle kasvaa tekoälyn yleistymisen ansiosta.

Liite 1

Tässä liitteessä on resurssien allokointijärjestelmän REST-rajapinnan pääpisteet ja niille kuuluvat reitit. Kaikki pääpisteet tukevat GET, POST ja PATCH metodeja REST-kyselyissä. Lisäksi jokaisella päätepisteellä on parametrit, joita REST-rajapinta tukee sovitussa formaatissa. Kaikki pääpisteet ovat käyttäjien saatavilla - poikkeuksena algorithms-päätepiste, jonka POST ja PATCH metodit ovat sallittuja ainoastaan HMAS Container-komponentille.

| **Resurssi** | **Reitti** |
|---|---|
| /holons | /api/v1/holons |
| /allocations | /api/v1/allocations |
| /tasks | /api/v1/tasks |
| /users | /api/v1/users |
| /algorithms | /api/v1/algorithms |
| /nli | /api/v1/nli |
| /settings | /api/v1/settings |
| /auth | /api/v1/auth |
| /status | /api/v1/status |
| /search | /api/v1/search |

| **/holons-päätepisteen parametrit** | **Metodit** |
|---|---|
| id (numero), type (teksti), name (teksti), gender (teksti), daily_work_hours (numero), latest_state (JSON), remote_address (teksti), api_token (teksti), availability_data (JSON), load_data (JSON), stress_data (JSON), cost_data (JSON), age (numero), experience_years (numero), created_on (päivämäärä), updated_on (päivämäärä), created_by (numero), is_available (boolean) | GET, POST, PATCH |

| **/allocations-päätepisteen parametrit** | **Metodit** |
|---|---|
| id (numero), request_by (numero), request (teksti), result (teksti), start_time (päivämäärä), end_time (päivämäärä), created_on (päivämäärä), completed_on (päivämäärä), updated_on (date), reallocate (boolean) | GET, POST, PATCH |

| **/tasks-päätepisteen parametrit** | **Metodit** |
|---|---|
| id (numero), type (teksti), name (teksti), description (teksti), estimated_time (numero), knowledge_tags (JSON), resource_demand (JSON), priority (numero), created_on (päivämäärä), created_by (numero), start_date (päivämäärä), due_date (päivämäärä), assigned_to (numero), updated_on (päivämäärä), completed_on (päivämäärä), is_completed (boolean) | GET, POST, PATCH |

| **/users-päätepisteen parametrit** | **Metodit** |
|---|---|
| id (numero), role (teksti), username (teksti), password (teksti), firstname (teksti), lastname (teksti), email (teksti), created_on (päivämäärä), last_login (päivämäärä), updated_on (päivämäärä) | GET, POST, PATCH |

| **/algorithms-päätepisteen parametrit** | **Metodit** |
|---|---|
| id (numero), type (teksti), name (teksti), description (teksti), created_on (päivämäärä), updated_on (päivämäärä), created_by (numero) | GET, POST, PATCH |

| **/nli-päätepisteen parametrit** | **Metodit** |
|---|---|
| text (teksti) | GET |

| **/settings-päätepisteen parametrit** | **Metodit** |
|---|---|
| id (numero), settings (JSON), created_on (päivämäärä), updated_on (päivämäärä), created_by (numero) (teksti) | GET, POST, PATCH |

| **/auth-päätepisteen parametrit** | **Metodit** |
|---|---|
| username (teksti), password (teksti) | GET |

| **/status-päätepisteen parametrit** | **Metodit** |
|---|---|
| Ei parametreja, ainoastaan GET-kutsu. | GET |

| **/search-päätepisteen parametrit** | **Metodit** |
|---|---|
| type (teksti), resource (teksti), latest_update (päivämäärä), ids (JSON) | POST |

# Lähdeluettelo


[1] G. Weiss, *Multiagent systems: second edition*. MIT press, 2016, s. 51–52.

[2] E. Adam, R. Mandiau ja C. Kolski, ”HOMASCOW: a holonic multi-agent system for cooperative work”, syyskuu 2000, s. 247–253, ISBN: 0-7695-0680-1. DOI: 10.1109/DEXA.2000.875035.

[3] ——, ”Application of a holonic multi-agent system for cooperative work to administrative processes”, *Journal of Applied Systems Studies (JASS)*, vol. 2, s. 100–115, tammikuu 2001.

[4] S. Rodriguez, V. Hilaire, N. Gaud, S. Galland ja A. Koukam, ”Holonic Multi-Agent Systems”, teoksessa. tammikuu 2011, vol. 37, s. 238–263, ISBN: ISBN 978-3642173479. DOI: 10.1007/978-3-642-17348-6_11.

[5] S. Rodriguez, V. Hilaire ja A. Koukam, ”Towards a holonic multiple aspect analysis and modeling approach for complex systems: Application to the simulation of industrial plants”, *Simulation Modelling Practice and Theory*, vol. 15, s. 521–543, toukokuu 2007. DOI: 10.1016/j.simpat.2007.01.005.

[6] ——, ”Formal Specification of Holonic Multi-Agent Systems Framework”, vol. 3516, toukokuu 2005, s. 719–726. DOI: 10.1007/11428862_98.

[7] C. Gerber, J. Siekmann ja G. Vierke, ”Holonic multi-agent systems”, 1999.

[8] K. Fischer, M. Schillo ja J. Siekmann, ”Holonic multiagent systems: A foundation for the organisation of multiagent systems”, teoksessa *International conference*

*on industrial applications of holonic and multi-agent systems*, Springer, 2003, s. 71–80.

[9] L. Ehrlinger ja W. Wöß, ”Towards a definition of knowledge graphs.”, *SEMANTiCS (Posters, Demos, SuCCESS)*, vol. 48, nro 1-4, s. 2, 2016.

[10] M. Kejriwal, C. A. Knoblock ja P. Szekely, *Knowledge Graphs: Fundamentals, Techniques, and Applications*. Cambridge, MA: The MIT Press, 2021.

[11] T. Yu, J. Li, Q. Yu, Y. Tian, X. Shun, L. Xu, L. Zhu ja H. Gao, ”Knowledge graph for TCM health preservation: Design, construction, and applications”, *Artificial intelligence in medicine*, vol. 77, s. 48–52, 2017.

[12] R. M. Keller, ”Building a knowledge graph for the air traffic management community”, teoksessa *Companion Proceedings of The 2019 World Wide Web Conference*, 2019, s. 700–704.

[13] W. Xiong, T. Hoang ja W. Y. Wang, ”Deeppath: A reinforcement learning method for knowledge graph reasoning”, *arXiv preprint arXiv:1707.06690*, 2017.

[14] Y. Jia, Y. Qi, H. Shang, R. Jiang ja A. Li, ”A practical approach to constructing a knowledge graph for cybersecurity”, *Engineering*, vol. 4, nro 1, s. 53–60, 2018.

[15] A. Hogan, E. Blomqvist, M. Cochez, C. d’Amato, G. d. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier et al., ”Knowledge graphs”, *Synthesis Lectures on Data, Semantics, and Knowledge*, vol. 12, nro 2, s. 1–257, 2021.

[16] Y. Duan, L. Shao, G. Hu, Z. Zhou, Q. Zou ja Z. Lin, ”Specifying architecture of knowledge graph with data graph, information graph, knowledge graph and wisdom graph”, teoksessa *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*, IEEE, 2017, s. 327–332.

[17] A. Hogan, ”Resource description framework”, teoksessa *The Web of Data*, Springer, 2020, s. 59–109.

[18] D. Brickley, *Foaf vocabulary specification 0.99*, 2022. url: http://xmlns.com/foaf/spec/#term_name.

[19] R. Angles, "The Property Graph Database Model.", teoksessa *AMW*, 2018.

[20] N. Johansson ja A. Löfgren, "Designing for Extensibility: An action research study of maximizing extensibility by means of design principles", B.S. thesis, 2009.

[21] A. Seffah ja E. Metzker, "The obstacles and myths of usability and software engineering", *Communications of the ACM*, vol. 47, nro 12, s. 71–76, 2004.

[22] K. Lai, L. Rasmusson, E. Adar, L. Zhang ja B. A. Huberman, "Tycoon: An implementation of a distributed, market-based resource allocation system", *Multiagent and Grid Systems*, vol. 1, nro 3, s. 169–182, 2005.

[23] S. Martello, D. Pisinger ja D. Vigo, "The three-dimensional bin packing problem", *Operations research*, vol. 48, nro 2, s. 256–267, 2000.

[24] M. Oprea, "Applications of multi-agent systems", s. 246–248, 2013.

[25] V. Surwase, "REST API modeling languages-a developer's perspective", *Int. J. Sci. Technol. Eng*, vol. 2, nro 10, s. 634–637, 2016.

[26] J. API. (2022). "JSON API Specification", url: https://jsonapi.org/format/.

[27] A. A. Sawant, P. H. Bari ja P. Chawan, "Software testing techniques and strategies", *International Journal of Engineering Research and Applications (IJERA)*, vol. 2, nro 3, s. 980–986, 2012.

[28] R. Unland, I. Al-Qaysi, Z. Othman, C. Weihs ja C. Branki, "Holonic Multi Agent System and Medical Diagnosis Decision", tammikuu 2010.

[29] S. Rodriguez, V. Hilaire ja A. Koukam, "Towards a holonic multiple aspect analysis and modeling approach for complex systems: Application to the simulation of industrial plants", *Simulation Modelling Practice and Theory*, vol. 15, s. 521–543, toukokuu 2007. DOI: 10.1016/j.simpat.2007.01.005.

[30] Y. Nurdin ja S. Muchallil, ”Holonic Multi-Agent System for Microgrid Hierarchical Control”, elokuu 2019, s. 66–71. DOI: 10.1109/CYBERNETICSCOM.2019.8875690.

[31] M. Abdoos, N. Mozayani ja A. Bazzan, ”Holonic Multi-agent Systems for Traffic Signals Control”, *Engineering Applications of Artificial Intelligence*, vol. 26, s. 1575–1587, toukokuu 2013. DOI: 10.1016/j.engappai.2013.01.007.

[32] V. Mascardi, D. Briola, A. Locoro, D. Grignani, V. Deufemia, L. Paolino, N. Bianchi, H. Lumley, D. Malafronte ja A. Ricciarelli, ”A holonic multi-agent system for sketch, image and text interpretation in the rock art domain”, *International Journal of Innovative Computing, Information and Control*, vol. 10, s. 81–99, helmikuu 2014.