Joni Suhonen

Graafitietokannan visualisointi- ja hallintatyökalu

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

1.5.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Joni Suhonen Graafitietokannan visualisointi- ja hallintatyökalu 42 sivua 1.5.2018
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	tieto- ja viestintätekniikka
Ammatillinen pääaine	Software Engineering
Ohjaajat	lehtori Peter Hjort toimitusjohtaja Janne Saarela

Insinöörityössä tutkittiin RDF (Resource Description Framework) -tietomallia, sen visualisointia ja sen luonnollisia haasteita ja potentiaalisia menetelmiä haasteiden ratkaisemiseksi. Tutkinnan pohjalta oli tarkoitus kehittää sovelluskomponentti, joka mahdollistaa RDF-tietomallin avulla esitettyjen graafien visualisoinnin hyödyntäen tutkinnan aikana löydettyjä ratkaisuja visualisointihaasteisiin.

Insinöörityössä tutkittiin RDF-tietomallin rakennetta, historiaa ja sen konkreettista käyttöä sovelluksissa ja projekteissa. Työssä tutkittiin myös RDF-tietomallin käyttöä tukevia semanttisen verkon teknologioita, kuten SPARQL (SPARQL Protocol and RDF Query Language) -kyselykieli ja OWL (Web Ontology Language) -ontologiakieli. Tutkinnassa tuli esille useita RDF:n luonnollisia ominaisuuksia, jotka hankaloittavat sen visualisointia, kuten esimerkiksi semantiikan esittäminen ja sen säilyttäminen visualisoinnissa, ja tiedon luonnollinen laajamittainen linkittäytyminen.

Työssä selvitettiin tarkemmin visualisointihaasteiden syitä ja vaikutuksia, ja samalla tutkittiin mahdollisia motivaatioita RDF:n visualisoinnille ja sen hyötyjä erilaisissa käyttötapauksissa. Tutkinnan aikana löydettiin useita tapoja parantaa graafin visuaalisen esityksen ymmärrettävyyttä, kuten erilaiset visualisointistrategiat, sijoittelualgoritmit ja visuaalista semantiikkaa lisäävät toiminnot.

Insinöörityössä kehitettiin sovelluskomponentti, joka koostui JavaScript-kirjastosta ja graafitietokantaan integroidusta rajapinnasta, jotka kokonaisuutena muodostivat visualisointityökalun, joka mahdollistaa RDF-graafien selaamisen ja tutkimisen. Visualisointityökalun toteutuksessa ilmeni myös haasteita, kuten suorituskykyongelmat ja visualisoitavan tiedon määrä oikeissa sovelluksissa, joita ei tutkintavaiheen aikana pystytty havaitsemaan tai ratkaisemaan.

Työn tuloksia ja työssä toteutettua tutkintaa voi hyödyntää RDF-tietomallin visualisoinnissa ja sen haasteiden ratkaisemisessa, etenkin käyttäjäläheisissä sovelluksissa. Insinöörityö antaa myös yleismallisen kuvauksen RDF-tietomallista ja sen käytöstä semanttisen verkon sovelluksissa.

Avainsanat	RDF, semanttinen verkko, linkitetty data, graafivisualisointi

Author	Joni Suhonen
Title	Visualization and editing tool for a graph database
Number of Pages	42 pages
Date	1 May 2018
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Software Engineering
Instructors	Peter Hjort, Senior Lecturer Janne Saarela, CEO

The purpose of this thesis was to primarily research the RDF (Resource Description Framework) metadata data model, its visualization, the naturally occurring challenges within RDF visualization and some potential solutions and methods for dealing with these challenges. A software component was to be developed based on the research on RDF visualization in order to implement the research in practice. The primary goal of the software component was to allow the visualization of heterogeneous RDF data in a wide variety of application environments by applying the methods discovered during the research.

This thesis studied the structure of the RDF data model, its history and its usage in concrete applications and projects. The thesis also covers some of the technologies used in conjunction with RDF, such as the SPARQL (SPARQL Protocol and RDF Query Language) query language and OWL (Web Ontology Language) ontology language. The study of RDF data model revealed some inherent visualization hindering properties of RDF, for example representing the semantics of the visualized data and the interlinked nature of data in the RDF data model.

This thesis also explored in greater details the cause and effects of different RDF visualization challenges, and also tried to discover some motivations and practical use cases for RDF visualization. A variety of methods for improving the clarity of the visualization were found during the research on RDF visualization, such as different visualization strategies, placement algorithms and features that improved the visual semantics of the data.

The software component was composed of a JavaScript library and a REST API integrated into a graph database. Together they provided a tool for RDF visualization, which allows for graph exploration and examining.

The results and findings produced by this thesis can be applied for general visualization of the RDF data model and for solving visualization related challenges, especially in a user-centric application. This thesis also provides an overview of the RDF data model and its usage in semantic web applications.

Keywords	RDF, Semantic Web, Linked Data, Graph visualization

Sisällys

Lyhenteet

RDF	*Resource Description Framework.* Semanttisen tiedon esittämisen mahdollistava tietomalli.
W3C	*World Wide Web Consortium*. WWW:n standardeja ylläpitävä konsortio.
XML	*Extensible Markup Language.* Merkintäkieli, joka mahdollistaa tiedyn merkityksen esittämisen dokumenteissa.
URI	*Unique Resource Identifier.* Resurssin tunnistamiseen käytetty merkkijono.
IRI	*Internationalized Resource Identifier.* Kansainvälisiä merkistöjä tukeva URI.
RDFS	*RDF Schema.* Sanasto, joka laajentaa RDF:n tarjoamaa sanastoa ja mahdollistaa yksinkertaisen ontologiamallinnuksen.
OWA	*Open World Assumption.* Logiikkamalli, jossa väitteiden todenmukaisuuteen ei oteta kantaa jos todenmukaisuutta ei voida todistaa.
LOD	*Linked Open Data.* Linkitetty tieto, joka on julkaistu avoimesti.
SPARQL	RDF:n kyselykieli, jolla voidaan hakea tietoa semanttisia kyselyitä käyttäen RDF-tietokannoista.
OWL	*Web Ontology Language.* Semanttista verkkoa varten kehitetty ontologiakieli.

1 Johdanto

Insinöörityössä on tarkoituksena tutkia RDF (Resource Description work) -tietomallin visualisointia graafitietokannan yhteydessä ja siihen liittyviä haasteita, joita RDF-tietomalli luonnostaan sisältää. Tutkimuksen lisäksi toteutetaan yritykselle sovelluskomponentti, joka mahdollistaa RDF-tietomallin visualisoinnin ja yleiset tietokannan hallintaan liittyvät toimenpiteet visuaalisen käyttöliittymän kautta. Sovelluskomponentin tarkoitus on pääasiassa helpottaa tietokannan hallintaa ja auttaa tietokannassa säilytettävän tiedon ymmärtämisessä, mutta komponentin on oltava mahdollisimman joustava visualisoitavan tiedon kannalta, jotta sitä voidaan hyödyntää myös asiakaskäytössä.

RDF ja visio semanttisesta verkosta, jonka toteutumista varten RDF on kehitetty, ovat konsepteina jatkuvasti kehittyvän tieto- ja viestintätekniikan alalla jo suhteellisen vanhoja. Tästä huolimatta tietoisuus ja kiinnostuminen semanttisesta verkosta, ja sitä toteuttavista teknologioista, on ollut viime vuosikymmenen aikana vähäistä. Semanttisen verkon teknologiat ovat kuitenkin viime aikoina saaneet enemmän suosiota, kun ymmärrys semanttisen verkon ominaisuuksista ja hyödyistä on kasvanut, ja myös kaupalliset yritykset ovat alkaneet hyödyntää semanttista verkkoa joissakin sovelluksissaan (Workman 2016: 39). Aihealueen kasvanut suosio oli osasyynä insinöörityön aiheen valintaan.

Insinöörityössä keskitytään erityisesti RDF-tietomallilla esitetyn tiedon visualisointiin tavoilla, joilla tieto säilyttää semanttisen arvonsa, mutta on myös samalla ihmisille ymmärrettävää ja helposti käytettävää. RDF-tietomallin avulla esitettävä tieto saattaa olla luonnostaan vaikeasti ymmärrettävää ilman, että sen semantiikkaa hyödynnetään käyttöliittymätasolla tiedon luettavuuden helpottamiseksi. Tämän vuoksi myös käsittelemättömän RDF-tiedon esittäminen visuaalisesti on usein hankalaa.

Insinöörityöraportissa käydään läpi kolme eri aihealuetta, jotka yhdessä muodostavat lukijalle käsityksen insinöörityön eri vaiheista ja RDF-tietomallin käytöstä. Ensimmäisenä aiheena käsitellään RDF-tietomallin rakenne, historia, tarkoitus ja sen keskeisimmät konseptit. Sen jälkeen raportissa pohditaan RDF-tietomallin olennaisien piirteiden ja ominaisuuksien vaikutusta sen visualisointiin ja tapoja, joilla näistä aiheutuvia haasteita voidaan estää tai niiden vaikutusta vähentää. Lopuksi insinöörityössä raportoidaan,

kuinka RDF-tietomallin visualisoinnin suorittava sovelluskomponentti toteutettiin, millainen komponentin toteutusprosessi oli ja kuinka aiemmin todettuja tapoja visualisoinnin parantamiseen pystyttiin hyödyntämään käytännössä.

2 RDF-tietomallin perusteet

2.1 RDF-tietomallin tarkoitus ja historia

RDF-tietomalli on World Wide Web Consortiumin (W3C) standardisoima ja suosittelema infrastruktuuri, jonka avulla voidaan ilmaista tietoa resursseista. Konseptuaalisesti resurssit voivat RDF:ssä ilmaista mitä tahansa, esimerkiksi dokumentteja, ihmisiä, fyysisiä kohteita tai jopa abstrakteja konsepteja (Schreiber & Raimond 2014). Tietomalli mahdollistaa ja etenkin edistää erilaisen metadatan (tietoa tiedosta) uudelleenkäyttöä ja jakamista. RDF on kehitetty XML (Extensible Markup Language) -merkintäkielen pohjalta, mutta se käyttää omaa sanastoaan merkintäkielen ilmaisukyvyn rajoittamiseen tiedon semantiikan esittämistä varten. (Needleman 2001: 58.) Verrattuna muihin merkintäkieliin, kuten XML tai HTML (Hypertext Markup Language), RDF on kehitetty erityisesti parantamaan mahdollisuutta ilmaista eri tietojen ja tietolähteiden välisiä suhteita (Workman 2016: 2).

Kuten jo aiemmin mainittu, RDF:n on kehittänyt W3C, etenkin osana trendiä siirtyä pois ennalta määritellyistä ohjelmistologiikoista kohti dynaamisesti generoitua ja pääteltyä logiikkaa semanttisissa teknologioissa (Workman 2016: 2). RDF ei kuitenkaan ollut ensimmäinen verkon semantiikan ilmaisemista varten kehitetty mekanismi. Ennen RDF:n luomista oli esimerkiksi verkossa olevan aineiston ja tiedon arvostelua ja kuratointia varten kehitetty mekanismi nimeltä PICS (Platform for Internet Content Selection). Organisaatiot ja verkossa kokoontuvat ryhmät pystyivät käyttämään PICS:ä arvostelemaan verkosta löytyvää tietoa sen relevanttiuden perusteella. PICS oli kuitenkin arkkitehtuurisella tasolla puutteellinen, eikä sitä pystytty käyttämään yleisesti semantiikan määrittelyyn. (Needleman 2001: 58 – 59.)

RDF:n kehittäminen alkoi vuonna 1997, kun W3C kokosi useita metatietoihin erikoistuneita yhteisöjä RDF Working Group -työryhmäksi. Työryhmän tarkoitus oli määritellä ja löytää vaatimukset yleiselle semanttiselle kehikolle, joka mahdollistaa metadatan kuvailun ja linkittämisen resurssien avulla useissa erilaisissa sovelluksissa. Vaikka RDF ei

sinänsä ole kehitetty minkään aiemmin olemassa olevan semanttisen kehikon pohjalle, se on saanut hyvin paljon vaikutteita jo aikaisemmin mainitusta PICS-mekanismista ja myös muista metatietostandardeista, kuten Dublin Core, Warwick Framework ja Meta Content Framework. (Needleman 2001: 59.) Virallisesti W3C julkaisi ensimmäisen RDF-standardin suosituksen vuonna 1999, ja sitä on myöhemmin päivitetty vuosina 2004 ja 2014 (Schreiber & Raimond 2014).

2.2 RDF-tietomallin rakenne ja keskeiset konseptit

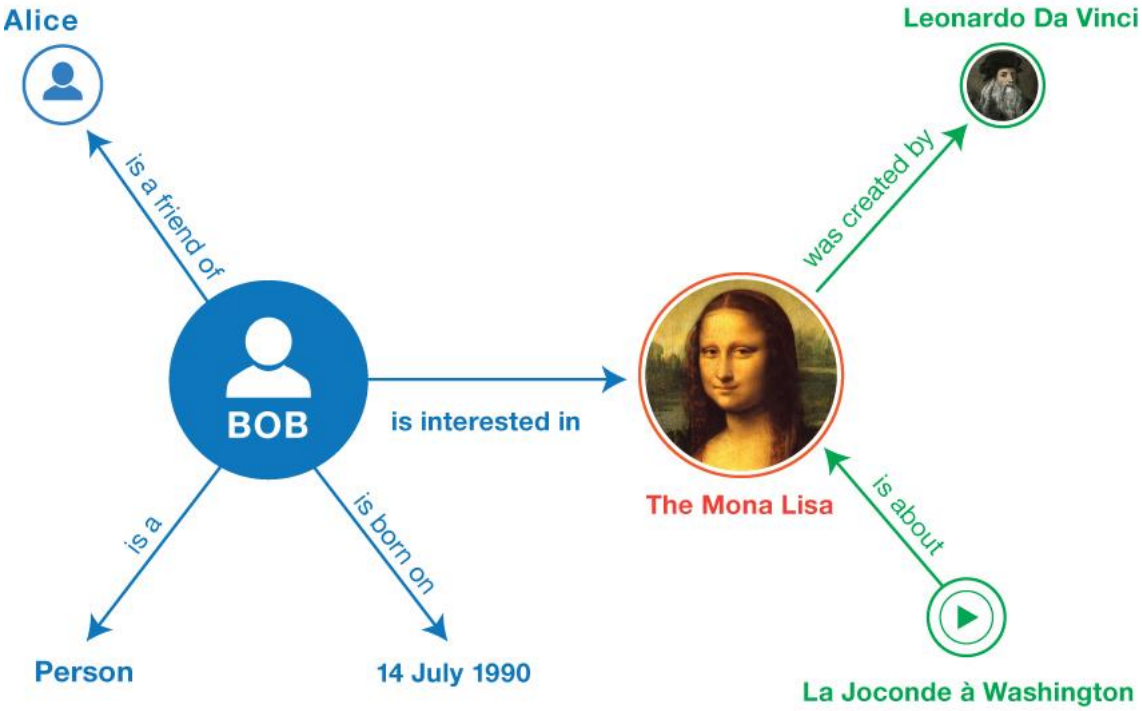
RDF-tietomallin pääasiallinen tarkoitus on siis mahdollistaa erilaisten resurssien kuvailu ja esitys. RDF-tietomallissa resurssilla tarkoitetaan mitä tahansa asiaa, joka voidaan tunnistaa URI:n (Uniform Resource Identifierin) avulla (Needleman 2001: 59). RDF koostuukin yleisellä tasolla resursseista, niiden ominaisuuksista ja resursseista tehtävistä toteamuksista tai lausunnoista (Workman 2016: 2). RDF-lausunnot seuraavat aina SPO (Subject, Predicate, Object) -rakennetta ja kuvaavat kahden resurssin välistä suhdetta. Lausunnon subjekti ja objekti ovat resurssit, joiden välille suhde muodostuu, ja lausunnon predikaatti kuvailee suhteen luonnetta. Kahden resurssin välinen suhde on aina suunnattu subjektilta objektille, ja sitä kutsutaan RDF:ssä ominaisuudeksi. Koska RDF-lausunnot koostuvat aina kolmesta elementistä, kutsutaan niitä yleensä kolmikoiksi tai tripleiksi. (Schreiber & Raimond 2014.) Saman resurssin URI esiintyy usein useassa eri triplessä. Esimerkkikoodissa 1 on kuvailtu suppea semanttinen verkko informaalisella RDF-esitystavalla triplejen avulla.

```
<Bob> <is a> <person>.
<Bob> <is a friend of> <Alice>.
<Bob> <is born on> <the 4th of July 1990>.
<Bob> <is interested in> <the Mona Lisa>.
<the Mona Lisa> <was created by> <Leonardo da Vinci>
<the video 'La Joconde à Washington'> <is about> <the Mona Lisa>
```

Esimerkkikoodi 1. Pienimuotoinen semanttinen verkko, joka kuvailee joitakin tietoja henkilöistä ja muista resursseista (Schreiber & Raimond 2014).

Kuten esimerkissä näkyy, resurssi ”Bob” on useassa RDF-lausunnossa subjektina ja resurssi ”the Mona Lisa” on yhdessä lausunnossa subjektina ja toisessa objektina. Koska resurssien URI voidaan sijoittaa lausunnossa objektiksi tai subjektiksi, on mahdollista löytää erilaisia yhteyksiä eri lausuntojen välillä. (Schreiber & Raimond 2014.) Tämä mahdollistaa myös sen, että RDF-tietomallia käytettäessä voivat useat lausunto-

ja lukevat sovellukset tai myös ihmiset päätellä annetuista tiedoista lisätietoja (Needleman 2001: 59). Edellä olevasta RDF-esimerkistä voi sen lukeva henkilö esimerkiksi päätellä täyden lauseen: ”Bob on ihminen ja hänen ystävänsä on Alice”. Kuvassa 1 visualisoidaan vielä aikaisemmin mainittu esimerkki informaalisena graafina.



Kuva 1. Informaalinen graafi aikaisemmin esitetystä RDF-esimerkistä. Graafissa kuvataan useita eri lausuntoja, tai triplejä, ja resurssien suhteita visuaalisesti. Graafi ei seuraa virallista RDF:n visualisointityyliä. (Schreiber & Raimond 2014.)

Kuten kuvasta huomaa, RDF-tietomallin avulla esitetty data on usein helpommin ymmärrettävissä visuaalisen esityksen avulla.

Yksi RDF:n tärkeimmistä ominaisuuksista on mahdollistaa resursseista luotavan metatiedon aggregointi eri tiedonlähteistä riippumatta siitä, onko tiedonlähde ihminen vai sovellus (Schreiber & Raimond 2014) . Kuka tahansa pystyy halutessaan luomaan jonkun URI:n esittämälle resurssille lisää metatietoja, josta RDF:ää käyttävät sovellukset voivat lukea, ymmärtää ja päätellä siitä lisätietoja. Koska RDF pohjautuu XML-merkkikieleen, on myös mahdollista, että metatiedot ovat dynaamisesti generoituja ja luettuja. (Workman 2016: 2.) RDF:n dynaamiset ja assosiatiiviset ominaisuudet johtuvat periaatteessa neljästä RDF:n ominaisuudesta: (1) Resursseja voidaan määritellä itsenäisesti, (2) RDF voidaan standardisoida sen jakamista varten, (3) RDF mahdollis-

taa pysyvien SPO-triplejen käytön ja (4) RDF mahdollistaa resurssien välillä periytyvät ominaisuudet (Workman 2016: 2–3).

RDF:llä esitetyt lausunnot voivat sisältää objekteina joko literaalisia eli atomisia arvoja, kuten merkkijonoja tai numeroita, tai resursseja, jotka viittaavat objektin olevan jokin muu esine, asia tai käsite. Näiden kahden eri tietotyypin suurin ero on se, että literaalisiin arvoihin ei ole mahdollista linkittää lisää metatietoja, kun taas resursseilla kuvastetaan lähestulkoon aina asiaa, johon on linkitetty muita metatietoja. Kuten jo aiemmin mainittu, resursseja kuvastaa aina URI tai mahdollisesti IRI (International Resource Identifier), joka on yleistetty versio URI:sta ja jossa pystytään käyttämään ASCII (American Standard Code for Information Interchange) -merkistön ulkopuolella olevia merkkejä (Schreiber & Raimond 2014). IRI voidaan sijoittaa myös mihin tahansa RDF-triplen kolmesta osasta, riippuen siitä, mitä lausunto sisältää. RDF-tietomalli ei ota kantaa siihen, mitä URI tai IRI esittää, mutta on kuitenkin mahdollista, että niille on annettu jokin tarkoitus tietyissä sanastoissa tai käytännöissä. (Schreiber & Raimond 2014.)

Literaaliset arvot, eli ”perusarvot” tai atomiset arvot, ovat lausunnossa olevia arvoja, joita ei esitetä IRI:nä. Literaaliset arvot voivat vain esiintyä lausunnon objektissa. Literaaleja ovat esimerkiksi merkkijonot, kuten ”Mona Lisa”, päivämäärät, kuten ”4.7.1990”, tai numerot, kuten ”27”. Literaalisiin arvoihin yhdistetään aina tietotyyppi, jonka avulla RDF:ää lukevat sovellukset tai ihmiset voivat ymmärtää arvon oikeassa formaatissa. Merkkijonoliteraaleihin voidaan myös lisätä kieltä ilmaiseva lisäarvo. Yleisesti RDF:n kanssa käytetään XML Schemassa määriteltyjä tietotyyppejä, mutta on myös mahdollista määrittää omia tietotyyppejä, joita ei XML Schemassa ole. (Schreiber & Raimond 2014.) Esimerkiksi literaali ”100”^^xsd:int kuvastaa kokonaislukua, joka on XML Schemassa määritellyssä kokonaislukuformaatissa. Vastaavasti literaali ”Leonardo da Vinci”^^xsd:string ilmaisee merkkijonoa. Esimerkkikoodissa 2 kuvaillaan jo aikaisemmin esitetty esimerkki pienestä RDF-graafista, tällä kertaa virallisessa RDF/XML-syntaksissa käyttäen IRI-tunnisteita ja literaalisia arvoja lausuntojen esittämiseen.

```
01    <?xml version="1.0" encoding="utf-8"?>
02    <rdf:RDF
03             xmlns:dcterms="http://purl.org/dc/terms/"
04             xmlns:foaf="http://xmlns.com/foaf/0.1/"
05             xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
06             xmlns:schema="http://schema.org/">
07       <rdf:Description rdf:about="http://example.org/bob#me">
08          <rdf:type
rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
```

```
09          <schema:birthDate
rdf:datatype="http://www.w3.org/2001/XMLSchema#date">1990-07-
04</schema:birthDate>
10          <foaf:knows rdf:resource="http://example.org/alice#me"/>
11          <foaf:topic_interest
rdf:resource="http://www.wikidata.org/entity/Q12418"/>
12       </rdf:Description>
13       <rdf:Description
rdf:about="http://www.wikidata.org/entity/Q12418">
14          <dcterms:title>Mona Lisa</dcterms:title>
15          <dcterms:creator
rdf:resource="http://dbpedia.org/resource/Leonardo_da_Vinci"/>
16       </rdf:Description>
17       <rdf:Description
rdf:about="http://data.europeana.eu/item/04802/243FA8618938F4117025F17
A8B813C5F9AA4D619">
18          <dcterms:subject
rdf:resource="http://www.wikidata.org/entity/Q12418"/>
19       </rdf:Description>
20    </rdf:RDF>
```

Esimerkkikoodi 2. Kuvan 1 esittämä semanttinen graafi kuvailtuna RDF/XML-syntaksissa käyttäen IRI-tunnisteita ja XML Scheman mukaisia literaaliarvoja (Schreiber & Raimond 2014).

Uusimmassa RDF-suosituksessa on aiemmin mainittujen RDF-tietomallin perusrakennuspalikoiden, eli literaalien ja IRI:en, lisäksi standardisoitu myös monimutkaisempia lausunnoissa ilmaistavia konsepteja. Perusmuodossa useimpia RDF-lausuntoja on mahdollista kirjoittaa IRI:en ja literaaliarvojen avulla. Joskus kuitenkin on hyödyllistä, että jokin konsepti, riippumatta siitä, onko se esimerkiksi fyysinen esine, ihminen tai mikä tahansa muu resurssi, voidaan ilmaista lausunnossa ilman, että sille luodaan tai annetaan uniikki tunniste. Resursseja, joilla ei ole RDF-tietomallissa uniikkia tunnistetta, kutsutaan tyhjiksi noodeiksi (englanniksi Blank Node). Tyhjiä noodeja voidaan verrata esimerkiksi algebrassa yksinkertaisiin muuttujiin, jotka esittävät jotakin asiaa kertomatta, mikä kyseisen asian arvo on. Lisäksi uusin W3C:n julkaisema RDF-suositus standardisoi uuden monikkograafimekanismin, joka mahdollistaa useiden RDF-graafien yhdistämisen yhteiseen IRI:iin. Monikkograafit esiintyivät alun perin RDF:n kyselykielen SPARQL:n standardoinnissa, mutta se yhdistettiin myöhemmin RDF-suositukseen. (Schreiber & Raimond 2014.)

Kuten jo aiemmin mainittu, RDF-tietomallissa metadataa tai konkreettista tietoa voi luoda kuka tahansa ihminen tai RDF-yhteensopiva järjestelmä. Tähän keskeisesti liittyvä konsepti on niin sanottu OWA (Open World Assumption) eli avoimen maailman oletus. OWA tarkoittaa käytännössä sitä, että jos jonkin väittämän todenmukaisuutta ei voida selvittää tietomallin kautta, joko suoraan tai päättelemällä, ei sen todenmukaisuu-

teen voida ottaa silloin kantaa. OWA:ssa oletetaan, että eksplisiittistä tietoa väittämän todenmukaisuudesta ei ole välttämättä vielä lisätty tietomalliin, jolloin tarkkaa vastausta todenmukaisuudesta ei voida muodostaa. (Drummond & Shearer 2006.) Tämän konseptin vuoksi RDF-tietomalli sisältää sivuvaikutuksen, jonka takia loogisesti toistensa vastakohtia kuvaavat tai toisensa kokonaan poislukevat lausunnot ovat täysin sallittuja RDF-tietomallissa, koska väittämää näiden loogisesta yhteydestä ei ole välttämättä vielä tiedossa tietomallissa. Esimerkiksi seuraavat RDF-lausunnot voidaan yhdistää samaan IRI:n, vaikka niitä lukevalle ihmiselle on selvää, että lausunnot loogisesti sulkevat toisensa pois: <henkilö> <on> <mies> ja <henkilö> <on> <nainen> tai <henkilö> <on aikuinen> <totta> ja <henkilö> <on aikuinen> <epätotta>. Loogisia yhteyksiä on kuitenkin mahdollista mallintaa erilaisilla ontologiakielillä, joiden avulla voidaan rajoittaa ja validoida tietomallin sisältämää tietoa.

Koska RDF-tietomalli ei ota kantaa siihen, mitä resurssien IRI:t esittävät, käytetään RDF:ää yleensä erilaisten sanastojen tai käytäntöjen yhteydessä. Nämä sanastot antavat semanttista tietoa resursseista. RDF-tietomalli sisältää RDF Schema -kielen tällaisien sanastojen luomiseen ja määrittelyyn. (Schreiber & Raimond 2014.) RDFS (RDF Schema) mahdollistaa yksinkertaisten ontologioiden mallintamisen. Vaikka XML Schemalla ja RDF Schemalla on samankaltaiset nimet, ne täyttävät erilaiset tehtävät semanttisessa verkossa. XML Schema rajoittaa dokumentissa olevien elementtien järjestystä ja yhdistelmiä, kun taas RDF Schema antaa tietoa siitä, kuinka elementit ja niiden arvot pitäisi tulkita. (Davies ym. 2003: 15.) RDFS käyttää käsitettä ”class”, eli suomeksi luokka, määrittämään kategorioita, joihin resurssit kuuluvat. Instanssin ja sen luokan välinen relaatio on ilmaistu ”type” eli tyyppi-ominaisuudella. RDFS:n avulla on mahdollista luoda hierarkioita luokkien ja aliluokkien ja myös tyyppien ja alityyppien välillä. Hierarkioiden lisäksi resurssien mahdollisia arvoja voidaan rajoittaa ”domain”- ja ”range”-ominaisuuksilla. (Schreiber & Raimond 2014.) Taulukossa 1 kuvaillaan RDF Schemassa määritelty sanasto, sen syntaksi ja lyhyt selvitys.

Taulukko 1. RDF Schema -sanaston eri rakenteet, niiden syntaksi ja lyhyt selitys niiden tarkoituksesta (Schreiber & Raimond 2014).

Sana	**Syntaksi**	**Selitys**
Class (luokka)	**C** rdf:type rdfs:Class	**C** (resurssi) on RDF-luokka
Property (luokka)	**P** rdf:type rdf:Property	**P** (resurssi) on RDF-ominaisuus
type (ominaisuus)	**I** rdf:type **C**	**I** (resurssi) on instanssi **C**:stä (luokka)

subClassOf (ominaisuus)	**C1** rdfs:subClassOf **C2**	**C1** (luokka) on **C2:n** aliluokka (luokka)
subPropertyOf (ominaisuus)	**P1** rdfs:subPropertyOf **P2**	**P1** (ominaisuus) on **P2**-aliominaisuus (ominaisuus)
domain (ominaisuus)	**P** rdfs:domain **C**	**P**:n (ominaisuus) domain on **C** (luokka)
range (ominaisuus)	**P** rdfs:range **C**	**P**:n (ominaisuus) range on **C** (luokka)

Kuten taulukosta 1 näkyy, RDFS:n sanasto on hyvin pienikokoinen ja tämän vuoksi RDFS:n kyky mallintaa ontologioita on rajoittunut.

3 RDF-tietomallin käyttö ja sitä tukevat teknologiat

3.1 RDF-tietomallin käyttökohteet

Vaikka RDF-tietomallin alkuperäinen tarkoitus oli toimia rakenteena semanttisen verkon tiedon ilmaisuun, se on vuosien varrella kehittynyt moneen eri käyttötarkoitukseen. Koska RDF-tietomalli pohjautuu metatietojen ja niiden semantiikan esittämiseen, on yksi tyypillisimmistä RDF:n käyttökohteista sitä hyödyntävän järjestelmän tietojen ja tietorakenteen säilytys tai esittäminen. Koska RDF-tietomalli esittää graafi-teorian mukaisia suunnattuja graafeja, se on yleensä parempi vaihtoehto ymmärtämisen mallintamiseen, esimerkiksi erilaisissa tekoälyä hyödyntävissä sovelluksissa kuin relaatio- ja ontologiatietokannat (Decker & Harth 2005: 71).

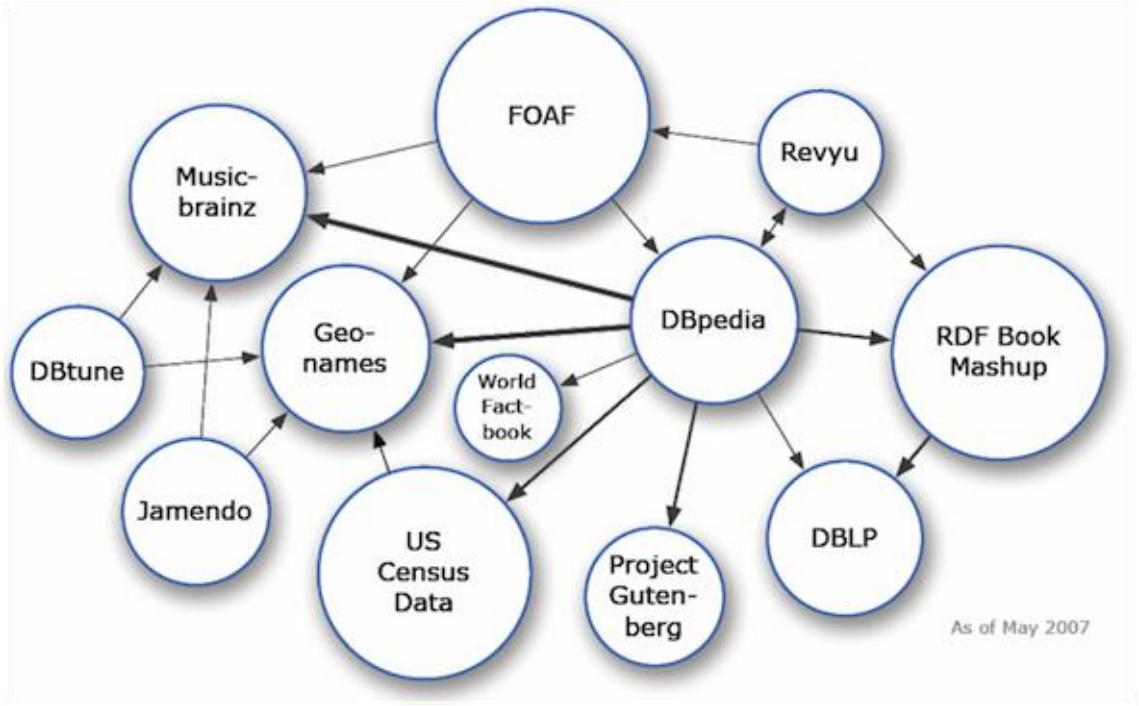
RDF-tietomallia käytetään myös esimerkiksi horisontaalisiin tietopalveluihin, tiedon integroitiin, hakukoneissa, verkko-oppimispalveluissa, verkkopalveluissa, multimediakokoelmien indeksoinnissa ja laitteiden yhteensovittamisessa (Antoniou & Harmelen 2008: 185). Tietomallia hyödynnetään myös sovelluksissa ja tutkimuksissa, joissa mallinnetaan luonnostaan verkostoksi muodostuvia käsitteitä, kuten esimerkiksi ihmisten välisiä sosiaalisia suhteita ja sosiaalisia piirejä, tieliikenneverkostoja ja liikenteen kulun malleja (Workman 2016: 39).

Myös isot internetyritykset ovat viime vuosikymmenen aikana alkaneet hyödyntää semanttista verkkoa useiden eri semanttisen verkon käyttöä edistävien trendien ja aloitteiden vuoksi. Esimerkiksi Googlen Knowledge Graph, Microsoftin Satori ja Facebookin Graph Search hyödyntävät semanttisen verkon teknologioita semanttisen tiedon ilmaisuun tai käsittelyyn. (Workman 2016: 39.) Yrityksien lisäksi myös tunnetut järjestöt

hyödyntävät semanttisen verkon teknologioita ja RDF-tietomallia. Esimerkiksi lisensointijärjestö Creative Commons käyttää RDF-tietomallia lisenssitietojen lisäämiseen mp3-tiedostoihin ja verkkosivustoihin, jotka hyödyntävät Creative Commons -lisenssejä. Myös Wikipediassa oleva tieto on avoimesti saatavissa RDF:nä DBpedia-projektin kautta (Workman 2016: 39).

Edellä mainittujen esimerkkien lisäksi monet vähemmän tunnetut järjestöt, organisaatiot ja projektit ovat ottaneet RDF-tietomallin käyttöön linkitetyn tiedon julkaisemiseen. Linked Open Data (LOD) -yhteisön ylläpitämä LOD cloud diagram -projekti on jo yli vuosikymmenen ajan kartoittanut eri tahojen julkaisemia avoimia linkitettyjä tietokantoja. Projektin tarkoituksena on seurata erilaisia tietokantoja, jotka on julkaistu avoimesti noudattaen LOD-yhteisön asettamia perusteita linkitetylle datalle, ja kartoittaa niiden väliset linkitykset ja assosiaatiot. (Linked Data - Connect Distributed Data across the Web; Abele ym. 2017.)

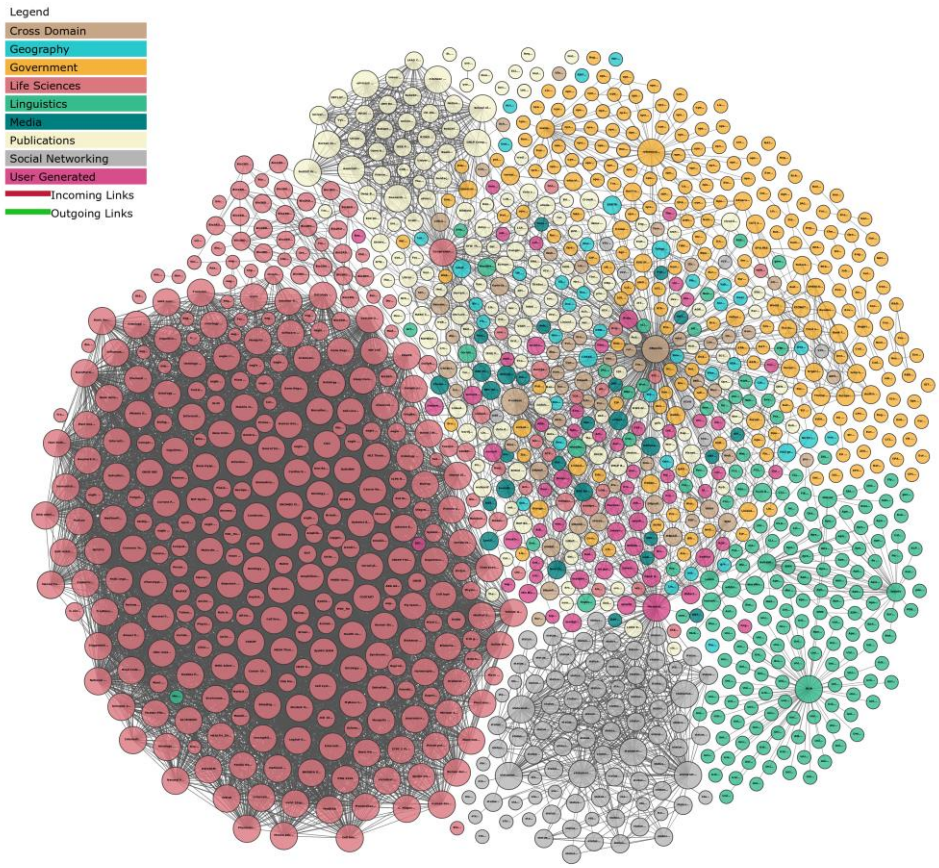
Ensimmäinen visuaalinen esitys eri tietokantojen yhteyksistä julkaistiin vuonna 2007, ja silloin LOD-yhteisö oli kartoittanut vain 12 linkitettyä tietokantaa. Kuvassa 2 esitetään vuonna 2007 avoimina julkaistut 12 tietokantaa ja niiden väliset linkitykset.



Kuva 2. LOD-yhteisön kartoittamat avoimet linkitetyt tietokannat, ja niiden väliset yhteydet vuoden 2007 toukokuussa. Kuvasta tulee ilmi, kuinka vähäisessä käytössä RDF-

tietomalli ja muut semanttisen verkon teknologiat olivat vielä noin 10 vuotta sitten. (Abele ym. 2017.)

LOD-yhteisö on päivittänyt kaaviota kuitenkin melkein vuosittain vuodesta 2007 lähtien, ja vuosikymmenen vaihteen jälkeen linkitettyjen tietomallien määrässä on nähtävissä huomattava nousu. Vuoden 2009 lopussa tietokantoja oli julkaistu vain 95, kun taas vuoden 2014 elokuussa tietokantojen määrä oli moninkertaistunut 570:een. Tietokantojen määrän kasvu jatkui myös tämän jälkeen, sillä viimeisin, vuoden 2017 elokuussa julkaistu erilaisten avointen linkitettyjen tietokantojen lukumäärä oli 1 163. (Linked Data - Connect Distributed Data across the Web; Abele ym. 2017.) Kuvassa 3 on esitetty graafina LOD-yhteisön kartoittamat avoimet tietokannat vuoden 2017 lopussa.



Kuva 3. LOD-yhteisön kartoittamien avoimien linkitettyjen tietokantojen yhteydet vuoden 2017 elokuussa. Tietokantojen määrä on kasvanut moninkertaiseksi viimeisen vuosikymmenen aikana, ja suurin osa avoimista tietokannoista voidaan luokitella muutamaan eri käyttötarkoitukseen. (Abele ym. 2017.)

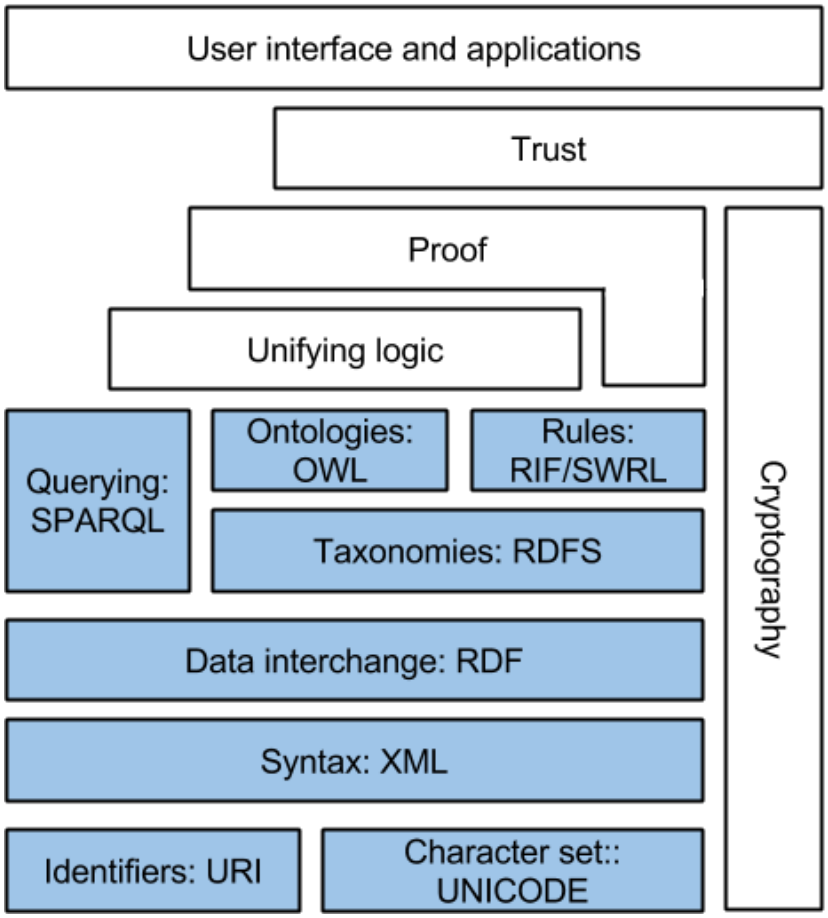
Kuten kuvasta 3 näkyy, semanttisen verkon avoin käyttö ja semanttisten määrittelyjen julkaisu ovat pitkälti keskittyneet julkiselle sektorille, etenkin erilaisten organisaatioiden

tutkimuskäyttöön. Varsinkin hyvinvointiin ja lääketieteeseen liittyvät semanttisen verkon tietokannat ovat yhteyksissä toisiinsa tiukasti.

Vaikka semanttisen verkon teknologioita on viime vuosikymmenen aikana ruvettu hyödyntämään enemmän ja semanttisen verkon visio on alkanut täyttyä paremmin, on silti hankala arvioida, jatkuuko sama trendi myös tulevaisuudessa. Semanttista verkkoa ja sen käytön laajenemista uhkaa varsinkin erilaisten tekoälyteknologioiden lisääntynyt käyttö, kuten esimerkiksi neuraaliverkot, koneoppiminen ja luonnollisen kielen käsittely. Tekoälyteknologioita hyödynnetään muun muassa jo olemassa olevan tiedon semanttiseen analysointiin automaattisesti, jolloin tiedosta koetetaan selvittää metatietoja suoraan ja näin poistetaan tarve erilliseen semantiikan määrittelyn, esimerkiksi RDF-tietomallin avulla, kun tietoa luodaan ja julkaistaan. (Anderson & Rainie 2010: 14 – 17.) Erilaisten tekoälysovellusten käyttö ei kuitenkaan suoraan sulje pois mahdollisuutta hyödyntää semanttisen verkon teknologioita, ja näitä kahta tietotekniikan aluetta on usein hyödynnetty yhdessä, varsinkin monissa tutkimushankkeissa. (Badr ym. 2010: 17 – 18). Voi siis hyvinkin olla, että erilaisten tekoälysovellusten laajeneva käyttö mahdollistaa myös semanttisen verkon teknologioiden paremman hyödyntämisen ja samalla edesauttaa semanttisen verkon laajenemista.

3.2 RDF:n käyttöä tukevat teknologiat

Kuten luvussa 2 jo mainittiin, RDF-tietomalli on vain yksi osa suurempaa ajatusta semanttisesta verkosta. Tarkemmin ilmaistuna RDF-tietomallin tehtävä semanttisen verkon teknologiapinossa on tiedon ilmaisu ja sen siirrettävyyden takaaminen. Melkein kaikissa RDF-tietomallia hyödyntävissä sovelluksissa käytetään myös muita semanttisen verkon teknologioita, sillä yksin RDF on hyödyllinen vain tiedon esittämiseen ja ilmaisuun eikä suoraan esimerkiksi tiedon hakemiseen tai ymmärtämisen ilmaisuun. Luvussa 2 mainittiin myös jo RDFS, joka määrittelee sanaston yksinkertaisten ontologioiden luomiseen. RDFS ei kuitenkaan ole suoraan osana RDF-suositusta, vaan se on oma W3C:n standardisoima suositus (Schreiber & Raimond 2014). Kuvassa 4 näkyy semanttisen verkon teknologiapino visualisoituna tasoittain mukaan lukien RDF ja RDFS sekä muita usein hyödynnettyjä semanttisen verkon teknologioita.



Kuva 4. Yleisimmin käytetty ja hyväksytty semanttisen verkon teknologiapino. Sinisellä merkityt semanttisen verkon osat ovat jo standardisoituja tai niiden standardisoiminen on työn alla. (Alsharif 2013.)

Kuvassa 4 esitetyistä semanttisen verkon teknologioista kaksi yleisimmin RDF-tietomallin kanssa käytettyä ovat SPARQL-kyselykieli (SPARQL Protocol and RDF Query Language) ja Web Ontology Language (OWL), jota käytetään ontologioiden mallinnuksen. Näitä teknologioita käytetään monessa RDF-yhteensopivassa sovelluksessa, sillä ne helpottavat, ja joissakin tapauksissa mahdollistavat, RDF-tietomallin tärkeimpien piirteiden käyttämisen reaalimaailman sovelluksissa. W3C suosittelee kummankin teknologian käyttöä ja hyödyntämistä semanttisessa verkossa ja sen yhteydessä ja kummastakin teknologiasta on julkaistu virallinen W3C:n suositus.

3.2.1 SPARQL-kyselykieli

SPARQL on RDF:ää varten kehitetty kyselykieli, jolla voidaan hakea ja käsitellä RDF-yhteensopivista tietokannoista RDF-tietomallilla esitettyä tietoa (Prud'hommeaux & Seaborne 2008). W3C kehitti ensimmäisen version SPARQL-suosituksesta vuonna

2008 SPARQL 1.0-versiossa, ja suositusta päivitettiin myöhemmin vuonna 2013. SPARQL-kyselykieli perustuu graafi-kaavojen sovittamiseen, joista yksinkertaisin on triple-kaava, joka ilmaistaan SPARQL-syntaksissa samalla tavalla kuin triple RDF:ssä, mutta se voi sisältää muuttujia (Antoniou & Harmelen 2008: 105). SPARQL-kysely voi muodostua triple (eli lausunto) -kaavojen lisäksi myös konjunktioista, disjunktioista sekä valinnaisista triple-kaavoista (Prud'hommeaux & Seaborne 2008). Triple-kaavoja yhdistämällä on mahdollista luoda kyselyitä, joiden vastauksena on kaavaa tarkasti vastaava graafi (Antoniou & Harmelen 2008: 105).

SPARQL-kyselykieli mahdollistaa neljän erilaisen kyselyn käyttämisen tiedon lukemiseen RDF-tietokannasta, ja niistä jokainen soveltuu eri tarkoitukseen:

- **SELECT**-kysely mahdollistaa muuttujasidontojen hakemisen taulukkoon RDF-tietokannasta.
- **CONSTRUCT**-kyselyllä voidaan hakea tietoa RDF-tietokannasta RDF-muodossa, jolloin kyselyn tuloksista muodostetaan kyselyä vastaava graafi.
- **ASK**-kysely tuottaa yksinkertaisen Tosi/Epätosi-vastauksen SPARQL-kyselyyn.
- **DESCRIBE**-kysely tuottaa RDF-muodossa olevaa tietoa, jonka sisällöstä päättää RDF-tietokanta.

Kaikkiin näihin neljään kyselyyn liitetään SPARQL-**WHERE**-lauseke, joka määrittelee, millä kriteereillä kysely tuottaa hakutulokset. (Prud'hommeaux & Seaborne 2008.) WHERE-lausekkeen lisäksi kysely koostuu monista muista lausekkeista, jotka rajoittavat kyselytuloksia tai helpottavat kyselyiden kirjoittamista. Näistä lausekkeista monet ovat vapaaehtoisia, eikä niitä usein tarvitse käyttää haluttujen tuloksien saamiseksi kyselyllä. Esimerkksi SELECT-kyselyiden rakenne on yleensä seuraava:

- **PREFIX** Nimiavaruuksien prefiksien määrittely.
- **SELECT** Kyselytuloksista valitaan haluttu tieto.
- **FROM** Mistä tietokannasta vastaukset haetaan.
- **WHERE** Millä kriteereillä vastaukset luodaan.
- **ORDER BY, LIMIT ja muut SPARQL-muuttujat.** Muuttujat, jotka muokkaavat kyselytuloksia niiden saamisen jälkeen esimerkiksi rajoittamalla niiden määrää tai muuttamalla niiden järjestystä.

Näiden lausekkeiden lisäksi SPARQL-kyselykielen sanasto sisältää useita muita lausekkeita, joita käytetään pääasiassa kyselyiden ja hakutulosten tarkentamiseen. (Wood 2013: 100 – 101.) Esimerkkikoodissa 3 kuvaillaan yksinkertainen SELECT-kysely, jos-

sa hyödynnetään aikaisemmin mainittuja SPARQL-kyselykielen eri lausekkeita henkilöiden paikkatietojen hakemiseen.

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix pos: <http://www.w3.org/2003/01/geo/wgs84_pos#>

select ?name ?latitude ?longitude
from <http://3roundstones.com/dave/me.rdf>1
from <http://semanticweb.org/wiki/Special:ExportRDF/Michael_Hausenblas>
where {
     ?person foaf:name ?name ;
          foaf:based_near ?near .
     ?near pos:lat ?latitude ;
          pos:long ?longitude .
}
LIMIT 10
```

Esimerkkikoodi 3. Yksinkertainen SPARQL-kysely, jossa haetaan taulukko henkilöiden paikkatiedoista FROM-lausekkeen määrittelemistä tietokannoista (Wood 2013: 101).

SPARQL-kyselykielen WHERE-lausekkeen kriteerejä voidaan tarkentaa esimerkiksi *FILTER*-lausekkeella, joka poistaa hakutuloksista kaikki sellaiset vastaukset, jotka eivät täytä lausekkeen määrittelemiä ehtoja. Yksi WHERE-lauseke voi sisältää useita FILTER-lausekkeita, joista jokaisen ehtojen tulee täyttyä, jotta yksittäinen vastaus voidaan palauttaa osana hakutuloksia. FILTER-lausekkeet voivat testata yksittäisen muuttujan ominaisuuksia sen tietotyypistä riippuen, esimerkiksi suorittamalla yksinkertaisia totuusarvojen testauksia, numeroiden vertailuja, merkkijonojen regular expression -testauksia ja päivämäärien vertailuja. (Wood 2013: 110 – 111.) Esimerkkikoodissa 4 demonstroidaan yksinkertaisen FILTER-lausekkeen käyttöä hakutuloksien rajoittamiseen englanninkielisiin vastauksiin.

```
select ?abstract
where {
    <http://dbpedia.org/resource/Linked_Data> dbpedia-owl:abstract ?abstract .
    FILTER (lang(?abstract) = "en")
}
```

Esimerkkikoodi 4. Esimerkki FILTER-lausekkeen käytöstä muuttujien kielen testaamiseen (Wood 2013: 111).

Toinen WHERE-lausekkeen yhteydessä hyödyllinen ja usein käytetty lauseke on *OPTIONAL*. FILTER-lausekkeella rajoitetaan hakutuloksiin tulevia vastauksia, mutta OP-

TIONAL-lauseke toimii päinvastoin, sillä sen avulla on mahdollista tuoda hakutuloksiin lisää vastauksia. Kaikki suoraan WHERE-lausekkeen alle määritellyt triple-kaavat tulee saada kyselyssä sovitetuksi, jotta tietty triple voidaan sisällyttää hakutuloksiin. OPTIONAL-lauseke kuitenkin mahdollistaa ei-pakollisten triplejen palauttamisen, mikä tarkoittaa, että OPTIONAL-lausekkeen sisälle määriteltyjä triple-kaavoja ei ole pakko saada sovitetuksi, jotta yksittäinen triple voidaan sisällyttää hakutuloksiin, mutta jos jokin triple vastaa näitä kaavoja, pakollisten kaavojen lisäksi, se lisätään hakutuloksiin mukaan. OPTIONAL-lausekkeet mahdollistavat siis käytännössä dynaamisempien kyselyiden luonnin. (Wood 2013: 109; Antoniou & Harmelen 2008: 107 – 108.) Esimerkkikoodissa 5 esitetään OPTIONAL-lausekkeen käyttöä hakemalla luennoitsijan nimen lisäksi hänen sähköpostiosoitteensa, jos se löytyy tietokannasta.

```
prefix uni: <http://www.mydomain.org/uni-ns#>
select ?name ?email
where {
    ?x rdf:type uni:Lecturer ;
          uni:name ?name .
    OPTIONAL { ?x uni:email ?email }
}
```

Esimerkkikoodi 5. Esimerkki OPTIONAL-lausekkeen käytöstä ei-pakollisten arvojen hakemiseen (Antoniou & Harmelen 2008: 108).

SPARQL on yleisin RDF:n kanssa käytetty kyselykieli, ja se on kehittynyt epäviralliseksi standardiksi RDF-tietomallin kyselyihin, vaikka muitakin kyselykieliä on olemassa. Suurin osa RDF-yhteensopivista sovelluksista, jotka tukevat myös RDF-tietomallin kyselyitä, käyttävät SPARQL:ää kyselykielenä, ja vain harvat sovellukset tarjoavat tukea muihin kyselykieliin. Muihin kyselikieliin kuuluvat esimerkiksi SeRQL ja RQL, joita on kehitetty pidempään kuin SPARQL:ää, mutta pienempi määrä sovelluksia tukee näitä kyselykieliä, mikä hankaloittaa kyselyiden uudelleenkäyttöä sovellusten välillä. (Antoniou & Harmelen 2008: 105.)

3.2.2 OWL-ontologiakieli

Toinen aikaisemmin mainituista RDF-tietomallin kanssa yleisesti käytetyistä teknologioista on OWL-ontologiakieli. OWL on tarkoitettu käytettäväksi sovelluksissa, joiden täytyy prosessoida tiedon sisältöä eikä vain esittää tietoa käyttäjille. OWL mahdollistaa

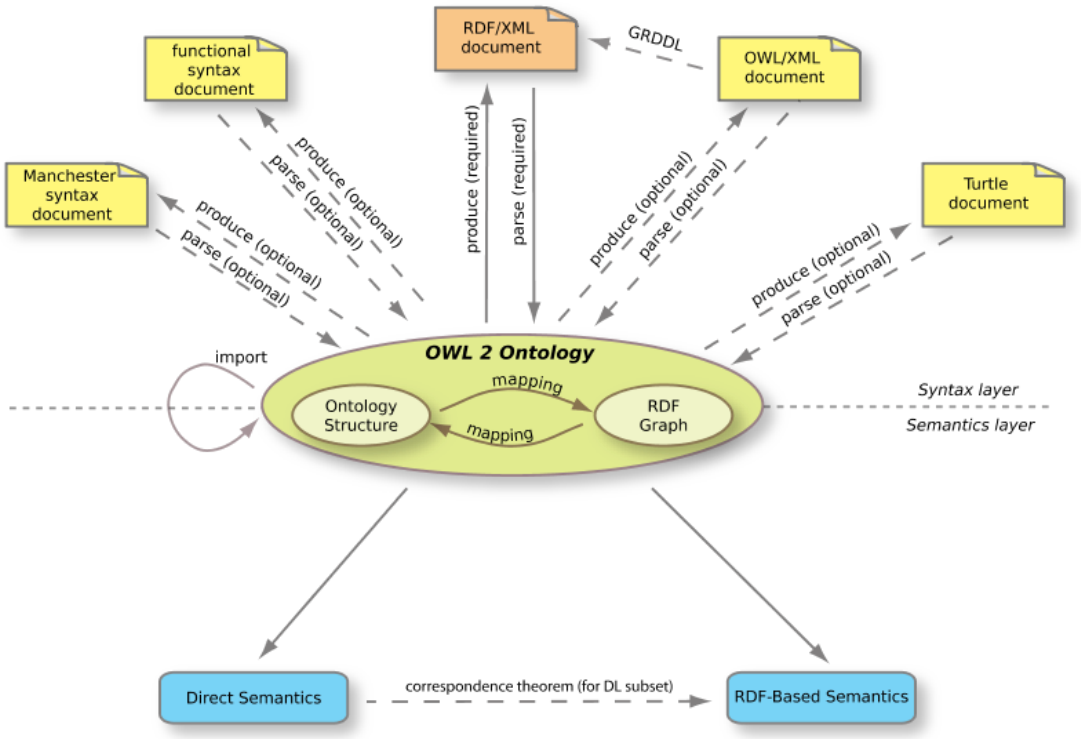
parempia konetulkintamahdollisuuksia kuin esimerkiksi XML, RDF ja RDFS määrittelemällä formaalisen semantiikan lisäsanaston. (McGuinness & Harmelen 2004.)

OWL voidaan jakaa kolmeen eri määrittelytasoon, joiden avulla on mahdollista ilmaista ontologia eritasoisilla ilmaisukyvykkyyksillä: OWL Lite, OWL DL ja OWL Full (McGuinness & Harmelen 2004). OWL:n täyttä ontologiakieltä kutsutaan OWL Fulliksi ja se hyödyntää kaikkia OWL:n primitiivejä. OWL Full on täysin yhteensopiva RDF:n kanssa, mikä tarkoittaa, että kaikki validit RDF-dokumentit ovat myös valideja OWL-dokumentteja syntaktisesti ja semanttisesti. OWL Full on kuitenkin ilmaisukyvyiltään niin tehokas, että sitä on melkein mahdotonta hyödyntää reaalimaailman sovelluksissa. Jotta OWL:a voidaan hyödyntää oikeissa sovelluksissa, on kielen tarjoamia ilmaisukyvykkyyksiä rajoitettava. Tätä varten on kehitetty OWL DL ja OWL Lite, jotka ovat OWL Fullin alikieliä ja tarjoavat rajoitetumman ontologian ilmaisukyvykkyyden, jotta niitä hyödyntävät sovellukset voivat säilyttää realistisen päättelytehokkuuden. (Antoniou & Harmelen 2008: 118 – 119).

OWL:n kehitti W3C:n Web Ontology Working Group -työryhmä, kun se huomasi samantapaisten käyttötilanteiden toistuvan useasti semanttisen verkon ontologiamäärittelytarpeissa, mutta käyttötilanteiden mahdollistaminen vaati enemmän esittämisvoimaa, kuin mitä RDF tai RDFS tarjosivat. RDF:n ja RDFS:n esittämisvoima on rajoittunut, sillä RDF on rajoitettu binäärisiin termeihin ja RDFS on rajoitettu aliluokkien ja ominaisuuksien hierarkiaan, mutta RDFS pystyy myös ilmaisemaan domain ja range -yhteyksiä ominaisuuksien välillä. (Antoniou & Harmelen 2008: 113). OWL:n ja RDFS:n käyttötarkoitus on kuitenkin pitkälti sama, eli luokkien, ominaisuuksien ja niiden välisten suhteiden mallinnus. Vaikka RDFS:llä voidaan mallintaa joitakin ontologisia tietoja, sen määrittelyistä puuttuu tärkeitä ominaisuuksia, kuten esimerkiksi ominaisuuksien lokaali näkyvyysalue, luokkien irtonaisuus, luokkien yhdistäminen joukko-operaattoreilla, kardinaliteetin rajoittaminen ja ominaisuuksien erikoistunnuspiirteet (esimerkiksi transitiivisuus, käänteisyys ja ainutlaatuisuus). Näiden ominaisuuksien puuttumisen takia semanttisen verkon ontologiamallinnukseen tarvittiin tehokkaampi kieli, jonka avulla voidaan mallintaa rikkaampia ontologioita ja parantaa päättelykykyä. (Antoniou & Harmelen 2008: 113; Yu 2007: 95.)

Ennen OWL:n kehittämistä useat tutkimusryhmät Euroopassa ja Yhdysvalloissa olivat tunnistaneet samanlaisia ontologiamäärittelytarpeita, ja niistä oli kehittynyt kaksi merkittävintä ontologiakieltä, DAML (DARPA Agent Markup Language) ja OIL (Ontology Infe-

rence Layer), joita usein käytettiin yhdessä nimityksen DAML + OIL yhteydessä. Web Ontology Working Group yhdisti virallisesti DAML + OIL -aloitteet ensimmäiseen OWL-standardiin vuonna 2004, ja OWL onkin suora jatke DAML + OIL -kehitykselle. (Antoniou & Harmelen 2008: 113; Workman 2016: 4). OWL on myös kehitetty suoraan RDFS:n päälle, jolloin kaikki RDFS:n määrittelemät luokat ja ominaisuudet ovat myös käytettävissä OWL-kielessä. Nykyään OWL on jo useamman standardin julkaisun jälkeen suosituin ontologiakieli, ja sitä hyödynnetään monissa RDF-tietomallia käyttävissä sovelluksissa. (Yu 2007: 95.) Kuvassa 5 esitellään OWL:n yleinen rakenne, siten kuin se on kuvailtu uusimmassa OWL 2 -standardissa.



Kuva 5. OWL:n yleinen rakenne (OWL 2 Web Ontology Language Document Overview 2012).

Kuvassa esitetään OWL:n uusimman standardin pääasialliset rakennuspalikat, joista ontologiakieli koostuu. Ontologiakielen osat on jaettu syntaksi- ja semantiikkatasoille, ja keskimmäinen alue esittää abstraktia käsitettä ontologiasta. OWL:n käyttäjien tarvitsee harvoin miettiä ontologiakieltä näin yleiseltä tasolta, sillä yleensä käytössä on vain yksi syntaksi ja yksi semantiikka. (OWL 2 Web Ontology Language Document Overview 2012.)

4 RDF-tietomallin visualisointi ja sen luonnolliset haasteet

Koska insinöörityö käsittelee pääasiassa RDF-tietomallin visualisointia, on tärkeää käydä läpi myös RDF-tietomallin luonnollisia ominaisuuksia, tai myös usein esiintyviä käytäntöjä, jotka vaikeuttavat sen visualisointia. Kuten jo aikaisemmin mainittu, RDF-tietomalli kuvaa tietoa suunnatussa ja nimitetyssä graafissa. Usein graafin esittämä tieto kuitenkin käsitellään jollain tavoin sovelluksen käyttöliittymätasolla ennen sen esittämistä käyttäjälle, jotta tieto on helpommin käyttäjän ymmärrettävissä (Graves 2015: 152). On esimerkiksi yleistä, että IRI:en tai URI:en sijasta käyttäjälle näytetään niihin liittyvä rdfs:label-tieto tai jonkin muun predikaatin literaaliarvo.

Virallisesti raakaa, eli käsittelemätöntä, RDF-tietomallilla mallinnettua tietoa kuitenkin yleisesti esitetään jossakin RDF:n serialisointia tukevassa tekstiformaatissa, kuten esimerkiksi RDF/XML. (Dokulil & Katreniakova 2009a: 459). RDF/XML ei ole ainut RDF:n serialisointiformaatti, vaan sen rinnalle on kehitetty myös eri syntaktisia esitystapoja. Serialisoidun tiedon lukeminen ja ymmärtäminen saattaa olla usein hankalaa, varsinkin suurikokoisissa ja komplekseissa tietomalleissa, sillä lukija joutuu käytännössä hahmottamaan tiedon esittämän RDF-graafin ja sen sisältämät yhteydet mielessään. Ihmisellä onkin huomattavasti tehokkaampi kyky ymmärtää tietoa, kun se esitetään visuaalisesti, verrattuna siihen, että sama tieto on esitetty esimerkiksi tekstiformaatissa tai numeraalisesti (Deligiannidis ym. 2007: 39). Tämän vuoksi useissa tilanteissa voi olla hyödyllistä esittää myös raakaa RDF:ää visuaalisesti, riippumatta sen sisällöstä tai käyttötarkoituksesta (Dokulil & Katreniakova 2009a: 459).

RDF-tietomallin visualisointi ei kuitenkaan yleensä ole helppoa. RDF-tietomallin esittämä semanttinen tieto on usein osana suurikokoista ja pitkälti yhteen liitettyä, sillä se on semanttisen tiedon tarkoituskin. RDF:nä esitetty tieto ei myöskään usein seuraa tarkasti määriteltyä skeemaa, vaan tieto saattaa olla muodoltaan hajautunutta. Näiden seikkojen takia RDF-tietomallin visualisointi ei skaalaudu hyvin isojen tietokantojen kanssa. (Li ym. 2012: 1293.)

4.1 Graafien visualisointistrategiat

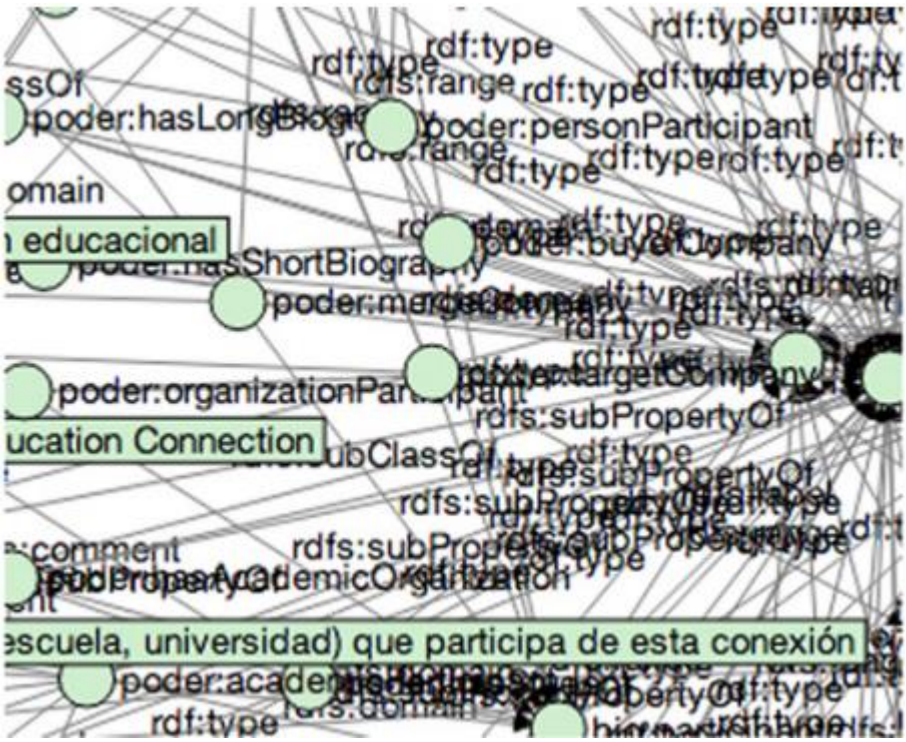
Koska RDF-graafien visualisointi on luonnollisesti hankalaa, on kehitetty erilaisia strategioita visualisointia varten. Jotkut näistä visualisointistrategioista keskittyvät enem-

män ongelmien kiertämiseen, kun taas toiset keskittyvät tietomallin esittämän tiedon yleiskuvan hahmottamiseen.

4.1.1 Display-at-once-strategia

Display-at-once-strategian toimintaperiaate on seuraava: koko RDF-tietoa sisältävä dokumentti analysoidaan ensiksi ja sen jälkeen siitä luodaan graafi, jossa esitetään kaikki alkuperäisen dokumentin sisältämät triplet (Athanassiades ym. 2009: 153). Subject- ja object-sijainnissa oleva tieto visualisoidaan joko suorakulmioina, jos tieto esittää literiaaliarvoa, tai ovaalina ympyränä, kun tieto esittää URI:a (Li ym. 2012: 1293). Triplejen predikaattiarvot esitetään noodien (engl. node), eli siis ympyröiden ja suorakulmioiden, välisinä suunnattuina viivoina. Tämän strategian avulla URI:t, joista lähtee tai joihin osoittaa monta nuolta, visualisoidaan vain kerran, jolloin esitettyjen subject- ja object-tietojen määrää voidaan vähentää. (Athanassiades ym. 2009: 153.)

Suurin hyöty tässä strategiassa on, että sen avulla voidaan visualisoida RDF-tietoa kokonaisuudessaan menettämättä yhtään tripleä (Athanassiades ym. 2009: 153). Tämä strategia ei kuitenkaan ratkaise aikaisemmin mainittua ongelmaa, eli RDF:n visualisoinnin huonoa skaalautumista. Visualisoidun graafin kasvaessa sen lukeminen vaikeutuu, kun esitettyjen noodien ja niiden välisten nuolien määrä kasvaa liian suureksi (Athanassiades ym. 2009: 153). Strategia säilyttää kuitenkin käyttökelpoisuutensa, kun esitettävän tiedon määrä on pieni. Kuvassa 6 annetaan esimerkki display-at-once-strategialla visualisoidusta graafista, jonka koko on kasvanut liian suureksi.



Kuva 6. Display-at-once-strategialla visualisoitu RDF-graafi, jossa esitetään suuri määrä noodeja, eli subject- ja object-sijainnissa olevaa tietoa, ja niiden välisiä suhteita (Graves 2015: 153).

Vaikka kuvassa ei näy koko graafia, on silti selvää, että graafi on päässyt kasvamaan liian suureksi display-at-once-strategialla visualisointia varten. Kuvassa olevasta graafista on melkein mahdotonta ymmärtää tai lukea sen kuvaamaa tietoa.

4.1.2 Navigaatiostrategia

Navigaatiostrategia perustuu visualisoinnin keskittämiseen valittuun noodiin, joka toimii keskimmäisenä noodina (Athanassiades ym. 2009: 153). Yleensä keskimmäisen noodin valinnan suorittaa suoraan visualisointityökalun käyttäjä, tai noodin valinta tehdään käyttäjän aiemmin toimenpiteiden perusteella. Visualisointi tällä strategialla esittää keskimmäisen noodin lisäksi myös kaikki muut triplet, joissa tämä noodi esiintyy subject-sijainnissa (Li ym. 2012: 1293).

Muita osia graafista voidaan avata ja visualisoida interaktiivisesti. Kaikki jo visualisoidut noodit, jotka on jo visualisoitu object-sijainnissa, voidaan laajentaa samanlaisiksi subject-graafeiksi kuin alkuperäiselle keskimmäiselle noodille luotu graafi. Käyttäjä voi näin ollen hallita, mitä osia graafista visualisoidaan. (Athanassiades ym. 2009: 153.)

Navigaatiostrategia tarjoaa dynaamisemman lähestymistavan graafien visualisointiin. Display-at-once-stategian luonnollinen heikkous suurien graafien visualisoinnissa käytännössä kierretään antamalla käyttäjälle vastuu visualisoidun tiedon valitsemisessa. Navigaatiostrategialla on myös mahdollista visualisoida monimutkaisempia ja suurempia RDF-tiedostoja, sillä koko tiedoston visualisointi kerralla ei ole pakollista. (Athanassiades ym. 2009: 153.) Tämäkään strategia ei kuitenkaan poista mahdollisuutta, että visualisoitu graafi menee lukukelvottomaksi suuren tietomäärän takia. Käyttäjä voi halutessaan navigoida monen noodin läpi ja tämän seurauksena avata ison määrän noodeja ja niiden yhteyksiä.

Navigaatiostrategialla on kuitenkin joitakin heikkouksia display-at-once-strategiaan verrattuna. Suurimpana heikkoutena tämä strategia tekee mahdottomaksi pienempien graafien visualisoinnin kerralla, ja tällaisten graafien kokonainen visualisointi vaatiikin käyttäjältä huomattavan työn, koska hän joutuu käytännössä navigoimaan koko graafin läpi. (Athanassiades ym. 2009: 153.)

4.1.3 Graph-centric-at-once -strategia

Graph-centric-at-once-strategiassa periaatteessa yhdistetään kaksi aikaisemmin mainittua strategiaa. Tässä strategiassa visualisoidaan kaikki tiedoston sisältämät triplet kerralla aloittaen jostakin keskimmäisestä noodista, joka on jollakin tavoin eksplisiittisesti tai satunnaisesti valittu. Tämän jälkeen noodit, jotka ovat object-sijainnissa keskimmäiseen noodiin katsottuna, avataan, jos ne ovat mukana muissakin graafin tripleistä. (Athanassiades ym. 2009: 153.)

Verrattuna kahteen aikaisempaan strategiaan graph-centric-at-once-strategiassa avattaessa noodia uudelleen toistuvat resurssit piirretään aina uusiksi eikä uusia nuolia piirretä vanhoihin noodeihin. Tällä metodilla poistetaan nuolien päällekkäisyys, jolloin lopputuloksena on helpommin luettava graafi. (Athanassiades ym. 2009: 153; Li ym. 2012: 1294.) Metodin heikkoutena on kuitenkin se, että tietystä resurssista lähteviä, tai siihen tulevia, yhteyksiä on hankala kartoittaa, sillä samaa resurssia esittäviä noodeja saattaa olla useita. Tämän lisäksi isojen RDF-tiedostojen käsittely vaatii huomattavasti enemmän prosessointitehoa tätä strategiaa käytettäessä, suuren noodimäärän takia (Athanassiades ym. 2009: 153).

Graph-centric-at-once-strategia toimii siis hyvin pienissä ja keskisuurissa RDF-tiedostoissa, mutta kärsii silti ongelmista isommissa tiedostoissa. Strategiaa hyödyntäen voidaan yhdistää joitakin osia kahden aikaisemman strategian hyvistä puolista ja samalla lieventää joitakin haasteita, jotka ilmenevät isoissa RDF-tiedostoissa (Athanassiades ym. 2009: 153). Tällä strategialla käyttäjät saavat enemmän tietoa näkyviin kerralla kuin navigaatiostrategialla, mutta säilyttävät samalla osittain mahdollisuuden valita, mitä tietoa visualisoidaan, toisin kuin display-at-once-strategialla.

4.2 Graafin sotkeutuminen ja sen syyt

Visualisoitavasta tiedostosta ja sen määrästä riippuen on tärkeää valita oikea strategia visualisointiin, jotta voidaan ennaltaehkäistä joitakin RDF-visualisoinnin haasteita. Kaikki aikaisemmin mainitut visualisointistrategiat kuitenkin kärsivät samasta ongelmasta: kun liian suurta määrää tietoa koetetaan visualisoida kerralla, on hyvin todennäköistä, että visualisoitu graafi sotkeutuu lukukelvottomaksi eikä sen esittämää tietoa voi enää visuaalisesti ymmärtää.

Oikea strategiavalintakaan ei siis välttämättä estä graafin sotkeutumista, vaan on hyödynnettävä muitakin tekniikoita sotkeutumisen estämiseksi. Strategiavalinnasta riippumatta RDF:ää visualisoivan ohjelman pitäisi mahdollistaa graafin tutkinta ja tarkastelu (Graves 2015: 152). Jotta käyttäjä voi tutkia graafia luonnollisesti ja sujuvasti, voidaan yleisesti olettaa, että työkalun täytyy myös silloin seurata Visual Information Seeking Mantran ajattelua, eli käytännössä tämä tarkoittaa yleiskuvan esittämistä ensiksi, sen jälkeen visualisoinnin tarkentamista ja tiedon rajoittamista ja lopuksi tarkennetun tiedon esittämistä käyttäjän pyynnöstä (Graves 2015: 152).

Tiedon sekoittuminen on ongelma, joka vaivaa useita visualisointiteknologioita RDF:n visualisoinnin lisäksi. Tämän takia ongelmaa on tutkittu useissa eri konteksteissa ja käyttötarkoituksissa, mukaan lukien graafien visualisoinnissa. (Graves 2015: 152.) RDF:n visualisointi on kuitenkin jossain mielessä erikoistapaus, joka poikkeaa normaalista graafivisualisoinnista, sillä geneerisissä graafeissa harvoin esitetetään noodien välisien yhteyksien semantiikkaa tai niiden merkitystä. Siksi osa tekniikoista, joita käytetään yleisten graafien sekoittumisen estämiseen, ei päde tai edes toimi RDF-graafien visualisoinnissa.

Yksi yleisesti käytetty tekniikka graafien sotkeutumisen estämiseen on noodien yhdistäminen ryhmiin ja samanlaisten kaarien (engl. edge), eli noodien välisten yhteyksien, yhdistäminen (Graves 2015: 153; Dokulil & Katreniakova 2009b: 62). Tämä voi olla geneerisissä graafeissa erittäin tehokas tapa vähentää graafin sotkeutumista, sillä parhaissa tilanteissa suuri osa noodeista ja kaarista voidaan yhdistää ilman, että tieto menettää merkityksensä. RDF:n visualisoinnissa tämä tekniikka ei kuitenkaan usein toimi, sillä uniikeille resursseille ja niiden välisille yhteyksille on hankalaa löytää kriteereitä, joilla yhdistää elementtejä ilman, että ne menettävät semantiikkansa (Graves 2015: 153).

RDF-graafien kaarien visualisointia hankaloittavat muutkin seikat. Esimerkiksi sekoittuneessa graafissa on hankalaa hahmottaa kaarien suuntaus, jolloin graafin esittämät RDF-lausunnot menettävät merkityksensä, kun lausunnon subject- ja object-sijainnit sekoittuvat. Toinen yleinen ongelma on, että kaarien tulisi aina esittää jotakin predikaatin arvoa, esimerkiksi nimikkeellä kaaren yläpuolella. Tämä vie huomattavasti tilaa jo valmiiksi helposti sekoittuvalta piirtoalueelta. Lisäksi RDF-tietomallin esittämä tieto hyödyntää usein ontologioita, kuten luvussa 3 käytiin läpi, ja tämän vuoksi visualisoitu graafi saattaa sisältää paljon samoilla predikaateilla esitettyjä yhteyksiä, jotka juontavat juurensa ontologiamäärittelyyn. Näiden ongelmien takia visualisoinnin kannalta on tärkeää, että yksittäiset noodit ja kaaret voidaan selvästi erottaa toisistaan. (Graves 2015: 153.)

4.3 Sotkeutumisen estäminen käytännössä

Visual Information Seeking Mantran ajattelua seuraten on mahdollista vähentää graafin sotkeutumista ilman, että graafin esittämää tietoa ja semantiikkaa menetetään kokonaan visualisoinnin piiristä. Yksi tapa vaikuttaa graafin selkeyteen, tai sen sotkuisuuteen, on visualisointistrategian lisäksi oikean asetelman valitseminen graafille. Graafiastelma ei käytännössä vaikuta siihen, mitä tai millaista tietoa esitetään, vaan graafiasetelman tarkoitus on päättää, miten yksittäiset noodit sijoitetaan vapaana olevalle piirtoalueelle.

Graafiasetelmia on periaatteessa melkein loputtomasti, sillä kuka tahansa voi kehittää oman algoritmin pisteiden (eli tässä tapauksessa noodien) asetteluun äärelliselle koordinaatistolle, ja jokaisen asetelman tehokkuus visualisoida RDF-tietomallia vaihtelee

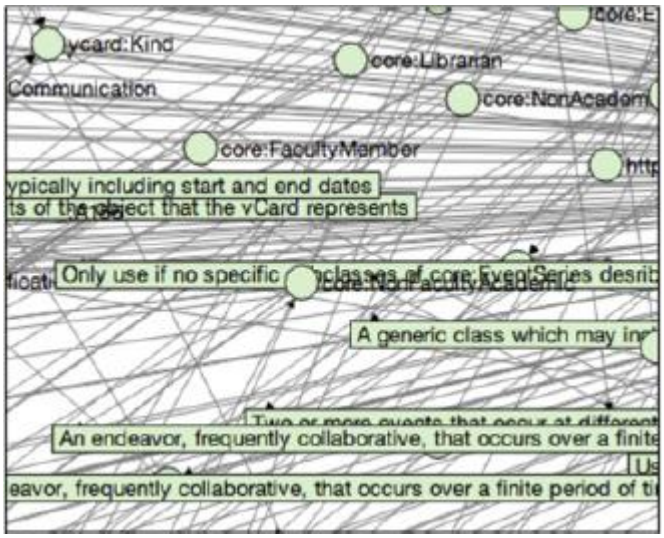
tietomallin käyttötarkoituksesta riippuen. On kuitenkin useita asetelmia, jotka on todistettu tehokkaiksi yleisissä RDF:n visualisointitapauksissa. Yleensä on silti suotavaa, että käyttötilanteesta riippuen erilaisia asetelmia testattaisiin niistä parhaan löytämiseksi (Athanassiades ym. 2009: 155). Yleisimpiin asetelmiin kuuluvat esimerkiksi

- erilaiset horisontaaliset puu-rakenteet, jotka soveltuvat usein ontologioiden mallintamiseen. Näistä puu-rakenteista on useita eri varientteja, joiden tehokkuus visualisoinnissa vaihtelee. (Athanassiades ym. 2009: 155.)
- radiaalinen puu, joka visualisoi noodit kehittäin jostakin alkunoodista lähtien. Kehien säde kasvaa alkunoodista ulospäin, kunnes kaikki noodit on visualisoitu. (Athanassiades ym. 2009: 155.)
- orgaaninen asetelma, jossa noodien sijainta arvotaan satunnaisesti. Usein tätä asetelmaa voi hyödyntää visualisoidessa tietoa, joka on luonnostaan orgaanista tai esimerkiksi käyttäjien syöttämää. (Athanassiades ym. 2009: 155.)
- hyberbolinen puu-rakenne, jolla yritetään keskittää visualisointi tiettyihin tärkeisiin noodeihin (Graves 2015: 152).

Oikean graafiasetelman valitsemisella voidaan huomattavasti vaikuttaa visualisoidun tiedon selkeyteen. Yleistä asetelmaa, jolla voitaisiin visualisoida kaikkiin käyttötarkoituksiin tarkoitettu tieto yhtä tehokkaasti, ei kuitenkaan käytännössä ole.

Visualisointistrategian ja graafiasetelman valitsemisen lisäksi RDF-tietomallia visualisoidessa täytyy valita ja hyödyntää muitakin sotkeutumista estäviä tekniikoita. Niitä valitessa on hyvä muistaa jo aikaisemmin esitelty Visual Information Seeking Mantra, jota seuraten voidaan luoda käyttöliittymiä, jotka tuntuvat luonnollisilta ja joiden käyttäminen on sujuvaa. Käytännössä sen seuraaminen tarkoittaa sellaisten tekniikoiden käyttämistä, jotka vähentävät esitettyä tietoa käyttäjän suorittamien toimenpiteiden seurauksena tai antavat esitetylle tiedolle jotakin lisämerkitystä ilman, että alkuperäinen merkitys muuttuu tai katoaa.

Joissakin tapauksissa visualisoitava tietomalli saattaa sisältää literaaliarvoja, joiden esittämät merkkijonot ovat todella pitkiä. Pitkien merkkijonojen esittäminen noodeina sotkee esitetyn graafin, sillä pitkiä merkkijonoja sisältävät literaalit vievät huomattavasti enemmän tilaa piirtoalueelta kuin esimerkiksi tavalliset URI:t. (Graves 2015: 153.) Kuvassa 7 näkyy graafi, jossa esitetään pitkiä merkkijonoja sisältäviä literaaliarvoja. Kuvasta näkee, kuinka pitkät merkkijonot hankaloittavat graafin lukemista, varsinkin kun literaalit venyvät toistensä päälle.



Kuva 7. Sotkeutunut RDF-tietomallia esittävä graafi, joka sisältää literaaliarvoja, jotka esittävät pitkiä merkkijonoja (Graves 2015: 154).

Pitkät literaaliarvot vievät turhan paljon vapaana olevaa visualisointitilaa, vaikka ne eivät semanttisesta näkökulmasta välttämättä lisää paljonkaan arvoa visualisoinnille. Tälle ongelmalle on kaksi yleistä ratkaisua. Ensimmäinen tapa ratkaista pitkien literaalien ongelma on yksinkertaisesti vain lyhentää literaaliarvot lyhyemmäksi ja antaa käyttäjälle tapa nähdä literaaliarvon lyhentämätön merkkijono, kun literaalia manipuloidaan. Toinen tapa on vieläkin yksinkertaisempi: literaaliarvot voidaan oletusarvoisesti piilottaa visualisoinnista, ellei käyttäjä halua nimenomaan nähdä niitä (Graves 2015: 154).

Toinen literaalien aiheuttama ongelma on se, että RDF-tietomallissa literaaliarvot eivät ole uniikkeja. Tämä tarkoittaa, että samanlaisia literaaleja, kuten esimerkiksi totuusarvo *”true”^^xsd:boolean,* saattaa esiintyä yhdessä graafissa useaan otteeseen. Tämänkin ongelman voi ratkaista aikaisemmin mainitulla oletusarvoisella literaalien piilottamisella, tai vaihtoehtoisesti literaaleja voidaan käsitellä URI:en tavoin, eli jokaiselle identtiselle literaaliarvolle luodaan vain yksi noodi (Graves 2015: 154).

Viimeisenä mainittavana sotkeutumista estävänä tekniikkana on myös aikaisemmin mainittu lisämerkityksen luominen visualisoidulle tiedolle. Tähänkin on monia eri tekniikoita, riippuen visualisoivan sovelluksen käyttötarkoituksesta, mutta tärkeimpiä ovat sellaiset, jotka lisäävät visuaalisia vihjeitä tiedon merkityksestä. Yksinkertaisimpana esimerkkinä on erilaisten muotojen tai kuvien käyttäminen noodien visualisointiin ja värien hyödyntäminen tiedon välittämiseen.

Muotoja ja värejä voi hyödyntää visualisoinnissa monin eri tavoin. Samoilla predikaateilla esitetyt kaaret noodien välillä voidaan värikoodata niin, että käyttäjä pystyy ymmärtämään kaarien esittämän tiedon ilman, että kaaren esittämän predikaatin arvoa tarvitsee lukea. Tällä tavoin käyttäjä voi myös yleisemmältä esitystasolta hahmottaa tarkemmin noodien välisiä yhteyksiä, eikä hänen tarvitse keskittää visualisointia tiettyyn kaareen. Värejä voi myös hyödyntää noodien visualisoinnissa, esimerkiksi antamalla noodeille, joihin liittyy tietty *rdfs:type*-predikaatti, erilaisen värin muihin noodeihin verrattuna (Graves 2015: 154). Värien sijasta noodien visualisoinnissa voi myös hyödyntää muotoja ja kuvia, esimerkiksi hyödyntäen *rdfs:type*-predikaattia tietyn kuvan näyttämiseen noodin päällä. Tavallisten URI:en ja Blank nodejen välille voidaan myös tehdä visuaalinen erottelu käyttämällä erilaisia muotoja niiden esittämiseen (Graves 2015: 154).

5 Graafitietokannan visualisointityökalu rdfvis.js

Insinöörityön tarkoituksena oli tutkia RDF:n visualisointia ja kehittää tutkinnan pohjalta sovelluskomponentti, tässä tapauksessa JavaScript-kirjasto, jolla voidaan visualisoida RDF-tietomallin esittämää tietoa käyttötarkoituksesta riippumatta. Kirjasto integroidaan erillisenä komponenttina visualisointityökaluksi graafitietokantaan, ja sen pääasiallisena tarkoituksena on mahdollistaa tietokannan sisältämän tiedon visuaalinen esittäminen erilaisissa käyttötilanteissa.

5.1 Visualisointityökalun tausta ja tavoitteet

Kuten luvussa 4 mainittiin, RDF-tietomallin esittämää tietoa käyttävät sovellukset ovat usein riippuvaisia jostakin käyttöliittymätason ratkaisusta tiedon esittämiseen käyttäjille. Usein näissä ratkaisuissa raaka tieto muutetaan formaattiin, jossa käyttäjälle on helpompi ymmärtää esimerkiksi eri käsitteiden väliset yhteydet. Sama pätee myös graafitietokantaan, jota varten visualisointityökalu kehitettiin. Ennen työkalun kehittämistä graafitietokanta ei tukenut geneerisen RDF:n visualisointia tai esittämistä käyttäjälle, vaan sitä varten piti kehittää jonkinlainen käyttöliittymäkerros. Vaikka kaikissa tilanteissa raa'an RDF:n visualisointi ei ole hyödyllistä käyttäjälle, on silti useita käyttötarkoituksia, joissa tätä ominaisuutta voitaisiin hyödyntää.

Pääasiallinen tavoite visualisointikomponentille, josta tästä eteenpäin käytetään nimeä rdfvis, on luoda konfiguroitava JavaScript-kirjasto, joka mahdollistaa RDF:n visualisoinnin. Ensimmäisenä käyttökohteena kirjastolle toimii graafitietokannan admin-käyttöliittymä ja kohdekäyttäjinä pääasiassa tietokannan ylläpitäjät. Kirjaston pitäisi olla mahdollisimman konfiguroitava, jotta sitä voidaan käyttää myös muissa käyttökohteissa ja sovelluksissa sekä myös useiden erilaisten tietomallien visualisointiin.

Visualisointityökalulle asetettiin useita eri vaatimuksia ja toivottuja piirteitä, mutta insinöörityön aikana niistä kaikkia ei ehditty toteuttamaan. Pääpiirteiltään visualisointityökalulta kuitenkin toivottiin seuraavia ominaisuuksia:

- RDF-graafi pitää pystyä visualisoimaan noodeina ja suunnattuina kaarina. Visualisoinnin täytyy tukea myös suuria graafeja.
- Työkalun pitää mahdollistaa yksinkertainen RDF-graafin selaus ja tutkinta. Graafin selaus alkaa tiettyä URI:a esittävän noodista, ja sen avaamalla pystytään selaamaan kaikki muut noodit, jotka ovat siihen suoraan sidoksissa.
- Noodeja pitää pystyä visualisoimaan käyttäen kuvia. Noodille asetettavan kuvan pitää olla konfiguroitavissa esimerkiksi *rdf:typen* perusteella.
- Visualisointityökalussa pitäisi olla yksinkertainen graafieditori, joka tukee operaatioita, kuten lisäys, poisto ja muokkaus noodien välisiin yhteyksiin ja literaaliarvoihin.
- Uusien resurssien luominen pitää olla mahdollista kopioimalla olemassa olevia visualisoituja resursseja.

Mainittujen lisäksi yksi työkalun tärkeimmistä kehitysvaatimuksista oli yleinen konfiguroitavuus, sillä työkalua piti pystyä hyödyntämään monissa eri ympäristöissä ja käyttökohteissa, ylläpitokäyttöliittymän lisäksi.

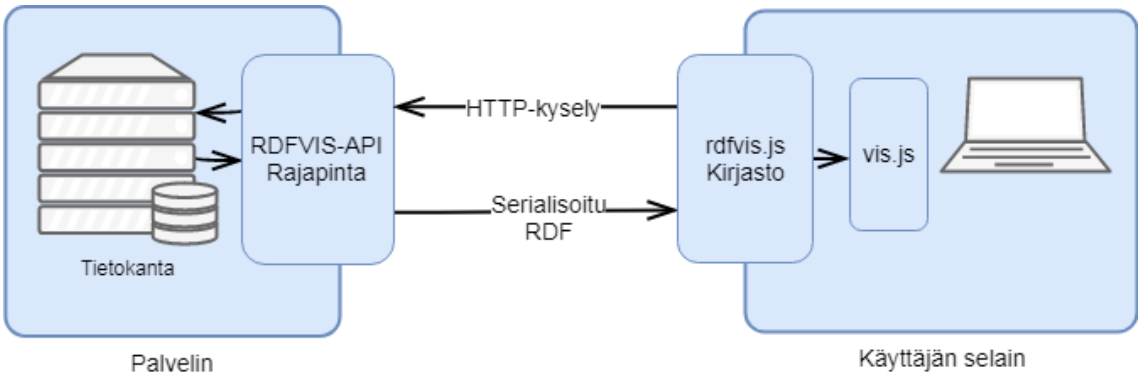
5.2 rdfvis.js-arkkitehtuuri

Jo ennen työkalun toteuttamisen aloitusta sen yleispiirteinen arkkitehtuuri ja toimintaympäristö olivat selviä. Visualisointityökalun täytyi olla erillinen komponentti, joka voidaan valinnaisesti ottaa käyttöön graafitietokannan käyttöliittymätasolla eli selaimissa. Luonnollinen valinta ohjelmointikieleksi työkalua varten olikin täten JavaScript. Valintaa edesauttoi myös se, että JavaScript-maailma tarjoaa useita jo valmiita kirjastoja moniin eri tarkoituksiin, mukaan lukien tiedon visualisointiin.

Työkalun pohjaksi vertailtiin joitakin JavaScript-kirjastoja, jotka tukevat graafien tai verkostojen esittämistä. Vertailuun ei kuitenkaan käytetty paljon aikaa, vaan kirjastoksi valittiin vis.js-kirjasto, joka monien muiden tietorakenteiden lisäksi pystyy visualisoimaan verkostoja, jotka ovat käytännössä sama asia kuin graafit. Valinta perustui muun muassa siihen, että vis.js on aktiivisesti ylläpidetty ja kehitetty ja sille on tarjottu hyvä dokumentaatio verkossa. Vis.js tarjoaa monia erilaisia moduuleita erilaisen tiedon visualisointiin, mutta työkalua varten hyödynnettiin vis.js:n Network-moduulia, joka tukee konfiguroitavia tyylejä, kuvia, värejä ja muotoja graafin esittämiseen ja kykenee esittämään tuhansia noodeja kerrallaan (vis.js. 2017).

Visualisointityökalu, eli rdfvis.js, jonka nimi muodostuu pohjana käytetystä vis.js-kirjastosta ja kirjaston käyttötarkoituksesta, joka on RDF-visualisointi, on siis käytännössä kehitetty vis.js kirjaston päälle ja käyttää sitä graafin visualisointiin sekä ensisijaisesti myös graafin ja käyttäjän välisiin vuorovaikutuksiin. Rdfvis.js taas vuorostaan hoitaa käytännössä kaikki muut visualisointia tukevat toiminnot. Näihin toimintoihin kuuluvat pääpiirteittäin lisätiedon hakeminen, RDF-serialisointiformaattiin serialisoidun tiedon jäsentely vis.js:lle sopivaan tietorakenteeseen, RDF-tiedon hyödyntäminen visualisoitavien elementtien luomisessa, tiedon suodattaminen ja käyttäjän tekemien toimintojen käsittely ja niihin reagointi.

Rdfvis.js on riippuvainen palvelintasolla implementoidusta RDFVIS-API-rajapinnasta, jonka kautta kirjasto hakee asynkronoidusti tietokannan sisältämää tietoa. Rajapinta on toteutettu käyttäen REST (Representational State Transfer) -arkkitehtuurimallia, ja näin ollen se vastaa Hypertext Transfer Protocollan (HTTP) määrittelemiin kutsuihin. Rajapinta palauttaa Notation3 (N3) -serialisointiformaatissa olevaa RDF-tietoa, jonka rdfvis.js jäsentelee JavaScript-objekteiksi, joita vis.js-kirjasto voi käyttää. Lisäksi kaikki rajapinnan palauttama tieto säilytetään selaimen muistissa, jotta sitä voidaan hyödyntää esimerkiksi noodien ja kaarien ulkonäön määrittelemiseen. Rdfvis.js ei ole suoraan riippuvainen tietystä rajapintaimplementaatiosta, vaan käytännössä kirjasto voi hakea tietoa mistä tahansa palvelimesta, jossa RDFVIS-APIn mukainen rajapinta on toteutettu. Kuvassa 8 on esitetty visualisointityökalun arkkitehtuuri yksinkertaistetusta näkökulmasta, ja kuvasta näkyy myös RDFVIS-APIn ja rdfvis.js:n välinen kommunikaatio.



Kuva 8. Työssä toteutetun rdfvis.js-visualisointityökalun yksinkertaistettu arkkitehtuuri.

Kuten kuvasta näkyy, visualisointityökalun arkkitehtuuri ei ole kovinkaan monimutkainen, sillä työkalussa on vain kaksi pääasiallista komponenttia: selaimessa käytettävä rdfvis.js-kirjasto ja palvelimella oleva RDFVIS-API-implementaatio.

5.3 Kehitysprosessi

Visualisointityökalun kehitysprosessi alkoi jo aikaisemmin mainitulla erilaisten visualisointikirjastojen vertailulla. Vertailulla pyrittiin löytämään valmis kirjasto, joka tukee mahdollisimman hyvin yleisten graafien esittämistä ja mahdollistaa graafien käsittelyn, ja erilaisten toimintojen suorittamisen niiden kanssa, kuten esimerkiksi noodien liikuttaminen tai niiden valitseminen. Vertailulla kartoitettiin myös vaadittavan työn määrää ja mahdollisia vaatimuksia ja toiveita, joita visualisointityökalulle oltiin esitetty. Niin kuin jo mainittu, visualisointikirjastoksi valittiin vis.js-kirjasto, ja valintaprosessissa ei kestänyt pitkään.

Visualisointikirjaston valitsemisen jälkeen työkalun kehitys eteni vaatimusmäärittelyyn, joka on karkeasti kuvattuna luvussa 5.1. Vaatimusmäärittelyllä pyrittiin kartoittamaan työkalulta vaaditut toiminnot ja ominaisuudet, joilla mahdollistetaan työkalun uudelleenkäytettävyys useissa eri käyttötarkoituksissa. Määrittelyllä pyrittiin myös karsimaan joitakin RDF:n visualisointiin liittyviä haasteita jo ennen kehittämisen alkua ja selvittämään ominaisuuksia, jotka mahdollistavat visualisoidun graafin selaamisen ja käyttämisen sujuvasti.

Kun visualisointikirjastolta vaaditut ominaisuudet ja piirteet olivat selvät, siirryttiin kehitysprosessissa niin sanotusti tutkintavaiheeseen, jossa esimerkiksi tutkittiin vis.js:n

käyttöä ja siihen liittyviä huomioitavia asioita myöhemmin kehitysprosessin aikana ja myös muita kirjastoja, joita tultaisiin tarvitsemaan visualisointikirjastolta vaadittavien piirteiden toteuttamiseen. Tutkintavaiheen aikana selvisi esimerkiksi tapoja, joilla graafin eri osia pystyttäisiin visualisoimaan eri tavoin riippuen annetusta tiedosta, ja myös joitakin rajoitteita, joita vis.js sisältää ja niiden mahdolliset vaikutukset myöhemmin kehitysprosessin aikana. Tutkintavaiheen tärkeimpinä tuloksina oli esimerkiksi JavaScript-kirjasto, joka hoitaa serialisoidun RDF:n jäsentelyn JavaScript-objekteiksi, ja vis.js-kirjaston suurimmat rajoitteet visualisointityökalun vaatimaa käyttötarkoitusta varten.

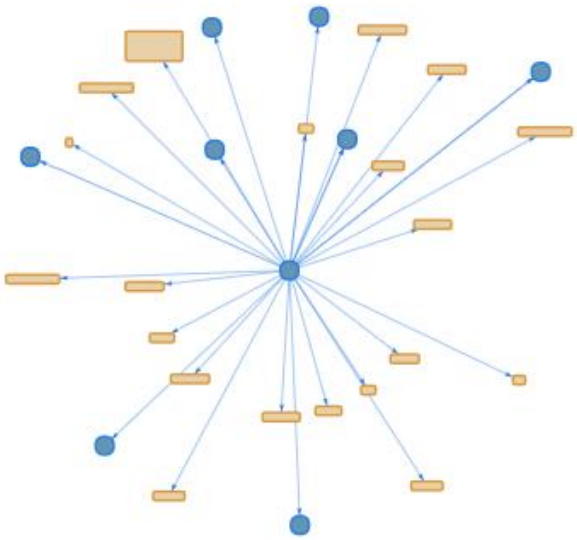
Kun tutkintavaiheessa oli tutkittu muuhun kehitysprosessiin vaikuttavat tekijät ja vaatimusmäärittelyssä selvitetty visualisointityökalulta vaaditut ominaisuudet, alkoi varsinainen työkalun kehitys niiden pohjalta. Kehitysprosessin aika yritettiin toteuttaa vaatimusmäärittelyssä vaaditut ominaisuudet samassa järjestyksessä, kuin ne on luvussa 5.1 kuvailtu, sillä vaatimukset on laadittu tärkeysjärjestyksessä visualisoinnin ja työkalun käytön kannalta. Koska vis.js-kirjasto ei suoraan tue RDF-tiedon visualisointia, oli kehitysprosessin kannalta tärkeää, että serialisoidun RDF-tiedon ja vis.js:n käyttämien JavaScript-objektien välinen jäsentely saataisiin toimimaan mahdollisimman aikaisessa vaiheessa kehitystyötä. Serialisoidun RDF:n jäsentely olikin ensimmäinen kehityspiirre joka työkaluun toteutettiin. RDF:n tulkitsemista ja jäsentelyä JavaScript-objekteiksi varten otettiin käyttöön tähän tarkoitukseen kehitetty JavaScript-kirjasto, jonka ympärille kehitettiin luokka, joka luo vis.js:n käyttämiä JavaScript-objekteja.

Kun serialisoitua RDF:ää pystyttiin työkalun avulla muuttamaan JavaScript-objekteiksi, oli seuraava askel kehitysprosessissa erilaisten visualisointistrategioiden ja graafiasetelmien vertailu ja valitseminen. Visualisointistrategiaksi valittiin suoraan navigaatiostrategia, sillä se täytti parhaiten vaatimuksissa toivotun graafin selauksen ja tutkimisen. Tässä vaiheessa kehitysprosessia tunnistettiin myös lisävaatimuksia työkalulle. Työkalun täytyi pystyä visualisoimaan sille myös käsin syötettyä tietoa, jota tietokanta ei välttämättä sisällä. Tämä siis tarkoitti, että navigaatiostrategian lisäksi työkalun täytyi tukea jonkinlaista display-at-once-strategiaa, jotta työkalulle syötetty serialisoitu RDF-tieto, joka ei tule suoraan RDFVIS-API-rajapinnasta, pystytään visualisoimaan kerralla. Navigaatiostrategiaa ei tässä tilanteessa voitu hyödyntää, koska oletuksena oli, että työkalulle syötetty tieto haluttiin nähdä visualiisesti kerralla eikä inkrementaalisesti noodeja navigoiden.

Navigaatiostrategioiden valinnan jälkeen piti seuraavaksi selvittää niille tehokkaimmat ja selvimmät graafiasetelmat. Tämä vaihe kesti melkein koko työkalun kehityksen ajan, sillä useita erilaisia asetelmia testattiin inkrementaalisesti. Ensimmäiseksi testattiin vis.js:n suorittamaa asettelua, mutta se todettiin nopeasti huonoksi menetelmäksi, sillä asetteluun ei voinut suoraan vaikuttaa, jolloin asettelun muodostamista graafeista tuli usein epäselviä tai liian sotkuisia RDF:n visualisointia varten.

Seuraavaksi testattiin jo luvussa 4.3 mainittua orgaanista asettelua, jolloin noodeille asetettiin pseudosatunnaiset sijainnit, mikä tarkoittaa, että noodien sijainnit eivät olleet täysin deterministisiä, mutta eivät myöskään täysin satunnaisia. Orgaaninen asettelukaan ei toiminut käytännössä hyvin, sillä pseudosatunnainen asettelu ei ollut kovin tehokasta ja noodien visualisoinnissa saattoi kestää usein jopa sekunteja, jolloin työkalun käyttökokemus saattoi kärsiä.

Visualisointiin valittiin lopuksi radiaalinen puu, jossa noodit asetellaan käyttäjän valitseman ”alkunoodin” ympärille kehiin. Tämän asetelman kehitys ja parantaminen jatkui koko kehitysprosessin ajan, pääasiassa visualisoinnin selkeyttämisen takia. Samaa asetelmaa käytettiin myös display-at-once-visualisointistrategiassa, vaikka käyttäjä ei tässä strategiassa valitsekaan alkunoodia, vaan kirjasto päättelee annetusta tiedosta todennäköisesti tärkeimmän alkunoodin, jonka ympärille muut noodit asetellaan. Kuvassa 9 näkyy keskeneräisen työkalun visualisoima graafi, jossa tieto on aseteltu käyttäen radiaalista puuta. Koska työkalu oli vielä kehitysvaiheessa, kuvassa ei näy tarkalleen, millaista tietoa työkalu esittää.



Kuva 9. Kehitysvaiheessa olevan visualisointityökalun esittämä RDF-graafi käyttäen radiaalista puuta, jossa noodit sijoitetaan yhden ”alkunoodin” ympärille kehiin.

Kuten kuvasta 9 näkyy, radiaalinen puu selkeyttää graafin visualisointi jakamalla noodit tasaisemmin kehille, jotka on erotettu toisistaan. Koska algoritmi radiaalisen puun luomiselle kehitettiin suoraan rdfvis.js-kirjastoon eikä toteutuksessa oltu riippuvaisia esimerkiksi vis.js-kirjaston sijoittelualgoritmistä, oli myös mahdollista vaikuttaa siihen, mitkä noodit sijoitetaan millekin kehille. Myöhemmin kehitysprosessissa algoritmi muutettiin sijoittamaan kaikki literaaliarvot sisäisimmille kehille, koska ne liittyvät vain ”alkunoodiin” eikä niistä lähde kaaria muihin noodeihin, ja resursseja kuvaavat noodit kaaren ulkopuolelle. Kuvassa 9 literaaliarvot on esitetty oransseina suorakulmioina ja resursseja kuvaavat sinisinä ympyröinä.

Visualisointistrategian ja graafiasetelman valitsemisen ja implementoinnin jälkeen visualisointityökalun ydintoiminnot olivat valmiina: työkalu pystyi vastaanottamaan serialisoitua RDF-tietoa ja esittämään sen graafisesti, vaikka graafin sotkeutumista ei ollut vielä tarkemmin estetty erilaisilla tekniikoilla. Tässä vaiheessa kehitysprosessia voitiin alkaa tarkemmin määritellä ja implementoida RDFVIS-API-rajapintaa. Rajapinnalle annettiin aluksi vain muutama yksinkertainen vaatimus: sen piti mahdollistaa subject-graafin hakeminen tietylle URI:lle ja valinnaisesti myös saman URI:n object-graafi samalla kyselyllä. Kehitysprosessin aikana rajapinnasta luotiin oletus-implementaatio erilliseksi moduuliksi graafitietokantaan Java-ohjelmointikielellä, jolloin graafitietokantaa hyödyntävät sovellukset voisivat halutessaan tukea rdfvis.js-kirjaston käyttöä, ja näin

ollen myös graafien visualisointia, ottamalla RDFVIS-API-moduuli käyttöön. Tällä toteutustavalla sovellukset voisivat myös ohittaa rajapinnan oletusimplementaation ja toteuttaa siitä oman version sovelluksessa, jos rajapinnan oletusimplementaatio esimerkiksi paljastaisi liikaa tietoa rajapinnan kautta.

Rajapinnan toteutus ja määrittely laajeni myös kehitysprosessin edetessä. Myöhemmin kehitysprosessin aikana rajapintaan toteutettiin esimerkiksi suodatusjärjestelmä, jolla jokainen rajapinnalle tullut kutsu pystyi määrittelemään tarkemmin, millaista tietoa rajapinnasta haettiin. Näin mahdollistettiin visualisointityökalun käyttökohteesta riippuen visualisoidun tiedon tarkennus, jolloin tietokannassa olevasta tiedosta voitiin rajata näkyviin vain tietty tieto. Myöhemmin suodatusjärjestelmää laajennettiin myös mahdollistaen, että suodattimet pystyivät hakemaan kyselyiden aikana myös visualisoidulle tiedolle metadataa, jolla pystyttiin vaikuttamaan visualisoidun graafin ulkonäköön esimerkiksi vaihtamalla noodien kuvia metadatan perusteella.

Seuraavana vaiheena rajapinnan implementoinnin jälkeen oli luonnollisesti rdfvis.js-kirjaston ja rajapinnan välisten yhteyksien luonti, eli rajapinnan ja kirjaston yhdistäminen yhdeksi kokonaisuudeksi. Tätä varten rdfvis.js-kirjastoon mahdollistettiin rajapinnan parametrien konfigurointi ja kirjastoon luotiin järjestelmä, joka mahdollistaa kyselyiden tuottamisen rajapinnalle ja rajapinnan palauttamien vastauksien käsittelyn, eli käytännössä palautetun serialisoidun RDF:n jäsentelyn JavaScript-objekteiksi, kuten jo aikaisemmin selitetty. Samaan aikaan, kun visualisointikirjasto yhdistettiin rajapintaan, toteutettiin myös siihen erilaisia käytettävyyttä parantavia ominaisuuksia. Yksi näistä ominaisuuksista oli mahdollisuus avata visualisoitujen noodien subject-object-graafi rajapinnan kautta kaksoisklikkaamalla visualisoitua noodia, mikä onkin pääasiallinen tapa hakea uutta tietoa visualisointityökaluun.

Kun rdfvis.js-kirjasto ja rajapinta oli konkreettisesti yhdistetty eli rdfvis.js pystyi dynaamisesti hakemaan rajapinnasta lisää tietoa käyttäjän tekemiin toimenpiteisiin reagoiden, oli aika integroida visualisointityökalu kokonaisuutenaan tietokannan ylläpitonäkymään. Koska visualisointityökalu integroitiin tietokannan ylläpitonäkymään, täytyi sen olla lähestulkoon konfiguroimaton, jotta mahdollisimman paljon tietoa saataisiin näkyviin kerralla eikä ylläpitonäkymä antaisi väärää tietoa tietokannan tilasta. Tässä vaiheessa kehitysprosessia ilmeni myös monia haasteita ja puutteita visualisoinnin selkeydessä. Esimerkiksi kun visualisoitiin tietokantaa, joka sisälsi paljon ontologiamallinnusta joko OWL:n tai RDFS:n avulla, menetti visualisointityökalu luettavuuden nope-

asti, koska työkalun aiemmat testausversiot riippuivat tiedon aktiivisesta suodattamisesta rajapinnan kautta ja työkalussa ei ollut vielä implementoitu monia muita visualisointia selkeyttäviä ominaisuuksia.

Visualisointityökalu integroitiin ensiksi tietokannan ylläpitonäkymään luomalla painike, joka avaa valitun URI:n visualisointi-ikkunaan samalla hakien sille rajapinnan kautta suodatinjärjestelmän antamat tiedot. Tällä tavoin tietokannan ylläpitäjä pystyi valitsemaan häntä kiinnostavan URI:n ja selaamaan siihen liittyvää tietoa. Myöhemmin visualisointityökalu lisättiin myös tietokannan ylläpitonäkymän insertointisivulle, josta tietokantaan voidaan lisätä RDF-tietoa web-liittymän kautta. Tälle sivulle lisättyyn visualisointityökaluun jouduttiin hyödyntämään aikaisemmin luotua disply-at-once-visualisointistrategiaa. Visualisointityökalulla pystyi nyt siis selaamaan tiettyyn URI:n liittyviä yhteyksiä ja visualisoimaan uutta tietoa ennen sen insertointia tietokantaan. Tässä vaiheessa visualisointityökalun olisi voinut integroida myös muihin ylläpitonäkymän sivuihin, kuten esimerkiksi SPARQL-tulossivu, mutta kehitysprosessissa päätettiin kerätä muilta tietokannan ylläpitäjiltä palautetta visualisointityökalun käytettävyydestä ja keskittyä käytettävyyden ja visualisoinnin selkeyden parantamiseen.

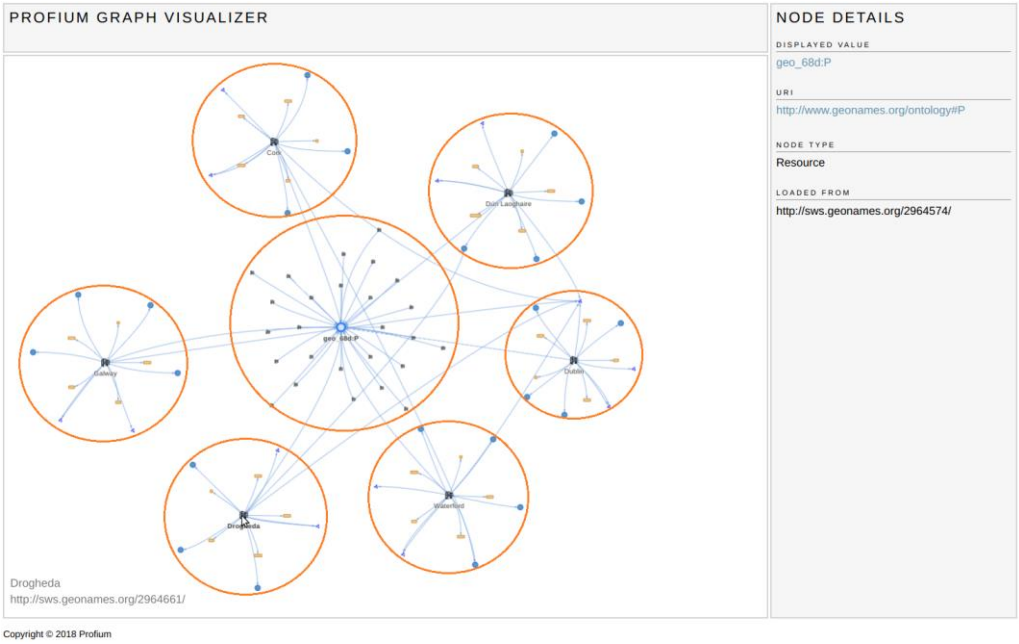
Kehitysprosessi jatkuikin siis visualisoinnin selkeyttämisellä ja käytettävyyden parantamisella lähes koko loppuprosessin ajan. Osa visualisointityökalun vaatimuksista määritteli ominaisuuksia, jotka mahdollistaisivat graafin visualisoinnin lisäksi myös sen muokkauksen. Näitä ominaisuuksia ei kuitenkaan ehditty kehitysprosessin aikana implementoida. Nämä graafin muokkausominaisuudet, ja muut visualisointityökalun integroimiseen graafitietokantaan liittyvät piirteet, jäävät jatkokehityspiirteiksi visualisointityökalun myöhempää kehittämistä varten. Vaikka kehitysprosessin loppuosassa keskityttiin visualisoinnin selkeyttämiseen, jäi tällekin kehitysalueelle monia piirteitä jotka voisivat parantaa sitä, kuten esimerkiksi noodien ryhmittely ja yhdistäminen jonkin kriteerin perusteella sekä muut luvussa 4 läpikäydyt strategiat.

5.4 Visualisoinnin haasteet ja niiden ratkaisut

Kun visualisointityökalua alettiin integroida ensimmäiseen oikeaan käyttökohteeseen eli graafitietokannan ylläpitokäyttöliittymään, huomattiin työkalussa useita haasteita visualisointiin liittyen. Tässä luvussa käydään tarkemmin läpi kehitysprosessin aikana havaittuja visualisointihaasteita ja tapoja, joilla nämä haasteet ratkaistiin.

Yksi suurimpia haasteita visualisoinnissa oli uusien avattujen noodien asettelu ja sijoitus vapaana olevalle piirto-alueelle. Visualisoinnissa käytetty vis.js-kirjasto pohjautuu HTML5-canvakseen ja on toteutettu JavaScriptillä, joten suorituskyvyltään se on jokseenkin rajoittunut. Tämä suorituskykyrajoitus tarkoitti, että uusia noodeja sijoitellessa piti pyrkiä mahdollisimman tehokkaaseen ratkaisuun, jotta visualisointityökalun käyttö tuntuisi myös responsiiviselta. Kuten jo luvussa 5.3 mainittiin, yksi testattu sijoitusstrategia oli orgaaninen sijoittelu, jossa graafissa visualisoidut noodit sijoiteltiin pseudosatunnaisesti piirtoalueelle. Tässä sijoittelustrategiassa törmättiin kuitenkin ensimmäistä kertaa vis.js:n suorituskykyrajoitteisiin, ja näitä rajoitteita tuli jotenkin kiertää.

Vis.js:n suorituskykyrajoitteiden välttämistä varten rdfvis.js-kirjastoon sovellettiin järjestelmä, jonka avulla noodit yhdistettiin alkunoodiin hyödyntäen sektoreita. Käytännössä järjestelmä laski uusien noodien määrän perusteella ympyrän muotoisen sektorin, johon noodit mahtuvat, ja allokoi tälle tilan sopivan tilan piirtoalueelta. Järjestelmän avulla uusia noodeja visualisoidessa ei enää tarvinnut tarkistaa noodikohtaisesti visualisointikonflikteja, eli tilanteita, joissa noodit sijoittuvat päällekkäin, vaan visualisointityökalu pystyi tarkistamaan konfliktit sektorien perusteella ja näin löytämään sopivan sijainnin uusille visualisoiduille noodeille. Kuvassa 10 on esitelty sektorimenetelmällä sijoitetut noodit visualisointityökalussa. Oranssilla merkityt sektoreita esittävät ympyrät on jälkikäteen lisätty, eikä niitä visualisoinnin yhteydessä näytetä käyttäjälle.



Kuva 10. Esimerkki insinöörityössä toteutetun visualisointityökalun sektorimenetelmällä sijoitetuista noodeista.

Kuten kuvasta näkyy, sektorimenetelmällä voi helposti sijoittaa uusia avattuja noodiryhmiä hyödyntäen jo käytössä olevaa graafiasetelmaa (tässä tapauksessa radiaalista puuta). Kun noodia kaksoisklikataan, se siirretään sopivan matkan päähän riippuen rajapinnan palauttamasta RDF-graafista, ja sille allokoidaan sopivankokoinen sektori visualisointialueelta, johon muut avatut noodit eivät voi siirtyä. Menetelmä on myös tehokas, sillä esimerkiksi kuvassa näkyvästä graafista vain avoinna olevat sektorit joudutaan tarkistamaan konfliktien varalta, verrattuna kaikkien noodien yksittäiseen tarkistamiseen.

Seuraava ongelma visualisoinnissa oli yksinkertaisesti liian suuri määrä visualisoituja triplejä. Tämä ongelma ratkaistiin jo luvussa 5.3 kuvaillulla rajapinnan suorittamalla suodatuksella, jonka avulla rdfvis.js-kirjasto pystyy antamaan rajapinnalle parametreinä erilaisten suodattimien nimet, joita rajapintatoteutus sitten käyttää esitettävän tiedon suodattamiseen, jo ennen kuin tieto palautetaan rdfvis.js:lle. Suodatinmekanismi toimii siten, että rajapinnalle konfiguroidaan erilaisia nimettyjä SPARQL-kyselyitä, jotka suoritetaan rajapinnan saamaa subject-object-graafia vastaan. SPARQL-kyselyiden tulokset poistetaan graafista ja rdfvis.js:lle palautettu serialisoitu RDF vastaa graafia, josta kaikki suodattimien löytämät triplet on poistettu. Esimerkkikoodissa 6 kuvaillaan konfiguroitava suodatin, joka suodattaa tuloksista kaikki triplet, jotka sisältävät *rdf:type*-URI:n lausunnon predikaattisijainnissa.

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
construct * where {
     ?uri rdf:type ?subject .
}
```

Esimerkkikoodi 6. Yksinkertainen SPARQL-kysely, joka voidaan konfiguroida rajapintatoteutukseen triplejen suodattamiseen, jos niissä esiintyy rdf:type-predikaatti.

Vaikka esimerkki onkin äärimmäisen yksinkertainen, suodatinmekanismille voidaan konfiguroida mikä tahansa SPARQL-kysely tulosten suodattamista varten, riippumatta sen monimutkaisuudesta. Suodatinmekanismilla on kuitenkin omat rajoitteensa. Mekanismia ei esimerkiksi voitu hyödyntää tehokkaasti graafitietokannan ylläpitokäyttöliittymässä, sillä tiedon poistaminen antaisi väärän kuvan tietokannassa säilytetystä tiedosta. Useimmissa käyttötapauksissa suodatinmekanismilla voidaan kuitenkin huomattavasti selkeyttää visualisoitua graafia ja parantaa työkalun käytettävyyttä.

Kun esitetyn tiedon määrää ja sen yleistä asetelmaa pystyttiin rajoittamaan ja hallitsemaan aikaisemmin mainituilla menetelmillä, keskityttiin visualisoinnin ongelmien kor-

jaamisessa semanttisen tiedon laajentamiseen. Käytännössä tämä tarkoitti siis sitä, että visualisoiduille noodeille ja kaarille annettiin lisää merkitystä käyttäjän näkökulmasta. Luvussa 4 esiteltiin jo joitakin tapoja lisätä visualisoidun tiedon merkitystä, kuten esimerkiksi värien ja erilaisten muotojen käyttö. Kuvissa 10 ja 9 näkyy jo eräänlainen implementaatio tiedon merkityksen lisäämisestä, sillä literaaliarvot ja resurssit on visualisoitu eri väreillä ja muodoilla. Tämä ei kuitenkaan aina riitä visualisoinnin riittävään selkeyttämiseen, joten visualisointityökalussa hyödynnettiin muitakin samankaltaisia menetelmiä esitetyn graafin selkeyttämiseen.

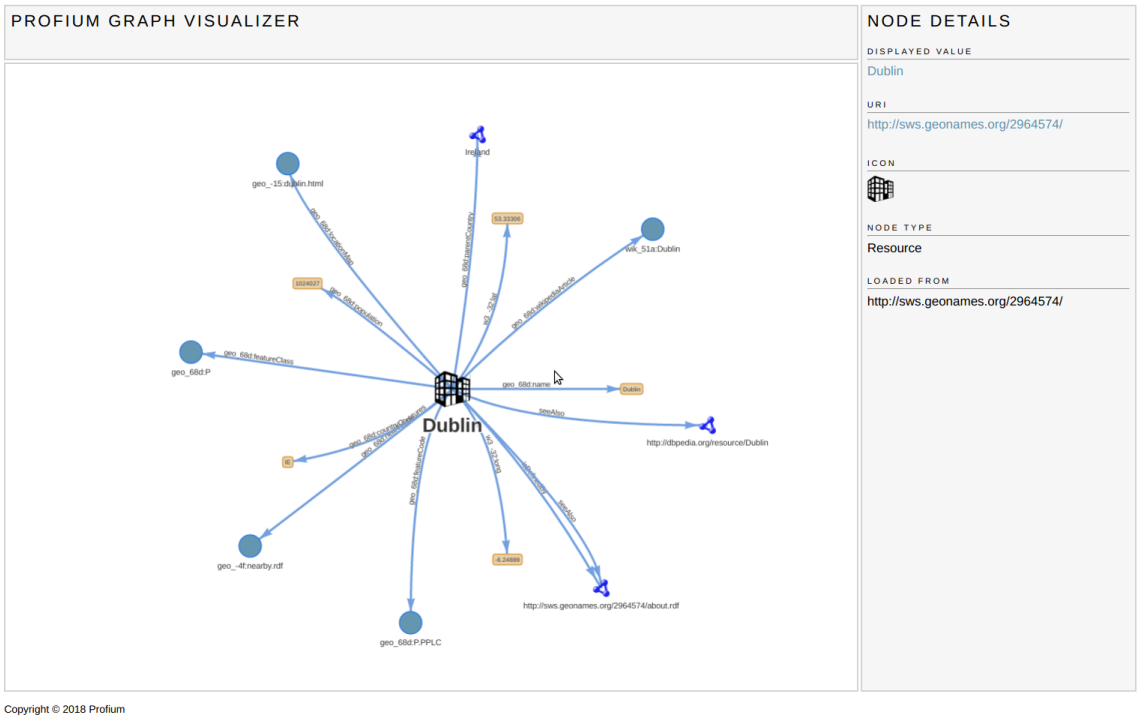
Ensimmäinen graafia visuaalisesti värein tai muodoin selkeyttävistä menetelmistä on kuvien käyttö noodien esittämiseen. Rdfvis.js voidaan konfiguroida etsimään tiettyjä triple-kaavoja visualisoidusta tiedosta jokaista resurssinoodia varten ja päättelemään tälle parhaiten sopiva kuva, josta käyttäjä pystyy päättelemään resurssin oikean tarkoituksen ja roolin tietokannassa. Toinen tapa lisämerkityksen luomiseksi on värien ja tyylien käyttäminen kaarien visualisointiin. Rdfvis.js:lle voidaan konfiguroida erilaiset tyylit kaarille, jotka vastaavat tiettyjä predikaatteja. Tällä metodilla pystytään käyttötarkoituksesta riippuen merkitsemään visualisoinnin kannalta merkitsevimmät noodien väliset yhteydet esimerkiksi antamalla niille tyyli, jossa kaaret erottuvat paremmin oletusarvoisesti esitetyistä kaarista.

Lisämerkitystä rdfvis.js-kirjastossa luodaan myös antamalla noodeille ja kaarille nimike, joka voidaan konfiguroida käyttötarkoituksesta riippuen. Koska visualisointialueen koko on rajattu, on tärkeä maksimoida visualisoitavien elementtien määrä selkeästi. Tätä vähentää huomattavasti pitkien URI:en tai literaaliarvojen esittäminen. Konfiguraatiolla voidaan määritellä samalla tavalla triple-kaava, jota käyttämälle noodille tai kaarelle etsitään nimike, kuin aikaisemmin mainitussa noodien kuvien konfiguroinnissa. Tällä metodilla noodille voidaan antaa sitä parhaiten kuvaava nimike, jonka myös käyttäjä pystyy ymmärtämään verrattuna esimerkiksi noodia esittävään URI:n esittämiseen. Tämän nimike-mekanismin lisäksi rdfvis.js tukee myös noodeille automaattisen etuliitteen generointia ja pitkien literaaliarvojen lyhentämistä konfiguroituun maksimipituuteen. Kaikki nämä menetelmät tähtäävät yhteen tarkoitukseen: pitkien merkkijonojen lyhentämiseen ja näin ollen visualisointiin varatun alueen parempaan hyödyntämiseen.

Yksi tärkeimpiä visualisointia selkeyttävistä metodeista oli jo aikaisemmin mainittu rajapinnassa suoritettu tiedon suodatus. Tämän mekanismin lisäksi rdfvis.js-kirjastoon kehitettiin sekundäärinen suodatinmekanismi, jonka avulla on mahdollista suodattaa jo

visualisoitua tietoa, jonka rajapinta on palauttanut. Tämä suodatinmekanismi oli kuitenkin pääasiassa toteutettu käyttäjien kontrolloimaksi, eli tämän mekanismin avulla ei oletusarvoisesti suodatettu mitään tietoa visualisoinnista poisteta, mutta käyttäjä pystyi halutessaan poistamaan joitakin visualisoituja elementtejä. Mekanismin toteutus kuitenkin jäi suppeaksi, sillä se mahdollisti vain literaaliarvojen ja joidenkin predikaattien avulla tiedon piilottamisen. Jatkokehityksen kannalta tätä mekanismia voidaan parantaa esimerkiksi mahdollistamaan tietotyypin tai ontologiatietojen perusteella noodien piilottamisen.

Viimeinen visualisointia selkeyttävä metodi, jota rdfvis.js-kirjastossa hyödynnettiin, oli tiedon hajauttaminen ja redundantti esittäminen. Tätä metodia käyttäen kirjaston yleinen käytettävyys parani, sillä kaikkea tietoa ei tarvinnut etsiä suoraan graafista, vaan esimerkiksi noodeihin ja kaariin liittyvät tiedot esitettiin erillisessä ikkunassa piirtoalueen oikealla puolella. Kuvassa 11 näkyy visualisointityökalun esittämä yksinkertainen graafi, jossa jokin noodi on valittuna käyttäjän toimesta ja noodiin liittyvät tiedot näytetään sen oikealla puolella niin sanotussa info-ikkunassa.



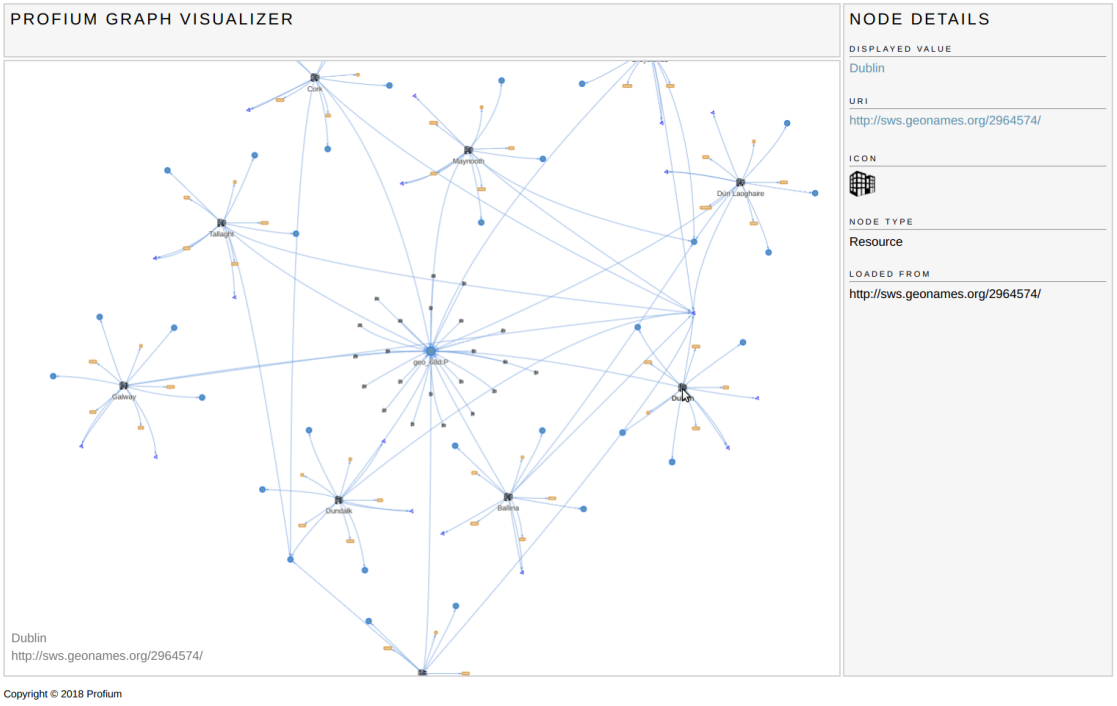
Kuva 11. Työkalun esittämä yksinkertainen graafi, jossa Dublin-noodi on valittuna. Joitakin noodin tietoja esitetään info-ikkunassa piirtoalueen oikealla puolella.

Kuten kuvasta näkyy, noodiin liittyvä tietoa on esitetty visualisointi-ikkunassa kahteen otteeseen. Noodille annettu nimike näkyy itse piirtoalueella noodin alla, samoin kuin siihen liitetty kuva, mutta myös info-ikkunassa. Lisäksi info-ikkuna esittää sen noodista tietoa, jota ei välttämättä muuten saa visualisoidusta graafista esille, kuten esimerkiksi minkä noodin kautta valittu noodi on tuotu visualisointityökaluun. Vaikka tiedon hajauttaminen ja sen redundantti esitys ei itsessään edesauta visualisoidun graafin selkeyttä suoraan, se parantaa silti sen käytettävyyttä, kun käyttäjä ei ole täysin riippuvainen graafin tutkinnasta jokaisessa käyttötilanteessa.

Visualisointityökalun kehityksen haasteellisin ongelma oli nimenomaan erilaisten visualisointia selkeyttävien menetelmien ja strategioiden löytäminen. Vaikka työkaluun kehitettiin monia toiminnallisuuksia sotkeutumisen estämiseksi, ei lopputulos siltikään ollut täydellinen, ja tarpeeksi suurta graafia esittäessä visualisointityökalu sotkeutuu väistämättä. Näin ollen työkalun jatkokehityksen kannalta visualisoinnin sotkeutumisen estäminen on vieläkin yksi prioriteeteista, ja työkalua aiotaan vielä tältä alueelta parantaa.

5.5 Tulokset

Loppujen lopuksi visualisointityökalulle asetuista tavoitteista pääasiassa saavutettiin itse visualisoinnille tärkeät tavoitteet ja vaatimukset. Graafin muokkaamista ja hallintaa varten asetetut tavoitteet jäivät käytännössä toteuttamatta kehitysprosessin aikana, ja ne siirtyivätkin työkalun jatkokehityksen piiriin. RDF-graafien visualisoinnin kannalta kuitenkin onnistuttiin toteuttamaan vaatimuksien mukainen työkalu, jota on mahdollista hyödyntää monissa erilaisissa käyttökohteissa, vaikka työkalu ei kuitenkaan ole täydellinen ja kärsii vielä joistakin RDF-visualisoinnin ongelmista. Kuvassa 12 esitetään työkalun yleiskuva, kun sitä on käytetty RDF-graafin visualisointiin ja tutkimiseen.



Kuva 12. Visualisointityökalun yleiskuva, kun sen avulla on jo esitetty graafi. Graafia on myös tutkittu jo jonkin verran, sillä alkunoodin lisäksi useita muita noodeja on avattu työkalun kautta.

Kuvan esittämässä graafissa ei ole kuitenkaan hyödynnetty monia rdfvis.js:n tukemia visualisointia selkeyttäviä ominaisuuksia, minkä takia graafin selkeys on jo hieman kärsinyt. Hyödyntäen tehokkaammin rdfvis.js:ää varten kehitettyjä konfigurointiominaisuuksia, on mahdollista vähentää graafin selkeyden heikkenemistä.

Kehitysprosessin aikana työkalu integroitiin graafitietokannan ylläpitokäyttöliittymään, ja tämä integraatio toimikin työkalun pilottiversiona. Työkalun pilotointi geneerisessä ympäristössä, kuten ylläpitokäyttöliittymässä, ei välttämättä ollut kaikista tehokkain tapa testata työkalua, sillä graafitietokannan ylläpitokäyttöliittymän täytyy pystyä esittämään käytännössä mitä tahansa tietoa, jolloin tähän käyttötarkoitukseen ei voitu hyödyntää monia konfiguraation mahdollistamia visualisointia selkeyttäviä mekanismeja, koska esitetty tieto ei olisi enää vastannut alkuperäistä tietokannan sisältämää tietoa tarpeeksi tehokkaasti. Tämä käyttökohde kuitenkin toimi hyvin jatkokehityksen kartoittamiseen, sillä se korosti automaattisien graafin selkeytysmekanismien puutetta ja tarvetta niiden kehittämiseen, jotta graafityökalu ei olisi niin riippuvainen sen riittävästä konfiguroinnista käyttötarpeita varten, vaan pystyisi passiivisesti ilman käyttäjän toimia tai konfigurointia selkeyttämään graafin esitystä.

Työkalun kehitysprosessi ei ole kuitenkaan kirjoittamisen ajanhetkellä vielä loppunut, vaan työkalun kehittämistä jatketaan kunnes kaikki vaaditut kriteerit on saavutettu. Työkalussa on paljon kehitettävää, mutta jo tässä vaiheessa kehitysprosessia siitä on saatu kehitettyä tehokas apuväline graafitietokannan sisältämän tiedon selaamiseen ja ymmärtämiseen.

6 Yhteenveto

Insinöörityössä tutkittiin RDF-tietomallin esittämien graafien visualisointia, sen haasteita ja ratkaisuja sekä kehitettiin tutkimuksen pohjalta sovelluskomponentti, joka mahdollistaa myös käytännössä graafien visualisoinnin. Sovelluskomponentti kehitettiin konfiguroitavaksi, jotta se mahdollistaisi monien erilaisten tutkintavaiheessa löydettyjen visualisointia selkeyttävien menetelmien hyödyntämisen selkeää graafivisualisointia varten useissa eri käyttötilanteissa.

RDF-tietomalli on W3C:n kehittämä ja ylläpitämä semanttisen verkon teknologiastandardi, jota käytetään semanttisen tiedon esittämiseen, mallintamiseen ja jakamiseen muodossa, joka on ohjelmistojen ja ihmisen tulkittavissa. RDF-tietomallilla esitetyn tiedon voidaan ajatella muodostavan suunnattu graafi, joka mallintaa asioiden välisiä yhteyksiä ja suhteita. Graafi ja sen sisältämät suhteet muodostuvat RDF-lausunnoista, jotka vuorostaan voidaan jakaa kolmeen osaan: subjekti, predikaatti ja objekti. Tietomallin esittämät asiat, esineet ja konseptit tunnistetaan usein uniikilla URI:lla.

RDF-tietomalli on teknologiana jo jokseenkin vanha, mutta se on saanut viimeisen vuosikymmenen aikana huomattavia määriä lisää käyttöä. RDF:ää käytetään moniin eri käyttötarkoituksiin, mutta siitä on etenkin kehittynyt standardi tapa julkaista avointa semanttista tietoa verkossa. Tietomalli on myös saanut osakseen jonkin verran kaupallista kiinnostusta, mutta suurin osa sen käytöstä on julkisissa projekteissa ja tutkimushankkeissa.

RDF esittää luonnostaan ja rakenteeltaan erilaisia haasteita liittyen sen visualisointiin, eikä näille haasteille ole aina yksinkertaista tai laajaa ratkaisua. Työssä tutkittiin erilaisia tapoja ratkaista visualisoinnin tuottamia haasteita. Haasteita tutkittiin monelta eri tasolta ja vaiheelta koko visualisointiprosessia, ja tutkinnalla yritettiin löytää mahdollisimman tehokkaita ratkaisuja, jotka eivät ole rajoittuneita tiettyyn käyttötarkoitukseen.

Tutkinnan tuloksien pohjalta valittiin erilaisia visualisointia selkeyttäviä metodeja visualisointityökalun kehitystä varten.

Visualisointityökaluksi kehitettiin JavaScript-kirjasto, joka on yhteydessä graafitietokantaan integroituun rajapintaan. Kokonaisuutena visualisointityökalu mahdollistaa konfiguroitavan graafien selaus- ja tutkintavälineen, jossa hyödynnetään monia erilaisia tapoja selkeyttää graafin luettavuutta. Työkalusta kuitenkin jäi toteuttamatta siltä alunperin vaaditut hallintakyvykkyydet, joilla visualisoitua graafia voitaisiin muokata. Työkalun visualisointitoteutus ei myöskään ole täysin vapaa RDF-tietomallin tuottamista visualisointiongelmista, vaan työkalu vaatii jatkokehitystä visualisointituloksien parantamiseksi.

RDF-tietomallin visualisointi voi monesti helpottaa tietomallin esittämän tiedon ymmärtämistä huomattavasti, verrattuna esimerkiksi raa'an tiedon tulkitsemiseen. Käytännössä visualisointi on kuitenkin äärimmäisen vaikeaa käyttäjälle ymmärrettävällä tavalla, mutta visualisoidun graafin selkeyttä voidaan usein parantaa erilaisilla menetelmillä. Yleispätevää ratkaisua visualisointiin ei tunnu olevan, vaan visualisoinnin selkeyden takaaminen usein vaatii joko ennakkotiedon esitettävästä tiedosta ja sen rakenteesta tai visualisoidun graafin täydellisyyttä heikentävien metodien hyödyntämistä.

Lähteet

Abele, Andrejs; McCrae, John P.; Buitelaar, Paul; Jentzsch, Anja & Cyganiak, Richard. 2017. Linking Open Data cloud diagram. Verkkoaineisto. LOD cloud diagram. <http://lod-cloud.net/>. Luettu 31.3.2018.

Alsharif, Mohammed. 2013. Semantic Web Core Technologies. Verkkoaineisto. <https://www.cse.wustl.edu/~jain/cse570-13/ftp/semantic/>. Luettu 9.11.2017.

Anderson, Janna Quitney, & Rainie, Lee. 2010. The fate of the Semantic Web. Verkkoaineisto. Pew Internet & American Life Project. <http://www.pewinternet.org/files/old-media/Files/Reports/2010/PIP-Future-of-the-Internet-Semantic-web.pdf>. Luettu 25.3.2018.

Antoniou, Grigoris & Harmelen, Frank Van. 2008. Semantic Web Primer. E-kirja. MIT Press.

Athanassiades, Aris; Kontopoulos, Efstratios & Bassiliades, Nick. 2009. Visualizing RDF documents. Artificial Intelligence Applications and Innovations, s. 151—156.

Badr, Youakim; Chbeir, Richard; Abraham, Ajith & Hassanien, Aboul-Ella. 2010. Emergent Web Intelligence: Advanced Semantic Technologies. E-kirja. Springer International Publishing.

Davies, John; Fensel, Dieter & Harmelen, Frank Van. 2003. Towards the Semantic Web. Chichester: John Wiley & Sons.

Decker, Stefan & Harth, Andreas. 2005. Optimized Index Structures for Querying RDF from the Web. 3rd Latin American Web Congress, s. 71—80.

Deligiannidis, Leonidas; Kochut, Krys J. & Sheth, Amit P. 2007. RDF data exploration and visualization. Proceedings of the ACM first workshop on CyberInfrastructure: information management in eScience, s. 39—46.

Dokulil, Jiri & Katreniakova, Jana. 2009a. RDF Visualization - Thinking Big. 20th International Workshop on Database and Expert Systems Application, s. 459—463.

Dokulil, Jiri & Katreniakova, Jana. 2009b. Using Clusters in RDF Visualization. Third International Conference on Advances in Semantic Processing, s. 62—66.

Drummond, Nick., & Shearer, Rob. 2006. The Open World Assumption. The Open World Assumption. 2017. Verkkoaineisto. The University of Manchester. <http://www.cs.man.ac.uk/~drummond/presentations/OWA.pdf >. Luettu 25.3.2018.

Graves, Alvaro. 2015. Techniques to reduce cluttering of RDF visualizations. Future Generation Computer Systems. Vol. 53, s. 152–156.

Liang Yu. 2007. Introduction to the Semantic Web and Semantic Web Services. New York: Chapman & Hall/CRC.

Linked Data - Connect Distributed Data across the Web. Verkkoaineisto. Linked Data community. <http://linkeddata.org/home>. Luettu 31.3.2018.

Li, Shuren; Yang, Tao; Hong, Na & Xu, Shiting. 2012. Assessment and Comparative Analysis of RDF Visualization Technology. International Conference on Computer Science and Service System, s. 1293—1296.

McGuinness, Deborah L & Harmelen, Frank Van. 2004. Owl Web Ontology Language Overview. Verkkoaineisto. W3C. <https://www.w3.org/TR/owl-features />. Luettu 9.11.2017.

Needleman, Mark. 2001. Rdf: The resource description framework. Serials Review. Vol. 27, nro. 1, s. 58–61.

OWL 2 Web Ontology Language Document Overview. 2012. Second Edition. Verkkoaineisto. W3C. <https://www.w3.org/TR/owl2-overview/>. Luettu 9.11.2017.

Prud'hommeaux, Eric & Seaborne, Andy. 2013. SPARQL Query Language for RDF. Verkkoaineisto. W3C. <https://www.w3.org/TR/rdf-sparql-query/>. Päivitetty 26.3.2013. Luettu 9.11.2017.

Schreiber, Guus & Raimond, Yves. 2014. RDF 1.1 Primer. Verkkoaineisto. W3C. <https://www.w3.org/TR/rdf11-primer/>. Luettu 30.10.2017.

vis.js. 2017. Verkkoaineisto. Almende B.V. <http://visjs.org/docs/network/>. Luettu 31.3.2018.

Workman, Michael. 2016. Semantic Web. E-kirja. Springer International Publishing.

Wood David. 2013. Linked Data Structured data on the Web. E-kirja. Manning Publications.