Aalto-yliopisto
Perustieteiden korkeakoulu
Master's Programme in Computer, Communication and Information Sciences

Alex Kourijoki

Linkitetyn datan validointi ja korjaus

Diplomityö
Espoo, 25. toukokuuta 2020

Valvoja:	Professori Eero Hyvönen
Ohjaajat:	Tekniikan tohtori Jouni Tuominen
	Apulaisprofessori (tenure track) Eetu Mäkelä

Aalto-yliopisto
Perustieteiden korkeakoulu
Master's Programme in Computer, Communication and Information Sciences

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Alex Kourijoki		
Työn nimi:	Linkitetyn datan validointi ja korjaus		
Päiväys:	25. toukokuuta 2020	**Sivumäärä:**	xi + 110
Pääaine:	Computer Science	**Koodi:**	SCI3042
Valvoja:	Professori Eero Hyvönen		
Ohjaajat:	Tekniikan tohtori Jouni Tuominen Apulaisprofessori (tenure track) Eetu Mäkelä		

Internet-verkossa avoimesti julkaistun linkitetyn datan määrä kasvaa ennennäkemätöntä tahtia. Näin julkaistu data on kuitenkin vain niin hyödyllistä kuin sen laatu suhteessa aiottuun käyttötarkoitukseen on. Datan kelvollistamisella pyritään pääsemään eroon datan artefakteista eli poikkeamista, jotka heikentävät datan laatua. Poikkeamat voidaan uusilla SHACL- ja ShEx-rajoitekielillä havaita muodostamalla erityisiä rajoitemuotoja, jotka ovat verkkomuotoista dataa kuvaavia osagraafeja. Poikkeamien havaitseminen on ensimmäinen askel niiden onnistuneeseen korjaamiseen. Linkitetyn datan korjaamiseen tarkoitettuja sovelluksia ei kuitenkaan aiemmin ole ollut avoimesti saatavilla.

Tässä työssä kehitettiin iteratiivinen menetelmä linkitetyn datan yhdistettyyn validointiin ja korjaamiseen. Menetelmä perustuu RDF-muotorajoitteisiin, jotka mahdollistavat virheen paikallistamisen ja korjaamisen puoliautomaattisesti. Yhdistetyn validoinnin ja korjaamisen menetelmää sovellettiin käytäntöön kehittämällä tätä mukaileva avoimen lähdekoodin sovellus ROTEVA. Sovellus on vapaasti käytettävissä verkko-osoitteessa `https://roteva.demo.seco.cs.aalto.fi`, ja sen lähdekoodi on saatavilla Aalto-yliopiston GitLab-verkkoversionhallintajärjestelmässä osoitteessa `https://version.aalto.fi/seco/Roteva`.

Sovellusta testattiin kahdella erityyppisellä koeaineistolla: SKOS-muotoisella Metatietosanastolla sekä CIDOC CRM -tietomallin mukaisella Sotasampo-projektin talvi- ja jatkosotien tapahtumagraafilla. Alustavat tulokset todentavat menetelmän toimivuuden; poikkeamia sekä muotoja onnistuttiin korjaamaan ja validoimaan käyttöliittymässä odotetusti. Suuret tietoaineistot aiheuttivat kuitenkin ohjelman suorituksessa selvästi havaittavaa hitautta, jonka ratkaiseminen optimoimalla toteutusta jätettiin sovelluksen myöhempiin jatkokehityskohteisiin.

Asiasanat:	datan validointi, linkitetty data, datan korjaus, datan laatu, semanttinen web, käyttöliittymäsuunnittelu
Kieli:	suomi

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author: Alex Kourijoki

Title:
Validation and Correction of Linked Data

Date:	May 25, 2020	**Pages:**	xi + 110
Major:	Computer Science	**Code:**	SCI3042
Supervisor:	Professor Eero Hyvönen		
Advisors:	Doctor of Science (Technology) Jouni Tuominen Assistant Professor (Tenure Track) Eetu Mäkelä		

The amount of openly published Linked Data on the Internet is increasing in an unforeseen rate. Such data is nonetheless as useful as its quality is with respect to the purposed use case. Validation introduces a method that can help one to overcome data artifacts, distinctive features that deteriorate the perceived quality of data. Contemporary Linked Data constraint languages, such as SHACL and ShEx, introduce a new tool for overcoming these artifacts: shapes. These shapes are subgraph descriptions of the schema for which the data graph must conform to. Detecting issues in the data precedes their successful correction. However, there has not previously been openly available software with the focus on correcting Linked Data.

This thesis introduces a novel approach suitable for iterative validation and correction of Linked Data. The approach is based on RDF shape constraints that allow semi-automatic correction and locating of underlying issues. An adhering open source implementation was developed in order to ensure the validity of the method. The software project was released in the Aalto Version Control System under the name ROTEVA and is freely accessible via `https://version.aalto.fi/seco/Roteva`. The software has been deployed and is readily accessible at web address `https://roteva.demo.seco.cs.aalto.fi`.

The developed software was tested against two different kinds of datasets: SKOS-modeled Metatietosanasto and CIDOC CRM inspired, event-based Winter and Continuation War event graph dataset from the WarSampo project. Preliminary results demonstrated the usability and usefulness of the approach; the software was capable of and provided means to not only iteratively validate and correct artifacts, but also the shapes used for validation. However, handling of large datasets was found to be inefficient. Optimization of the implementation was left for future work.

Keywords: data validation, Linked Data, data correction, data quality, Semantic Web, user interface design

Language: Finnish

Esipuhe

Tämä diplomityö on aloitettu aikanaan Aalto-yliopiston Semanttisen laskennan tutkimusryhmässä syksyllä 2015, jolloin useista mahdollisista rahoitusta saaneista diplomityöaiheista päädyin valitsemaan ja sain mielestäni kaikkein mielenkiintoisimman: linkitetyn datan validoinnin ja korjauksen, osana opetus- ja kulttuuriministeriön rahoittaman Avoin tiede ja tutkimus -hankkeen Linked Open Data Science Service -projektia (2014–2017). Tuolloin W3C-standardointiyhteisöllä ei ollut vielä valmista rajoitekielisuositusta linkitetylle datalle. Se työ, kuten omani, pitkittyi hieman, mutta samalla tulin oppineeksi monia hyödyllisiä taitoja ja kasvaneeni ihmisenä.

Diplomityön tulisi ihanteellisesti olla tieteellinen tutkimustyö. Tämä sai minut pohtimaan, että miten tuottaa tarpeeksi tieteellinen työ. Työn aikana tehty sovellus tuntui aluksi epäoleelliselta tämän tämän tavoitteen kannalta. Mutta Albert Einsteinin sanomaa toistaen voin nyt itselleni todeta, että itse itselleen kehittämiä ongelmia ei voi ratkaista samalla ajattelulla kuin ne on luonutkin. Ratkaisu on siis ajattelutavan muutos, jolloin pienimmätkin, mahdollisesti aluksi mitättömiltä tuntuneet uudet havainnot – kuten yksittäinen sovellus tai sen toiminta – voi nähdä arvokkaana tietona – tieteenä.

Haluan kiittää valvojaani professori Eero Hyvöstä mielenkiintoisesta tutkimusaiheesta ja täten työni mahdollistamisesta. Lisäksi haluan kiittää ohjaajiani, tekniikan tohtori Jouni Tuomista ja apulaisprofessori Eetu Mäkelää paitsi kriittisestä palautteesta, myös asiantuntevasta ohjauksesta sekä avusta teknisissä haasteissa. CSC – Tieteen tietekniikan keskus Oy:lle haluan antaa suuret kiitokset käytettävissä olleista laskentaresursseista. Lopuksi haluan kiittää ihanien lasteni äitiä, vaimoani Lindaa loppumattomasta rakkaudesta sekä vankkumattomasta uskosta ja tuesta työn loppuunsaattamiseksi.

Espoossa 25. toukokuuta 2020

Alex Kourijoki

Työssä käytetyt lyhenteet

ABox	Assertion Box
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BCP	Best Current Practice
CC	Creative Commons
CIDOC	International Documentation Committee
CRM	Conceptual Reference Model
CSS	Cascading Style Sheets
CWA	Closed World Assumption
DC	Dublin Core
DL	Description Logic
DNS	Domain Name System
DOM	Document Object Model
DQTP	Data Quality Test Pattern
DSP	Description Set Profiles
ECMA	European Computer Manufacturers Association
ELI	European Legislation Identifier

GUI	Graphical User Interface
HD	High-definition
HDT	Header-Dictionary-Triples
HMI	Human-Machine Interaction
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICOM	International Council of Museums
ICV	Integrity Constraint Validation
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IP	Internet Protocol
IRI	Internationalized Resource Identifier
ISBD	International Standard Bibliographic Description
ISBN	International Standard Book Number
ISO	International Organization for Standardization
JS	JavaScript
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
LESS	Leaner Style Sheets
LOD	Linked Open Data
MTS	Metatietosanasto
MVC	Model-View-Controller
NP	Non-deterministic polynomial time

OSLC	Open Services for Lifestyle Collaboration
OWA	Open World Assumption
OWL	Web Ontology Language
PTIME	Polynomial time
RDA	Resource Description and Access
RDF	Resource Description Framework
RDFa	Resource Description Framework in Attributes
RDFS	RDF Schema
REST	Representational State Transfer
RFC	Request for Comments
RIOT	RDF I/O Technology
ROTEVA	Työn aikana tehty sovellus
sbt	Scala Build Tool
SeCo	Semantic Computing Research Group
SHACL	Shapes Constraint Language
ShEx	Shape Expressions
ShExC	Shape Expressions Compact Syntax
SKOS	Simple Knowledge Organization System
SPARQL	SPARQL Protocol and RDF Query Language
SPIN	SPARQL Inferencing Notation
SVG	Scalable Vector Graphics
SWRL	Semantic Web Rule Language
TBC	TopBraid Composer
TBox	Terminology Box
TCP	Transmission Control Protocol

UCS	Universal Coded Character Set
UML	Unified Modeling Language
UNA	Unique Name Assumption
***NIX**	UNIX-tyylisten käyttöjärjestelmien joukko
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	Unicode Transformation Format
VV	Validointivaatimus
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
WWW	World Wide Web
XML	Extensible Markup Language
XSD	XML Schema Definition Language
YSA	Yleinen suomalainen asiasanasto
YSO	Yleinen suomalainen ontologia

Sisällysluettelo

Tiivistelmä **ii**

Tiivistelmä (englanniksi) **iii**

Esipuhe **iv**

Työssä käytetyt lyhenteet **v**

Sisällysluettelo **ix**

1 Johdanto **1**

2 Linkitetty data **5**

2.1 Tietomalli ja käsitteistö 6

2.2 Datan laadun arviointi ja virhetyypit epätäydellisessä ja muuttuvassa maailmassa 9

3 Datan validointi **14**

3.1 Validointiprosessi 15

3.2 Rajoitteet 16

3.3 Linkitetyn datan validointi 19

3.3.1 Rajoitekielet 21

3.3.2 Olemassa olevat validaattorit 35

4 Menetelmä linkitetyn datan validoimiseksi ja korjaamiseksi **40**

5 ROTEVA-rajoitevalidaattori **44**
5.1 Arkkitehtuuri 44
5.2 Käyttöympäristö 46
5.2.1 Palvelinympäristö 46
5.2.2 Ohjelman lataaminen 46
5.2.3 Ohjelman kokoaminen 47
5.2.4 Ohjelman ajaminen 48
5.3 Ohjelmallinen rajapinta 49
5.4 Graafinen käyttöliittymä 52
5.4.1 Näkymän tuottaminen 53
5.4.2 Datan syöttö, esittäminen ja manipulaatio 57
5.4.3 Rajoitteiden sovittaminen dataan 60
5.4.4 Datan validointi ja tulosten esittäminen 65
5.4.5 Virheiden korjaaminen ja korjatun aineiston tallentaminen 68

6 ROTEVAn soveltaminen koeaineistoihin **73**
6.1 Metatietosanasto 74
6.2 Sotasampo: talvi- ja jatkosodan tapahtumat 78

7 Tulosten arviointi **82**
7.1 Koeaineistoissa havaitut puutteet 83
7.2 Sovelluksen käyttökelpoisuus 84

8 Yhteenveto ja jatkokehitys **87**

Lähdeluettelo **91**

LIITTEET **103**

A REST-rajapinta **103**

B Luettelo ei-standardeista pikanäppäimistä **105**

C SPARQL-ominaisuuspolkulausekkeet SHACL-kielellä 106

D Metatietosanasto-testiaineistolle työn aikana kehitetyt rajoitemääritykset 107

E Sotasampo-testiaineistolle työn aikana kehitetyt rajoitemääritykset 109

Luku 1

Johdanto

Internetissä julkaistun *linkitetyn avoimen datan* (engl. linked open data, LOD) määrä on semanttisen webin tekniikoiden kehittyessä ja vakiintuessa kasvanut ennennäkemätöntä tahtia. Esimerkiksi The Linked Open Data Cloud -palvelussa[1] luetteloitujen, toisiinsa viittaavien RDF-datajoukkojen (Resource Description Framework) [113] määrä on kasvanut vuoden 2007 toukokuun 12:sta datajoukosta vuoden 2019 maaliskuuhun mennessä 1239:ään. Samalla ylläpidettyjen datajoukkojen RDF-kolmikoiden [31] määrä on kasvanut nopeasti; esimerkiksi laaja-alaisen, Wikipedia-tietoperustaisen[2] DBpedia-projektin[3] vuoden 2014 julkaisu käsitti kaikkiaan 3 miljardia kolmikkoa, vuoden 2015 ensimmäinen julkaisu (niin kutsuttu 2015-04-julkaisu) jo 6,9 miljardia ja vuoden 2016 toinen julkaisu (2016-10) jo huimat 13 miljardia kolmikkoa[4]. Muita mainittavan suuria viimeaikaisia datapalveluita ovat esimerkiksi LOD-a-lot[5] [40], joka tarjoaa yhtenä indeksoituna HDT-tiedostona (Header-Dictionary-Triples) [41] 28 miljardia kolmikkoa yli 650 000 datajoukosta ja LODStats[6] [36] (149 miljardia kolmikkoa ja 2973 datajoukkoa).

Suurten tietomäärien kerääminen ja julkaiseminen on kuitenkin vasta ensiaskel linkitetyn datan täysimittaiseen hyödyntämiseen, sillä kuten Zaveri ja kumppanit kirjallisuuskatsauksessaan [112] toteavat, data on vain yhtä hyödyllistä kuin sen laatu on. Datan laadun kelpoisuuden tarkastamiseen käytetään niin kutsuttuja *validaattoreita*, jotka validoivat eli tarkastavat sen

[1] `https://lod-cloud.net`

[2] `https://wikipedia.org`

[3] `https://dbpedia.org`

[4] Kyseiset julkaisut ovat vapaasti ladattavissa osoitteesta `https://downloads.dbpedia.org`.

[5] `http://lod-a-lot.lod.labs.vu.nl`

[6] `http://lodstats.aksw.org`

käypäisyyden jonkin määritelmän (rajoitteen) suhteen. Nämä näin havaitut laatuvirheet tulee tietysti korjata dataan, mutta kuten Labra Gayo ja kumppanit kirjassaan [68] huomauttavat, RDF-datan muokkaaminen (korjaaminen) on virhealtista ja käyttäjäepäystävällistä työtä. He muistuttavat kuitenkin, että jos datan rakenne (tietomalli) on etukäteen tiedossa, on ohjelmallisesti mahdollista helpottaa tätä dataan tehtävää korjaustyötä. Tällaisia ohjelmia ei ole kuitenkaan syystä tai toisesta juuri tehty, vaikka erilaisia RDF-datan validaattoreita onkin ollut saatavilla jo pitkään. Osasyynä tilanteeseen saattaa olla se, että aiemmin käytössä olleet rajoitekielet (tai sellaisiksi soveltuvat kielet kuten RDF-datan yleinen kyselykieli SPARQL [SPARQL Protocol and RDF Query Language] [50]) eivät ole tarpeeksi hyvin tukeneet tällaista yhdistettyä validointi- ja korjausprosessia.

Vuonna 2014 W3C-konsortio[7] (World Wide Web Consortium) perusti RDF Data Shapes -työryhmän[8] tuottamaan W3C-tason RDF-validointikielisuosituksen. Tavoitteena oli tuottaa ja standardoida kieli, joka soveltuu datagraafien rakenteiden (osagraafien) kuvailuun ja validointiin. Työryhmä sai työnsä valmiiksi 2017, jolloin se julkaisi valmiin SHACL-rajoitekielisuosituksen (Shapes Constraint Language) [63], joka yhtenä potentiaalisena rajoitekielivaihtoehtona mahdollistaa paitsi rakenteisen RDF-datan validoimisen, myös sen korjaamisen. Suosituksesta kilpaillut ShEx-rajoitekieli (Shape Expressions) [95] tarjoaa myös varsin varteenotettavat ominaisuudet, ja Labra Gayo ja kumppanit toteavatkin [68], että nämä kaksi rajoitekieltä tulevat näyttelemään tärkeää roolia sekä RDF-tietomallin tulevaisuudessa että koko semanttisen webin työkalupakissa, sillä alati kasvavassa semanttisen datan maailmassa yhä useammilla sovelluksilla on tarve kuluttaa tällaista dataa, jolloin vastaavasti tarve laadukkaalle ja yhteistoiminnalliselle, RDF-validoidulle datalle tulee täysin perustavanlaatuiseksi.

Tämän työn ensimmäisenä tavoitteena (ja ainoana tutkimuskysymyksenä) on kehittää menetelmä, joka mahdollistaa linkitetyn datan yhdistetyn validoinnin ja korjaamisen. Toisena tavoitteena on tuottaa tämän mallin mukainen ohjelmallinen, avoimen lähdekoodin ja lisenssin työkalu. Kehitettävällä ohjelmalla tulee olla tilaton, REST-perustainen (Representational State Transfer) [43] ohjelmointirajapinta sekä selainperustainen, graafinen käyttöliittymä linkitetyn datan validoinnin ja korjaamisen helppokäyttöisyyden tuottamiseksi. Työkalun käytettävyyteen (helppokäyttöisyyteen) tullaan panostamaan erityisesti. Toissijaisena, kolmantena tavoitteena on kehittää työkalusta täysipainoinen, moniulotteinen RDF-datan validointi-, korjaus-, kysely- ja muuntoalusta. Lisäksi työn ohjelmallisessa osassa käytettäväksi valitta-

[7] `https://www.w3.org`
[8] `https://www.w3.org/2014/data-shapes/`

van RDF-tietomallin validointikielen tulee olla tarpeeksi ilmaisuvoimainen. Seuraavassa on listattu muutamia poimintoja tällaisista työn ohjelmallisen osan kannalta riittävän ilmaisuvoiman edellytyksistä Labra Gayon ja kumppanien kirjassansa [68] tunnistamista[9] kehittyneeltä RDF-validointikieleltä vaadittavista validointivaatimuksista (VV):

1. Korkean tason kieli
2. Tiiviit ja ymmärrettävät rajoitteet
3. RDF-perustainen syntaksi[10]
4. Kielen mukaisen validointiprosessin pystyttävä palauttamaan validoinnin kohteena oleva solmu
5. Mahdollisuus esittää rajoitteita tuleville ja lähteville kaarille
6. Mahdollisuus kuvata poluilla graafin solmujen välisiä suhteita
7. Solmun tyyppi oltava rajattavissa johonkin kolmesta RDF-datan tyypistä (nimetty/anonyymi solmu, literaaliarvo) tai näiden yhdistelmiin
8. Mahdollisuus kuvata solmun mahdolliset datatyypit
9. Mahdollisuus esittää kielimääreisten literaaliarvojen rajoitteita
10. Tuki Boolen operaattoreille $\wedge$, $\vee$ ja $\neg$
11. Tuki monikertarajoitteille (kardinaliteettirajoitteet)
12. Tuki suljetuille (vain nämä kolmikot) ja avoimille muodoille
13. Mahdollisuus vertailla ominaisuuden arvoa kiinteään arvoon
14. Mahdollisuus vertailla eri ominaisuuksien arvoja keskenään
15. Mahdollisuus vertailla eri solmujen ominaisuuksien arvoja keskenään[11]
16. Tuki aritmeettisille operaatioille
17. Mahdollisuus abstrahoida, modularisoida ja yhdistellä rajoitteita

[9] Osan vaatimuksista he paljastavat lainanneensa SHACL-rajoitekielen käyttötapaukset ja vaatimukset -dokumentista [100].

[10] Tämä vaatimus on hieman Labra Gayon ja kumppanien vaatimusta tiukempi; he vaativat kirjassaan rajoitekieleltä vain tuttua syntaksia.

[11] Tätä rajoitetta ei täsmällisesti ole mainittu Labra Gayon ja kumppanien listauksessa.

18. Tuki sekä koko datagraafiin sovellettaville rajoitteille että yksittäissolmu- ja valintaperustaisille kohdesolmurajoitteille
19. Tuki validointivirheiden tulostamiselle
20. Tuki validointiraportille

Työssä käytetyissä esimerkeissä tullaan pitäytymään eksplisiittisesti esitetyissä kolmikoissa; päättely ja RDF-graafien täydentäminen rajataan työn ulkopuolelle, vaikka nämä voisivatkin tuoda huomattavaa lisäarvoa työn aikana kehitettävään validaattoriin[12].

Tämän työn rakenne on seuraavanlainen: luvussa 2 käydään läpi linkitetyn datan käsitteistö, tietomalli ja siinä mahdollisesti esiintyvät datan laatuun vaikuttavat virhetyypit. Luku 3 esittelee yleisellä tasolla datan validoinnin sekä validoinnissa käytettävien rajoitteiden perusperiaatteet, joiden lisäksi siinä käsitellään nykykirjallisuuteen perustuen linkitetyn datan validoinnissa käytetyt rajoitekielet ja työn kannalta tärkeimmät olemassa olevat validaattorit. Luku 4 vastaa ensimmäiseen tutkimustavoitteeseen esitellen uuden, iteratiivisen menetelmän linkitetyn datan validoimiseksi ja korjaamiseksi. Toisen tutkimustavoitteen mukainen työkalu ja tämän käyttöönotto sekä käyttäminen on kuvattu luvussa 5. Työn aikana kehitetyn järjestelmän toimivuutta testataan luvussa 6 tarkemmin esitettyihin, koeaineistoina käytettyihin Metatietosanastoon sekä Sotasampo-projektin talvi- ja jatkosotien tapahtumagraafiin, minkä jälkeen luvussa 7 kuvataan ja arvioidaan analyyttisesti sovelluksen koeaineistojen testaamisessa löytämät virheet ja tulokset sekä arvioidaan yleisesti koko sovelluksen toiminta, rajoitukset ja käyttökelpoisuus. Luku 8 päättää työn yhteenvetoon, aiheesta heränneisiin ajatuksiin sekä potentiaalisiin jatkokehityskohteisiin.

[12]Kuten Melo toteaa väitöskirjassaan [78], päättely on yksi tärkeimmistä semanttisen webin lähtökohdista, sillä se mahdollistaa uusien johtopäätösten tekemisen, vähentäen näin datagraafin epätäydellisyyttä. Mutta kuten hän lisää, tällainen päättely saattaa toisinaan olla ongelmallista, sillä se voi väärän datan tapauksessa johtaa väärien johtopäätösten leviämiseen ja täten jopa koko datagraafin virheellisyyden kasvuun.

Luku 2

Linkitetty data

Internet-tietoverkossa toimiva WWW-verkko (World Wide Web) on jo sen syntyhetkistä asti perustunut koneluettaviin ja -haettaviin, toisiinsa osoittaviin URL-osoitteisiin (Uniform Resource Locator) [16]. Näillä osoitteilla ja niiden osoittamalla datalla on kuitenkin alkujaan ollut se heikkous, että ne eivät ole olleet koneymmärrettäviä. Tätä puutetta paikkaamaan on kehitetty semanttinen web[1], jonka tekniikat mahdollistavat dataa kuvailevan datan eli niin kutsutun metadatan tuottamisen ja välittämisen. Kuten Tim Berners-Lee – nykyisen WWW:n kehittäjä – asian näkee [14], kuka tahansa voi tehdä mielivaltaisesta datastaan linkitettyä dataa toteuttamalla seuraavat neljä asiaa:

1. Nimeä datasi asiat ja käsitteet URI-tunnisteilla (Uniform Resource Identifier) [18].
2. Käytä HTTP-protokollaa (Hypertext Transfer Protocol) [42] hyödyntäviä URI-tunnisteita, jotta kuka tahansa voi seurata näitä datalähteelle.
3. Kun joku tai jokin (käyttäjä) seuraa URI-tunnistetta, tarjoa hyödyllistä tietoa semanttisen webin tekniikoita käyttäen.
4. Sisällytä dataasi linkkejä toisiin URI-tunnisteisiin muodostaaksesi verkon/verkkoja, jotta käyttäjä voi löytää enemmän dataa.

Näin muodostetussa linkitetyssä datassa voidaan kuvata ja etsiä mitä tahansa dataa, mikä mahdollistaa näin pelkän hypertekstiperustaisen verkon muuntamisen aiempaa laajemmaksi, aidoksi ja tietokoneymmärrettäväksi dataverkoksi.

[1] `https://www.w3.org/2001/sw/`

Tämä luku on kaksijakoinen: ensiksi aliluvussa 2.1 paneudutaan kaiken semanttisen webin taustalla oleva tietomalliin ja käsitteistöön tämän työn puitteissa, jonka jälkeen aliluvussa 2.2 käydään läpi linkitetyn datan laadun arvioinnissa käytettyjä ulottuvuuksia ja metriikoita.

2.1 Tietomalli ja käsitteistö

W3C asetti jo vuonna 1997 työryhmän[2] vastaamaan uuden WWW-verkkoon soveltuvan yhteismitallisen tietomallin kehittämisestä ja standardoimisesta. Työryhmän kehittämä suositus julkaistiin vuonna 1999, luoden alkupisteen nykyään tunnetulle semanttiselle webille. Ehdotettu tietomalli, joka myös RDF-viitekehyksenä tunnetaan, mahdollistaa yhteentoimivan ja koneymmärrettävän datan esittämisen eri Internet-sovellusten välillä ilman toimialakohtaisia rajoitteita. [73]

RDF-tietomalli perustuu subjekti-predikaatti-objekti-kolmikoihin, joissa *resursseja*[3] (subjekti) kuvataan *ominaisuuksilla*[4] (predikaatti), jotka saavat *arvoja*[5] (objekti). Tällaisia kolmikoita kutsutaan myös RDF-lauseiksi. Samoja resursseja käsittelevät lauseet muodostavat laajempia RDF-osagraafeja ja nämä osagraafit puolestaan kokonaisia graafitiedostoja[6]. Nimetyt resurssit esitetään URI-tunnisteilla[7], jotka ovat yksiselitteisiä ja kaikenkattavia[8]. Nimetön (anonyymi) resurssi on nimensä mukaisesti nimeämätön. Tästä huolimatta niitä voidaan hyödyntää RDF-graafeissa esimerkiksi reifikaatiossa (väitteiden esittämisessä väitteistä) ja rakenteisten entiteettien muodostamisessa. Literaalit ovat sallittuja arvoja vain objekteissa – ne ovat dataprimitiivejä, joille ei voi antaa ominaisuuksia. On huomioitavaa, että mahdollinen kielikoodi on osa merkkijonomuotoista literaalia, eikä sillä ole erillistä ilmentymää tietomallissa. Yhteinen kattokäsite kaikille literaaleille ja resursseille on *RDF-termi*. Käsitettä *solmu* saatetaan käyttää, kun puhutaan graafimuotoisen datan subjektista tai objektista. [73]

[2] W3C RDF Model and Syntax Working Group

[3] Resurssi on abstrakti objekti, joka voi kuvata mitä tahansa asiaa tai käsitettä.

[4] Ominaisuus on nimetty resurssi, joka kuvaa subjektin suhdetta objektiin.

[5] Sallittujen arvojen arvojoukko riippuu predikaatin määrittelystä.

[6] Tiedostot on sarjallistettava jollain semanttisen webin teknologialla. XML-perustainen (Extensible Markup Language) [21] RDF/XML-formaatti [46] on kaikkien RDF-työkalujen tukema sarjallistamismuoto.

[7] Jos tunnisteessa on tiedon osaa kuvaava ankkuri (niin kutsuttu URI-fragmentti), kohdistuu väite tähän resurssin osaan.

[8] Mille tahansa asialle tai käsitteelle voidaan luoda yksilöivä URI-tunniste.

RDF 1.1 [31] on W3C:n julkaisema semanttisen webin uusin abstrakti viitekehys, jossa on kehitetty alkuperäisen RDF-spesifikaation ja tämän perusteella laaditun, aiemmin käytössä olleen vuoden 2004 RDF 1.0 -suosituksen [76] tietomallia eteenpäin. Wood on koostanut muutoksista tiedotteen [110], joista tämän työn kannalta merkittävimmät on lueteltu alla.

- Siirtyminen URI-tunnisteista IRI-tunnisteiden (Internationalized Resource Identifier) [35] käyttöön. Mahdollinen tunnistefragmentti tulkitaan IRI-protokollan puitteissa aiemmin käytetyn RDF/XML-esitysmuodon sijaan.
- Uudet sarjallistamismuodot (RDF/XML ei ole enää ainoa suositeltava formaatti).
- Nimettömät resurssit ovat vain ja ainoastaan paikalliseen (tiedoston tai RDF-tietokannan sisäiseen) käyttöön kelpaavia tunnisteita.
- Uutena käsitteenä on esitetty *RDF-datajoukko*, joka on määritelty RDF-graafien kokoelmaksi. RDF 1.1:ssä on mahdollista myös esittää ja nimetä useita graafeja samassa tiedostossa uusilla, tietomallin tukemilla sarjallistamisformaateilla.
- Kielikoodien tulee olla IETF-järjestön[9] (Internet Engineering Task Force) ylläpitämän RFC 5646/BCP 47 -normin (Request for Comments, Best Current Practices) [86] mukaisia ja kaikki kielikoodilliset literaalit saavat *http://www.w3.org/1999/02/22-rdf-syntax-ns#langString* (IRI)-tunnisteisen[10] tietotyypin.

IRI-protokollan mukaiset tunnisteet täydentävät aiemmin käytössä olleen URI-protokollan ASCII-merkistöperustaisia (American Standard Code for Information Interchange) [24] URI-tunnisteita mahdollistamalla yleismaailmallisen UCS-merkistön (Universal Coded Character Set) määrittävän kansainvälisen ISO/IEC 10646 -standardin[11] (International Organization for Standardization, International Electrotechnical Commission) [5] mukaisten kirjainten käytön. Koska ASCII-merkistön kirjaimet sisältyvät UCS-merkistöön, on jokainen URI-tunniste myös IRI-tunniste. Kaikki IRI-tunnisteet voidaan myös muuntaa vain URI-tunnistekäytäntöä tukevien sovellusten käyttämiksi URI-tunnisteiksi. [35] Tyypillinen merkistökoodaus IRI-tunnisteita

[9] `https://ietf.org/`

[10] Tässä työssä IRI-tunnisteita merkitään yleisesti korostettuina ilman kulmasulkeita.

[11] Myös Unicode Consortium osallistuu tämän standardin kehittämiseen ja julkaisee aika ajoin tämän kanssa jälkijättöisesti yhteensopivia Unicode-merkistöstandardiversioita.

sisältävälle tiedostolle on UTF-8 (Unicode Transformation Format) [111], joka mahdollistaa esimerkiksi ä- ja ö-kirjainten käytön näissä tunnisteissa. Yleinen IRI-tunnisteiden välinen yhtäsuuruus määritellään kirjain kirjaimelta -vertailulla [35].

RDF 1.1 -sarjassa julkaistut/päivitetyt sarjallistamisformaatit N-Triples [12], N-Quads [23], RDF/XML [46], TriG [19] ja Turtle [13] esittävät kaikki eri tapoja esittää RDF-tietomallin mukaisia lauseita. Näiden lisäksi W3C julkaisi JSON-perustaisen (JavaScript Object Notation) [3] JSON-LD-formaatin (JavaScript Object Notation for Linked Data) [96] ja päivitetyn, HTML-merkintäkielen (Hypertext Markup Language) [38] mukaisia dokumentteja rakenteisella RDF-tiedolla laajentavan RDFa-suosituksen (Resource Description Framework in Attributes) [6], joista JSON-LD:tä on hyödynnetty ihmisluettavan Turtlen ohella tämän työn esimerkkikoodilistauksissa. On huomioitavaa, että kaikissa edellä mainituissa sarjallistamismuodoissa ei ole mahdollista esittää kaikkia RDF 1.1:n mukaisia ominaisuuksia, kuten esimerkiksi RDF-datajoukkojen määrittelyjä[12].

Anonyymien solmujen identiteetti täytyy joskus voida säilyttää siirrettäessä tietoa eri ohjelmien tai rajapintojen välillä. Tällöin jokaiselle anonyymille resurssille tulee kehittää uusi, yksilöllinen IRI-tunniste. Tämä voidaan toteuttaa käyttäen skolemointia [31]:

1. Käytä skeemaa[13], joka tukee *yleisesti tunnettuja* (engl. well-known) IRI-tunnisteita rekisteröidyllä *genid*-tunnisteella.

2. Aloita IRI-tunnisteen polkukomponentti kiinteämittaisella merkkijonolla */.well-known/genid/*.

3. Luo hallitsemaasi tunnisteympäristöön uusi, yksilöllinen tunniste. Käytetty muoto voisi esimerkiksi HTTP-skeeman tapauksessa olla kaavan *http://$DOMAIN/.well-known/genid/$UID* mukainen, jossa $DOMAIN on verkkotunnus ja $UID luotu yksilöllinen ja mahdollisesti takaisin alkuperäiseen[14] nimettömään resurssiin sillattava tunniste.

Skolemointiprosessin lopputuloksena anonyymistä resurssista on muodostettu niin kutsuttu *Skolem IRI -tunniste*, joka on jaettavissa ja käytettävissä myös alkuperäisen tiedoston tai RDF-tietokannan lukeneen järjestelmän ulkopuolella. Skolem IRI -tunniste takaa sen, että eri järjestelmät voivat tun-

[12]Turtle-esitysmuoto ei esimerkiksi mahdollista graafien nimeämisiä.

[13]Esimerkiksi HTTP:tä tai tämän laajennosta HTTPS:ää (Hypertext Transfer Protocol Secure) [90].

[14]Kaikissa RDF:n sarjallistamismuodoissa tämä ei ole mahdollista.

nistaa käsittelevänsä samaa nimetöntä resurssia ja täten tuottaa toistensa kanssa vertailukelpoisia tuloksia ja kuvauksia.

RDF-viitekehyksen tarkempi semanttinen malli [51], joka muun muassa määrittelee RDF-termien ja niiden välisten suhteiden tulkinnan ja täten päättelyn, on julkaistu omana spesifikaationaan. RDF-viitekehys olisi täysin tietämätön lauseissa esitetyistä väittämistä ja niissä käytetyistä ominaisuuksista ja luokista, ellei siihen oltaisi sisäänrakennettu kiinnitettyä sanastoa[15], joka määrittelee tietomallin komponentit ja niiden väliset suhteet. Tätä tukemaan on lisäksi määritelty virallinen RDF Schema -sanasto (RDFS) [22], joka on RDF-perustainen RDF-viitekehyksen semanttinen laajennos. RDFS tarjoaa tavan kuvata samankaltaisia resursseja ja niiden luokkia, määrittäen esimerkiksi niiden välisten ominaisuuksien määrittelyjoukon ja maalijoukon [22]. Sanastojen sisäiset IRI-tunnisteet on tavanomaista kuvata yhteisellä alimerkkijonolla, jota myös IRI-nimiavaruudeksi kutsutaan. Joissakin sarjallistamisformaateissa on tapana lyhentää tällaiset nimiavaruudet etuliitteillä eli prefikseillä, jolloin esimerkiksi *http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral* voidaan lyhentää muotoon *rdf:XMLLiteral*, kun *rdf*-etuliitteen nimiavaruus on määritelty olemaan *http://www.w3.org/1999/02/22-rdf-syntax-ns#*. [31]

2.2 Datan laadun arviointi ja virhetyypit epätäydellisessä ja muuttuvassa maailmassa

Ideaalimaailmassa data olisi linkittynyttä, oikeellista ja täydellistä. Muuttuvassa maailmassa tähän ei kuitenkaan päästä kuin tarkkaan rajatuissa aineistoissa ja niissäkin harvoin, jos koskaan. Tästä syystä semanttisen webin on hyvä olla virhesietoista ja eteenpäin yhteensopivaa suunnittelultaan, aivan kuten esimerkiksi verkkoselainten tukema HTML-standardikin on. Tämä varmistaa sekä vanhojen, suurella työllä tehtyjen järjestelmien rikkoutumattoman käytön datan muuttaessa muotoaan että datan uudelleenkäytön.

Listauksessa 2.1 on esitetty virheellinen RDF-graafi. Siinä syntaksi rikkoo suoraan RDF-spesifikaation sääntöjä[16], estäen täysin tällaisen graafin koneellisen jäsentämisen ja tulkitsemisen. Edellä esitetty on vain yksi esimerkki virheellisestä datasta ja sen vaikutuksista datan kelpaavuuteen ja täten käytössä havaittuun laatuun. Koska datan käyttökelpoisuuden varmistaminen on perustavanlaatuinen ja koko linkitetyn datan sovellusalaa koskeva haaste,

[15]Sanasto on kokoelma IRI-tunnisteita ja niiden määrittelyitä.

[16]Rivillä 8 olevaa literaalia ei ole päätetty lainkaan.

```
1  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3
4  <Kala> rdf:type rdfs:Class ;
5         rdfs:comment "Kala on vedessä elävä eläin"@fi .
6
7  <Hauki> rdfs:subClassOf <Kala> ;
8          rdfs:comment "Hauki on petokala@fi .
```

Listaus 2.1: Esimerkki virheellisestä RDF-datasta.

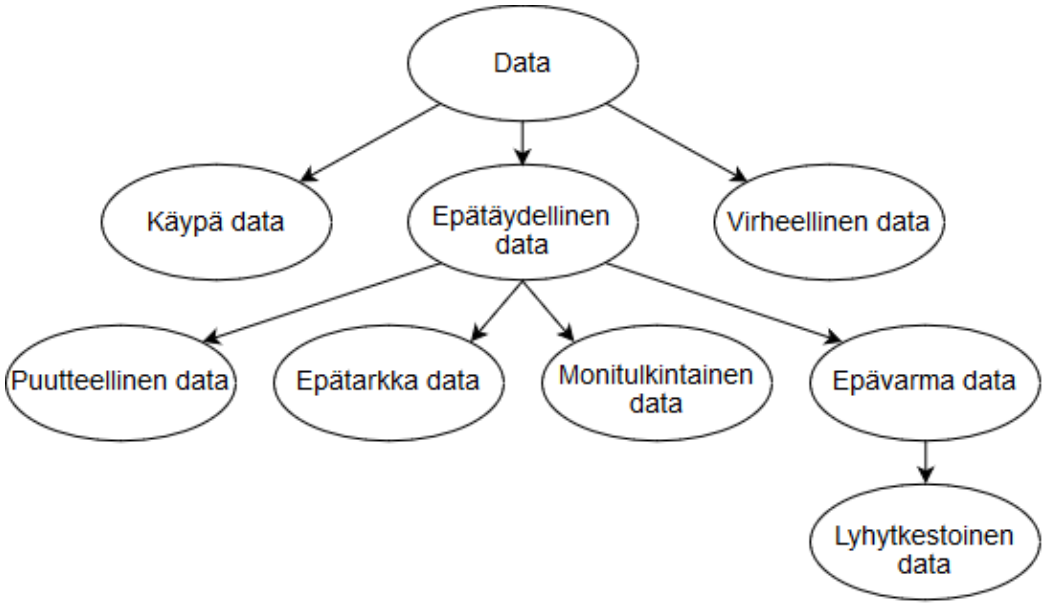
on käyttökelpoisuuden arviointiin kehitetty useita erilaisia metriikoita, joista ehkä tunnetuin on Tim Berners-Leen viiden tähden malli [14]. Mallissa tähtiä annetaan datan ominaisuuksien perusteella inkrementaalisesti seuraavien kriteerien perusteella:

1. Saatavilla Internetissä missä tahansa formaatissa
2. Saatavilla koneluettavassa, rakenteisessa muodossa
3. Formaatin oltava avoin (ei-yksityisomistuksellinen)
4. Avointen W3C-standardien käyttö asioiden ja käsitteiden tunnistamiseksi, kuvaamiseksi ja niihin osoittamiseksi
5. Datan linkittäminen muiden luomaan dataan kontekstin luomiseksi

Lisäksi dataa voi nimittää edellisten sääntöjen puitteissa yhtä monen tähden linkitetyksi avoimeksi dataksi, jos se on julkaistu avoimella lisenssillä, joka ei estä datan vapaata käyttöä ilman lisämaksua. Esimerkiksi CC Nimeä 4.0 Kansainvälinen -lisenssi[17] (Creative Commons) vastaa näitä ehtoja. [14] Edellä esitetty metriikka ei kuitenkaan aivan täysin vastaa tämän työn kannalta oleelliseen kysymykseen datan laadusta, sillä se kuvaa dataa enemmänkin kokonaisuutena ja ulkopuolelta katsottuna, eikä täten kunnolla ota huomioon esimerkiksi datan sisäisen johdonmukaisuuden ja oikeamuotoisuuden tärkeää ulottuvuutta. Koska kaikkien olemassa olevien laatumetriikoiden tarkka esittely on tämän työn laajuuden ulkopuolella, kehotetaan lukijaa tutustumaan kirjoittajan tietoon ajantasaisimpaan ja kattavimpaan, Zaverin ja kumppanien tuottamaan kirjallisuuskatsaukseen [112]. Tämän lisäksi esimerkiksi Sam ja kumppanit [92], Vandenbussche ja kumppanit [106], Feeney ja kumppanit [39], Ali ja Alchaita [7], Radulovic ja kumppanit [89],

[17] `https://creativecommons.org/licenses/by/4.0/deed.fi`

Galárraga ja kumppanit [45], Debattista ja kumppanit [33] ovat viime aikoina tehneet merkittävää tutkimusta aiheen tiimoilta. Tähän tutkielmaan riittävän linkitetyn datan laatu-ulottuvuuksien esittelyn on tarjonnut Laitio diplomityössään [72], jossa hän ansiokkaasti esittää, jaottelee ja määrittelee tunnistamansa datan laatuun vaikuttavat ulottuvuudet. Kuvassa 2.1 on esitetty nämä Laition tunnistamat linkitetyn datan laatutyypit.



Kuva 2.1: RDF-datan laatutyypit jaoteltuna Laition mukaan. [72]

Kuten kuvasta 2.1 nähdään, on Laitio jakanut laatutyypit kolmeen päähaaraan: *käypään dataan*, *epätäydelliseen dataan* ja *virheelliseen dataan*. Näistä ensimmäiseen kuuluu hänen mukaansa kaikki data, joka ei sisällä epätäydellisen datan virhetyyppejä. Nämä epätäydellisen datan virhetyypit ovat epätäydellisen datan alijaotteluiden ulottuvuuksia, sillä epätäydellinen data on Laition hahmottelemassa jaottelussa yläkategoria kaikille ei-toivotuille datan ominaisuuksille. Nämä ominaisuudet voivat olla ei-toivottuja tarkalleen jonkin tietyn sovelluksen kontekstissa tai yleisemmin. Erona virheelliseen dataan Laitio näkee epätäydellisen datan täyttävän käyvän datan vaatimukset, mutta datalta vaadittu laatu ei kaikkien virhetyyppien osalta välttämättä yllä hyväksyttävälle tasolle[18], kun taas virheellisen datan tiedetään olevan virheellistä, mahdollisesti jopa datan semantiikan tasolla. Lisäksi epätäydellinen

[18]Hyväksyttävä taso on datajoukon käyttäjän esimerkiksi datan tarkkuudelle asettama vähimmäistaso.

data on Laition mukaan korjattavissa kelvolliseksi dataksi suhteellisen pienellä työllä, toisin kuin virheellisen datan tapauksessa. Tämä ei kuitenkaan tarkoita Laition mukaan sitä, että virheellinen data olisi täysin hyödytöntä ja heti poistettava, sillä sitä voidaan hyödyntää esimerkiksi joissain sovelluksissa johonkin käsitteeseen tai asiaan liittyvien yleisten väärinkäsitysten eksplisiittiseen vääräksi merkitsemiseen. [72]

Epätäydellisen datan Laitio on puolestaan jakanut kuvan 2.1 mukaisesti puutteelliseen dataan, epätarkkaan dataan, monitulkintaiseen dataan ja epävarmaan dataan, joista viimeisestä on vielä nostettu esiin lyhytkestoisen datan ulottuvuus. Näistä puutteellinen data vastaa Laition mukaan tilannetta, jossa data ei validiudestaan huolimatta kuvaa tai esitä tarpeeksi tarkkaan jotakin asiaa tai käsitettä. Laitio nostaa työssään esimerkiksi vaillinaisen historiallisen tekstin, joka kokotekstin sijaan sisältää vain joitakin katkelmia siitä. Epätarkalla datalla Laitio tarkoittaa sovelluskohtaisesti liian rakeista tai ylimalkaista datan kuvaamista – sama data saattaa nimittäin olla jollekin toiselle sovellukselle täysin kelvollista ja käypää dataa. Datan monitulkintaisuus, jota ilmentää esimerkiksi homonyymien[19] aiheuttama haaste sanojen oikeiden vastineiden tunnistamisessa automatisoidussa luonnollisen kielen koneprosessoinnissa, on Laition mukaan yksi tapaus, toisen esimerkin ollessa useiden eri lähteiden antamat ristiriitaiset tulkinnat kuten esimerkiksi asiantuntija-arviot samasta historiallisesta tapahtumasta, joiden välille ei voida muodostaa eroa luotettavuudessa. Epävarman datan laatu-ulottuvuuden eli huolen datan todenmukaisuudesta sekä epävarmuustekijöiden syyt todenmukaisuuden kyseenalaistamiseen Laitio jakaa työssään vielä kahtia objektiivinen (esimerkiksi epäkuntoinen mittari) ja subjektiivinen (esimerkiksi henkilön mielikuva jostain asiasta) -jaolla, joihin molempiin voidaan lisäksi hänen mukaansa liittää todennäköisyysarvio datan paikkansapitävyydestä. Viimeisenä Laitio nostaa vielä tarkempaan tarkasteluun datan väliaikaisuuden datan epävarmuuden aliulottuvuutena, sillä joissakin tapauksissa data voi olla käypää ja suhteellisen varmaa luomishetkellä, mutta sen nopea vaihtuvuus (esimerkiksi tuulen suunnan ja nopeuden mittarointi- ja tallentamistapauksessa) johtaa epätäydelliseen dataan. [72]

Kuten taulukosta 2.1 nähdään, käsittelevät nämä edellä esitetyt Laition tunnistamat laatutyypit monia Zaverin ja kumppanien kirjallisuuskatsauksen [112] linkitetyn datan laatu-ulottuvuuksien ryhmittelyjä[20]. Tämän aliluvun kattavuuden takaamiseksi on huomioitavaa, että tavoitettavuusulottu-

[19]Homonyymi on sana, jolla on useita eri merkityksiä. Esimerkiksi konna (rikollinen tai lyhyelmä kilpikonnasta) on tällainen suomen kielen sana.

[20]Zaverin ja kumppanien käyttämä luokittelu perustuu laajennettuun ja muunneltuun Wangin ja Strongin tutkimusartikkelissa [108] laadittuun laatuattribuuttien ryhmittelyyn.

Laatu-ulottuvuus (Zaveri ja kumppanit)	**Laition työssä katettu osio**
Esittämismuoto	Implisiittinen viittaus käyttää vakiintuneita sanastoja
Kontekstuaalisuus	Käsitelty työssä / otettu huomioon joissakin laatutyypeissä
Sisäinen	Epätäydellinen data niiltä osin kun se on riippumaton käyttäjän kontekstista
Dynaamisuus	Lyhytkestoinen data
Tavoitettavuus	-
Luottamus	Epävarma data

Taulukko 2.1: Laition tunnistamat ja käsittelemät laatutyypit [72] suhteessa Zaverin ja kumppanien linkitetyn datan laatu-ulottuvuuksien jaotteluun [112].

vuutta ei juuri käsitellä Laition työssä; sen Zaveri ja kumppanit määrittelevät viisiulotteisena ryhmittelynä sisältäen käytettävyyden, lisensoimisen, samoja entiteettejä esittävien tunnisteiden välisen linkittämisen, turvallisuuden ja tehokkuuden. Lisäksi erityisesti esittämismuotoulottuvuudesta on käsitelty Laition työssä vain pieni osa: se käsittää kokonaisuudessaan datan ymmärrettävyyden, tulkinnan, monipuolisuuden sekä esitysmuodon tiiviyden ja yhdenmukaisuuden näkökulmat.

Luku 3

Datan validointi

Datajoukkojen kasvaessa yhdeksi yhä enenevässä määrin tärkeämmäksi toiminnoksi on noussut näiden sisältämän sisällön ristiriitaisuuksien hallinta. Tähän tarpeeseen on jo ennen luvussa 2 esitetyn linkitetyn datan keksimistä kehitetty validaattoreita, jotka yleensä toteuttavat jonkin tavan havaita ja esittää käyttäjälle datassa piileviä puutteita ja virheitä. Validointi eli datan kelpoisuuden tarkastaminen perustuu ohjelmallisiin rajoitteisiin, jotka nimensä mukaisesti rajoittavat dataa sen sisältämien ominaisuuksien perusteella. Nämä datan ominaisuudet (ja täten validointi) voidaan puolestaan jakaa kahtia seuraavasti:

- Syntaktiset ominaisuudet
  - Syntaktinen eli kirjoitetun datan lauseopillinen oikeellisuus
- Semanttiset ominaisuudet
  - Datan sisältämän tiedon semanttinen eli merkitysopillinen oikeellisuus

Jotkin validaattorit voivatkin tarkastaa sille syötteenä annetun datan vain esimerkiksi syntaktisen oikeellisuuden perusteella, jolloin saadaan selville vain käytetyn kielen mahdolliset lauseopin vastaisuudet, mutta ei sisällön epäjohdonmukaisuuksia. Tällainen suomen kielen validaattori voisi esimerkiksi hyväksyä syötteen ”kissat ovat koiria”, sillä tämä on kieliopillisesti oikein muodostettu suomen kielen lause. Esitetyssä väitteessä ei kuitenkaan ole järkeä olettaen[1], että kissana oleminen on poissulkevaa koirana olemisen kans-

[1]YSOn (Yleinen suomalainen ontologia) [59] mukaan kissa kuuluu eri petoeläinten heimoon koiran kanssa, josta voidaan vetää se johtopäätös, että ne ovat toistensa poissulkevia, vaikkakaan tätä ei ole YSOn datassa eksplisiittisesti ilmaistukaan.

sa. Tällaisessa tilanteessa data ei ole semanttisesti oikeellista huolimatta sen syntaktisesta oikeellisuudesta[2].

Tämä luku rakentuu seuraavasti: aliluvussa 3.1 käsitellään datan yleinen validointiprosessi. Rajoitteet, jotka ovat keskeinen osa kaikkien validaattoreiden toimintaa, kuvataan aliluvussa 3.2. Linkitetyn datan validoinnissa käytetyt rajoitekielet ja tämän työn kannalta tärkeimmät validaattorit esitetään aliluvussa 3.3.

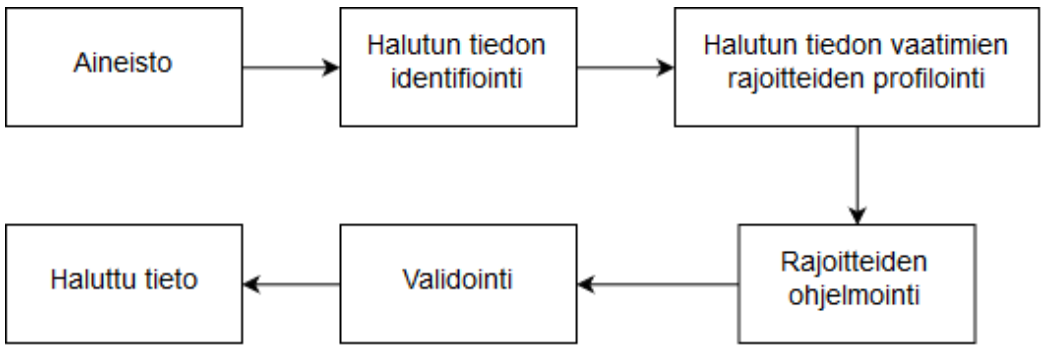
3.1 Validointiprosessi

Datan kelpoisuuden tarkastaminen jonkin sen sisältämän ominaisuuden suhteen on monimutkainen ja -vaiheinen prosessi, joka vaatii käyttäjältä sekä ajatustyötä että tietoa käytettävissä olevasta aineistosta. Datan validointi voidaan yleisesti jakaa seuraaviin kuvassa 3.1 esitettyihin prosesseihin:

1. Aineisto (sen hankkiminen tai luominen)
2. Käyttäjän haluaman tiedon identifiointi
3. Halutun tiedon saamiseen tähtäävä rajoitteiden profilointi
4. Profiloinnissa havaittujen tarpeiden (rajoitteiden) ohjelmointi
5. Ohjelmoitujen rajoitteiden ajaminen validaattorilla aineistoa vasten
6. Haluttu tieto (validaattorin tuottama raportti)

Näistä kohdat 1–3 jäävät tyypillisesti käyttäjän vastuulle, sillä lukuun ottamatta kohdassa 1 mainittua aineiston hankkimista tai luomista mikään kuviteltavissa oleva ohjelma – ainakaan ennen täydellistä HMI-rajapintaa (Human-Machine Interaction) – ei voi etukäteen tietää käyttäjän tarpeita, eikä täten pysty avustamaan näissä prosesseissa. Kohta 4 onkin täten tyypillisesti ensimmäinen vaihe, jossa sovellus voi avustaa käyttäjää muodostamaan ohjelman tukemien rajoitteiden muodostamista. Tästä eteenpäin sovelluksessa käytetty validaattori sanelee sen hyväksymän syötteen muodon ja kielen,

[2]Data voi myös olla semanttisesti oikeellista, vaikkei se olisi esitetty syntaktisesti oikein. Tällöin sen semanttinen tietomalli vastaa todellisuutta, mutta esitysmuoto ei ole sen esittämiseen käytetyn kielen kieliopin mukainen.



Kuva 3.1: Datan validointiprosessi.

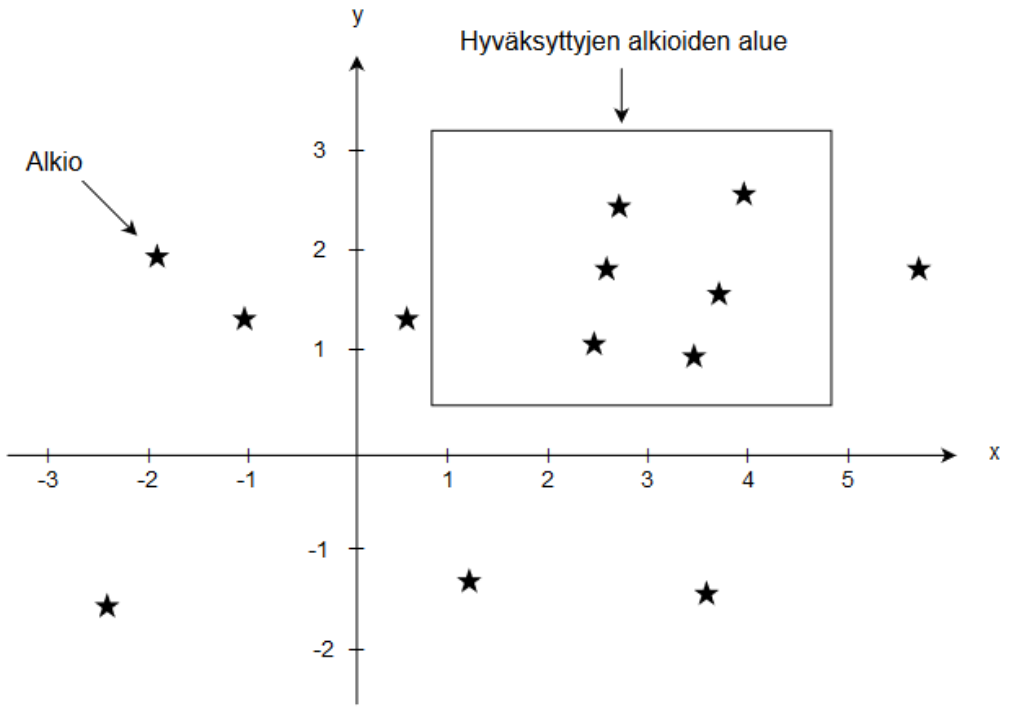
vaikuttaen näin kohdassa 4 luotaviin rajoitteisiin[3]. Lopulta validaattori tuottaa annetun aineiston ja tähän kohdistettujen rajoitteiden validoinnin perusteella validointiraportin, joka sisältää tarkemmat tiedot validoinnin aikana havaituista datan poikkeamista ja virheistä. Käyttäjän tehtäväksi jää tämän tiedon käsitteleminen ja tarpeen vaatiessa siihen reagoiminen.

3.2 Rajoitteet

Rajoitteiden periaatteellisena tehtävänä on suodattaa dataa (yleensä jonkin fasetin suhteen) ja mahdollistaa näin käyttäjää kiinnostavan tiedon helpompi hahmottaminen. Käyttäjää kiinnostava tieto voidaan esittää joko inklusiivisella eli ehdon täyttävällä (koeaineisto sisältää jonkin asian) tai eksklusiivisella eli poissulkevalla (koeaineisto ei sisällä jotakin asiaa) tavalla. Näiden kahden tavan yhdistäminen on mahdollista käytetyn rajoitekielen sen salliessa ja varsin hyödyllistä, sillä jotkin rajoitteet on helpompi esittää tietokoneymmärrettävässä muodossa inklusiivisella ja toiset taas eksklusiivisella tavalla. Dataa koskevat rajoitteiden rajaamat säännöt voivat olla myöskin hyvin erilaisia; jotkin rajoitteet voivat käyttää eksistenssikvanttoreita, toiset taas universaalikvanttoreita ja kolmannet molempia näistä yhdessä monimutkaisempia rajoitteita esitettäessä, aivan kuten predikaattilogiikassa.

Kuvassa 3.2 on havainnollistettu rajoitteen yleinen toimintaperiaate. Joukko alkioita (dataa) on suorakulmion muotoisen hyväksyttyjen alkioiden alueella ja osa niistä sen ulkopuolella. Jos käyttäjä haluaa kuvassa näkyvän suora-

[3]Tämä sanelu vaikuttaakin tähän neljänteen vaiheeseen siten, että se voidaan tarpeen vaatiessa jakaa alivaiheisiin 4a) Rajoitteiden ohjelmointi ja 4b) Ohjelmoitujen rajoitteiden verifiointi eli oikeaksi todentaminen.



Kuva 3.2: Rajoitteen toimintaperiaate: vain osa alkioista osuu hyväksytylle (rajoitetulle) alueelle.

kulmion sisällä olevan osan alkioista, tulee hänen ohjelmoida rajoite, joka hyväksyy vain nämä hyväksytyllä alueella sijaitsevat alkiot. Hypoteettinen rajoite[4], jolla tämä voisi hoitua, voisi olla kuvan 3.2 tapauksessa esimerkiksi seuraavanlainen:

$$\begin{aligned} 2 \leq x \leq 4 \\ -1 \leq y \leq 3 \end{aligned} \tag{3.1}$$

Tässä x on x-akselin mukainen koordinaatti ja y y-akselin.

Ajamalla kaavan 3.1 mukaisesti ohjelmoitu rajoite kuvitteellisella validaattorilla voidaan saada esimerkiksi taulukon 3.1 mukainen raportti. Kuten taulukosta nähdään, on joukossa sekä hyväksyttyjä että hylättyjä alkioita. Näistä hylätyt tulokset indikoivat, että kyseinen alkio ei läpäissyt validaattorin sisältämää testiä. Yksikin hylkäys riittää muuttamaan raportin lopputuloksen

[4]On huomioitavaa, että kaavassa 3.1 esitetyt rajoitteiden määritykset eivät vastaa kuvan 3.2 hyväksyttyjen alkioiden alaa, mutta rajaavat yhtä kaikki samat hyväksytyt alkiot kuin alkuperäinen alue.

Alkion koordinaatti (x, y)	**Tulos**
-2.5, -1.5	Hylätty
-2, 2	Hylätty
-1.125, 1.375	Hylätty
0.5, 1.375	Hylätty
1.125, -1.125	Hylätty
2.375, 1.125	Hyväksytty
2.5, 1.875	Hyväksytty
2.625, 2.5	Hyväksytty
3.375, 1	Hyväksytty
3.5, -1.375	Hylätty
3.625, 1.625	Hyväksytty
3.875, 2.625	Hyväksytty
5.625, 1.875	Hylätty

Taulukko 3.1: Kuvan 3.2 alkioiden validoinnin tulokset kaavan 3.1 mukaan.

hylätyksi, sillä tällöin data ei kokonaisuudessaan ollut rajoitteiden mukaista eli validia.

Rajoitteita voidaan soveltaa myös verkkomuotoiseen dataan, jolloin niiden tulee kohdistua verkon osiin tai niiden välisiin kaariin. Esimerkiksi aliluvussa 2.1 esitetty linkitetty data on kolmikoista koostuva verkko[5], jota voidaan käydä läpi solmu solmulta tietynlaisia osaverkkoja etsien. Listauksessa 3.1 on esitetty RDF-graafi, johon voidaan kohdistaa seuraavan kaavan mukainen rajoite:

$$(\forall x)(P_1 x \wedge P_2 xy) \tag{3.2}$$

Tässä $P_1 x$ on predikaatti ”x on tyyppiä *skos:Concept*[6]” ja $P_2 xy$ on predikaatti ”x:lle on olemassa *skos:prefLabel*-predikaatti[7] kielellä y”. Kaava siis varmistaa, että jokaisella luokan skos:Concept ilmentymällä on suositeltava nimike jollakin tietyllä kielellä. Ajamalla tällainen rajoite listauksen 3.1 graafille arvolla y = ’fi’ saadaan taulukon 3.2 mukainen totuustaulu, jossa epätoden arvon (E) saaneet subjektit eivät läpäise tarkastusta. Tarkemmin tuloksia tulkittaessa havaitaankin, että <väärin_1>-resurssi ei sisällä skos:prefLabel-predikaattia, vaan samannäköiseltä näyttävän skos:prefLabeI-predikaatin (huomaa iso i-kirjain) ja <väärin_2>-subjektiin kiinnitetty skos:prefLabel

[5]Tässä puhutaan yksinkertaisuuden vuoksi verkosta, vaikka annettu data voi aivan yhtä hyvin muodostua useista eri verkoista ja täten nelikoista.

[6]`http://www.w3.org/2004/02/skos/core#Concept`

[7]`http://www.w3.org/2004/02/skos/core#prefLabel`

```
1  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3  @prefix skos: <http://www.w3.org/2004/02/skos/core#> .
4
5  <oikein> rdf:type rdfs:Class, skos:Concept ;
6           skos:prefLabel "Oikein on oikein"@fi .
7
8  <väärin_1> rdf:type rdfs:Class, skos:Concept ;
9           skos:prefLabeI "Väärin on väärin"@fi .
10
11 <väärin_2> rdf:type rdfs:Class, skos:Concept ;
12          skos:prefLabel "Tarkkuutta kielikoodin kanssa!"@en .
```

Listaus 3.1: RDF-datagraafi kaavan 3.2 rajoitteen testaamiseksi.

Subjekti (x)	**Totuusarvo**
<oikein>	T
<väärin_1>	E
<väärin_2>	E

Taulukko 3.2: Listauksen 3.1 RDF-graafin totuustaulu kaavan 3.2 mukaan, kun y='fi'.

on vahingossa määritelty englanninkieliseksi (@en) suomenkielisestä kuvauksestaan huolimatta.

Edellinen esimerkki kohdistui subjekteihin, mutta aivan yhtä hyvin predikaatteja ja objektejakin voidaan validoida. Otetaan toinen esimerkki käyttäen kaavaa

$$(\exists x)(Ox \wedge Lx) \tag{3.3}$$

jossa predikaatti Ox on "x on objekti" ja predikaatti Lx "x on literaali". Tällainen eksistentiaalinen rajoite tuottaa vain vastauksen kyllä/ei (listauksen 3.1 tapauksessa kyllä) riippuen siitä, onko yksikin graafin objekti literaali. Huolimatta rajoitteen korkeasta granulariteetista (karkeusasteesta), tällainenkin rajoite voi tuottaa arvokasta tietoa yksikkötestausten tapaan.

3.3 Linkitetyn datan validointi

Kuten Labra Gayo ja kumppanit kirjassaan [68] esittävät, linkitetyn datan validointiprosessissa tarkastellaan sitä, että noudattaako datamassa jotakin sille ajatellun skeeman tietomallia. Erityisesti RDF-tietomallin ollessa ky-

seessä kysymys redusoituu hyvin matalan tason operaatioihin: toteuttavatko nämä graafin (ei siis ontologian) ominaisuudet tämän korkeamman tason skeeman? Toisaalta he myös toteavat, että linkitetyn datan datajoukkojen kuvauksista ja validoinnista kiinnostuneiden datainsinöörien mielenkiinto kohdistuu erilaisiin asioihin kuin ontologiakehittäjien, sillä datainsinöörien tavoitteena on mallintaa data mahdollisimman tehokkaasti käytettäväksi, kun taas ontologiakehittäjät pohtivat asiaa ensisijaisesti ontologiamallin esitysmuodon kautta. [68]

Labra Gayo ja kumppanit ovat lisäksi kirjassaan [68] luetelleet muun muassa seuraavat linkitetyn datan validointiin lisähaasteita aiheuttavat seikat:

- Datailmentymien tyypitysten tarkastelu ei ole riittävää, sillä RDF-solmuja ei välttämättä ole merkitty (annotoitu) täysin toisistaan eroavilla (ja mahdollisesti poissulkevilla) tyypeillä.
- RDF:n joustavuus: joidenkin predikaattien[8] maalijoukko voi koostua sekä literaaleista että resursseista.
- Samaa ominaisuutta voidaan joissain tapauksissa käyttää eri tarkoituksiin samassa datajoukossa[9].

Monessa tapauksessa lisähaasteisiin johtanut syy on se, että linkitetyn datan sisäänrakennettu skeemattomuus aiheuttaa tämänkaltaisia ”epäloogisuuksia”. Nämä asiat huomioon ottaen onkin pidettävä mielessä, että linkitetyn datan validointi on aina tehtävä käyttökonteksti huomioiden, sillä kelpo data jossain kontekstissa ei välttämättä ole kelpoa dataa toisessa.

Validointiprosessia mutkistaa myös linkitetyn datan yhteydessä käytetty niin kutsuttu *avoimen maailman oletus* (engl. open world assumption, OWA), jonka takia datagraafin kaikkia väitteitä ei välttämättä ole tiedossa sen ajohetkellä. Yleensä datan validointivaiheessa tehdäänkin tästä syystä *suljetun maailman oletus* (engl. closed world assumption, CWA), jossa huomioon otetaan vain validaattorin välittömästi saatavilla olevat kolmikot[10]. Tällöin rajoitteita tutkitaan annetun datan suhteen tiukasti, sillä esimerkiksi listauksen 3.1 RDF-graafi on täydennettävissä kaavan 3.2 rajoitteen toteuttavilla,

[8]Labra Gayo ja kumppanit käyttävät esimerkkinään <http://schema.org/creator>-predikaattia.

[9]Labra Gayo ja kumppanit mainitsevat kirjassaan erimuotoiset arvojoukot sallivan <http://schema.org/productID>-ominaisuuden, jolla voidaan esimerkiksi kuvata saman kirjaresurssin ISBN-10- (International Standard Book Number) ja ISBN-13-tunnukset.

[10]Tämä tarkoittaa mahdollisesti myös implisiittisesti saatavilla olevien kolmikoiden laskemista.

esimerkiksi listauksessa 3.2 esitetyillä, toisessa syötetiedostossa tai Internet-verkossa sijaitsevilla graafin ulkopuolisilla kolmikoilla, mutta ne eivät ole validoinnin aikana käytettävissä. Koska avoimessa maailmassa sijaitsevien mahdollisten kolmikoiden selvittäminen on vähintäänkin hidasta, käytetään suljetun maailman oletusta useissa eri validaattoritoteutuksissa tehostamaan ja suoraviivaistamaan muutenkin monimutkaista linkitetyn datan validointia.

```
1 @prefix skos: <http://www.w3.org/2004/02/skos/core#> .
2
3 <väärin_1> skos:prefLabel "Oikein"@fi .
4 <väärin_2> skos:prefLabel "Suomeksi, kiitos."@fi .
```

Listaus 3.2: Listauksesta 3.1 puuttuvat kolmikot, jotta taulukon 3.2 kaikki subjektit palauttaisivat totuusarvon T validoitaessa.

Tämän aliluvun loppu on kaksijakoinen: aliluvussa 3.3.1 esitetään ensin linkitetyn datan rajoitteiden tunnetuimmat esityskielet esimerkkeineen, jota seuraa näiden rajoitekielten ominaisuuksien yhteenveto ja vertailu työn tutkimuskysymyksiin ja tavoitteisiin peilaten. Aliluvussa 3.3.2 esitellään tämän työn kannalta oleelliset, viimeisintä tekniikkaa edustavat niin kutsututut *state-of-the-art*-validaattoritoteutukset.

3.3.1 Rajoitekielet

Rajoitekielet toteuttavat jonkin konkreettisen tavan ilmaista linkitetyn datan validoinnissa tarvittavia rajoitteita. Monesti rajoitteetkin ilmaistaan linkitetyn datan alalla linkitettynä datana, vaikka tarkkaan ottaen tämä ei ole pakollista, kuten aliluvussa 3.2 nähtiin (kaavojen 3.2 ja 3.3 käsittely). Tällä on kuitenkin useita etuja, kuten Bosch ja Eckert artikkelissaan [20] luetteloivat:

- Rajoitteet voidaan yhdenmukaisesti tallentaa ontologioiden ja RDF-datan joukkoon.
- Rajoitteita voidaan helposti jakaa dataverkoissa.
- Rajoitevalidointi voidaan suorittaa automaattisesti.
- Rajoitteita voidaan prosessoida jo olemassa olevilla RDF-työkaluilla.
- Rajoitteet linkittyvät suoraan RDF-dataan.

Koska RDF-perustaisilla rajoitekielillä on näin merkittäviä etuja, esitellään seuraavaksi vain tarkkaan määritellyt ja hyvin tunnetut RDF-perustaiset ra-

joitekielet sekä rajoitekieleksi soveltuva RDF-datan kysely- ja tiedonhallintakieli SPARQL, joista soveltuvin on valittu luvussa 5 kuvatun, työssä tehdyn ohjelmallisen toteutuksen perustaksi. SPIN-sääntö- ja rajoitekieli (SPARQL Inferencing Notation) [65] on tarkoituksella rajattu tämän esittelyn ulkopuolelle, sillä se on nyttemmin käytännössä korvattu W3C:n virallisella SHACL-suosituksella, jota SPIN-kielen pääkehittäjä Holger Knublauchin mukaan voidaan pitää SPINin aitona seuraajana [62]. Tästä huolimatta SPIN-kieltä tuetaan yhä, ja se – aiemmasta de-facto-teollisuusstandardiasemastaan johtuen [61] – on edelleen laajalti käytössä tuotanto-ohjelmissa ja täten tässä erityismaininnan arvoinen [62].

RDFS

RDFS-sanasto [22] määrittelee kaksi keskeistä ominaisuutta, joiden avulla voidaan esittää RDF-datan luokka- ja ominaisuushierarkioita (*rdfs:subClassOf*[11] ja *rdfs:subPropertyOf*[12], vastaavasti). Kuten aliluvussa 2.1 esitettiin, näiden lisäksi RDFS-sanasto määrittelee tavan esitellä resurssien välisille ominaisuuksille rajoitteita. Määrittelyjoukkorajoitteet (*rdfs:domain*[13]) rajoittavat subjektin arvojoukkoa, kun taas maalijoukkorajoitteet (*rdfs:range*[14]) rajoittavat objektin arvojoukkoa.

Listauksessa 3.3 on esitetty yksinkertainen RDF-graafi, jonka validointi UNA (Unique Name Assumption)/CWA-oletusyhdistelmällä[15] eli yksilöllisen nimen[16] ja suljetun maailman oletuksilla antaa virheen, sillä ominaisuuden <naimisissa> yläominaisuudeltaan perimä maalijoukkorajoite ei toteudu datassa. Sitä vastoin avoimen maailman oletuksella[17] tämä on validi RDF-graafi, sillä jossain voi olla kolmikko, joka selittää esimerkiksi yksilön <koira_1> olevan tyypiltään myös luokkaa <Ihminen>. On huomioitavaa, että rivillä 5 olevasta kolmikosta seuraa se, että jokainen luokan <Nainen> yksilö on myös luokan <Ihminen> yksilö. Tästä seuraa rivillä 15 esitetyn määrittelyjoukkorajoitteen toteutuminen rivin 19 lauseelle.

[11] `http://www.w3.org/2000/01/rdf-schema#subClassOf`
[12] `http://www.w3.org/2000/01/rdf-schema#subPropertyOf`
[13] `http://www.w3.org/2000/01/rdf-schema#domain`
[14] `http://www.w3.org/2000/01/rdf-schema#range`
[15] RDFS:ssä ei tavallisesti tehdä tätä oletusyhdistelmää.
[16] Tällöin eri nimien oletetaan kuvaavan eri entiteettejä.
[17] Avoimen maailman oletuksen kanssa UNA-oletusta ei yleensä käytetä.

```
1  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3
4  <Ihminen> rdf:type rdfs:Class .
5  <Nainen> rdfs:subClassOf <Ihminen> .
6  <Eläin> rdf:type rdfs:Class .
7  <Koira> rdfs:subClassOf <Eläin> ;
8          rdfs:label "koira"@fi ;
9          rdfs:label "hauva"@fi .
10
11 <nainen_1> rdf:type <Nainen> .
12 <koira_1> rdf:type <Koira> .
13
14 <ihmissuhde> rdf:type rdf:Property ;
15               rdfs:domain <Ihminen> ;
16               rdfs:range <Ihminen> .
17 <naimisissa> rdfs:subPropertyOf <ihmissuhde> .
18
19 <nainen_1> <naimisissa> <koira_1> .
```

Listaus 3.3: <naimisissa>-ominaisuuden määrittely- ja maalijoukkorajoite.

OWL 2

Web Ontology Language (OWL) versionumero 2 [81] on W3C:n OWL-työryhmän[18] kehittämä ontologiakieli, joka laajentaa sen ensimmäistä versiota[19] [77]. Kaikki OWL 1 -yhteensopivat ontologiat ovatkin valideja OWL 2 -ontologioita. [107] OWL mahdollistaa RDFS-sanastoon verrattuna esimerkiksi aliluvussa 3.2 esitettyjen kaavojen mukaisten rajoitteiden esittämisen ja validoinnin, sillä se on kykenevä ensimmäisen asteen predikaattilogiikalla kuvattujen rajoitusten täysimääräiseen ilmaisemiseen. On kuitenkin huomioitavaa, että OWL-kielellä esitettävissä olevien rajoitusten joukko ei ole välttämättä loogisesti ratkeava tai laskennallisesti tehokkaasti ratkaistavissa, minkä takia validoinnissa käytetäänkin usein OWL-terminologian mukaisesti niin kutsuttuja *profiileja*. Nämä profiilit ovat alijoukkoja OWL Full -kielestä, joka sisältää kaikki OWL-ontologiakielellä esitettävissä olevat rajoitteet ja ominaisuudet. Validoinnin kannalta laskennallisesti laajin, tehokkaasti ratkeava OWL 2 -alijoukko on OWL 2 DL (Description Logic) [81], joka vastaa $\mathcal{SROIQ}^{(D)}$-tason [55] kuvailulogiikkakieltä. On myös huomioitavaa, että OWL 2 DL:n mukaisten RDF-graafien joukko on suurempi kuin OWL 1 DL -graafien. Tämä on suoraa seurausta siitä, että OWL 2:n taustalla olevan

[18] `https://www.w3.org/2007/OWL/wiki/OWL_Working_Group`

[19] Tässä työssä ensimmäiseen versioon viitataan tästedes termillä OWL 1.

logiikkamallin rajoitteita on uudistuksessa lievennetty OWL 1:n vastaaviin nähden[20] [107].

Listauksen 3.4 riveillä 4–5 on esitetty listauksen 3.3 <ihmissuhde>-ominaisuuden maalijoukkovirheen havaitseva *owl:AllDisjointClasses*-rajoite[21], joka määrittelee luokista <Ihminen> ja <Eläin> muodostetun leikkauksen olevan tyhjä joukko[22]. On merkille pantavaa, että datan validoimiseksi OWL ei aksioomaperustaisuutensa vuoksi kaikissa tapauksissa vaadi suljetun maailman oletusta RDFS:n tapaan. Tämä on seurausta siitä, että OWL-aksioomien kanssa loogisessa ristiriidassa olevat RDF-lauseet tulkitaan virheinä datassa.

```
1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2 @prefix owl: <http://www.w3.org/2002/07/owl#> .
3
4 [] rdf:type owl:AllDisjointClasses ;
5    owl:members ( <Ihminen> <Eläin> ) .
```

Listaus 3.4: Listauksen 3.3 <Ihminen>- ja <Eläin>-luokkien yhteisten yksilöiden poissulkevuusrajoite OWL-ontologiakielellä esitettynä.

SPARQL

SPARQL-kyselykieli [50] poikkeaa RDFS:stä ja OWL:sta siinä, että sen pääasiallinen käyttötarkoitus – kuten kyselykielillä yleensäkin – on palauttaa vastauksia tietokantaan kohdistuviin mielivaltaisiin[23] kyselyihin. Datan validoinnin kannalta olennaisimmat SPARQL-kyselytyypit ovat ASK, SELECT ja CONSTRUCT, joista ensimmäinen palauttaa totuusarvon kyselyyn, mahdollistaen näin suoraan rajoitteen täyttymisen kysymisen ilman tarkempaa tietoa siitä, mikä kolmikko tai kolmikkoryhmä aiheutti rikkeen. SELECT-kysely palauttaa kyselyssä palautettaviksi määritellyt muuttujat arvoineen kaikille vastauksille eli validoinnin kannalta mahdollistaa siis kaikkien rikkeiden luetteloinnin arvoineen. CONSTRUCT-kyselytyyppi puolestaan mahdollistaa uusien kolmikkojen palauttamisen ja täten esimerkiksi rakenteisen palauteraportin tuottamisen.

[20] Koska lievennykset kasvattavat käytettävän logiikkamallin ilmaisuvoimaa, perustuu OWL 1 DL hieman heikompaan, $\mathcal{SHOIN}^{(D)}$-tason [53] kuvailulogiikkakieleen.

[21] `http://www.w3.org/2002/07/owl#AllDisjointClasses`

[22] Tarkalleen ottaen rajoite tarkastaa yksilöiden osalta sen, että onko yksilö määritelty olemaan samaan aikaan sekä luokan <Ihminen> että luokan <Eläin> ilmentymä. Koska listauksessa 3.3 <koira_1> on luokan <Koira> yksilö (joka on luokan <Eläin> alaluokka), on ominaisuuden <naimisissa> objektille määritelty vaade väistämättä epätosi.

[23] Mielivaltaisella tarkoitetaan tässä ”Millä tahansa kielen syntaksin mukaisella”.

Listauksessa 3.5 on esitetty listauksen 3.3 <naimisissa>-ominaisuuden määrittely- ja maalijoukkorajoitteiden validointi ASK-kyselyllä. Paluuarvo *totta* tarkoittaa sitä, että data ei ole näiden rajoitteiden mukaista. On huomioitavaa, että tämä kysely olisi helposti laajennettavissa käsittämään kaikkien datassa esiintyvien predikaattien määrittely- ja maalijoukkorajoitteiden validoinnin vain vaihtamalla yksinkertaisesti <naimisissa>-ominaisuuden tilalle esimerkiksi muuttujan ?p.

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3
4  ASK {
5   <naimisissa> rdfs:subPropertyOf*/rdfs:domain ?domain .
6   <naimisissa> rdfs:subPropertyOf*/rdfs:range ?range .
7   ?s <naimisissa> ?o .
8   OPTIONAL { ?s rdf:type ?s_class }
9   OPTIONAL { ?o rdf:type ?o_class }
10
11  FILTER (NOT EXISTS { ?s rdf:type ?s_class } ||
12          NOT EXISTS { ?o rdf:type ?o_class } ||
13          NOT EXISTS { ?s_class rdfs:subClassOf* ?domain } ||
14          NOT EXISTS { ?o_class rdfs:subClassOf* ?range })
15 }
```

Listaus 3.5: Listauksen 3.3 <naimisissa>-ominaisuuden määrittely- ja maalijoukkorajoitteiden validointi SPARQL-kyselyllä.

SHACL

SHACL [63] on W3C:n suositus RDF-rajoitekieleksi. SHACL-terminologiassa rajoitteet esitetään niin kutsuttuina *muotoina* (engl. shape), joiden muodostama muotograafi määrittelee datagraafin *kohdesolmu(i)lta*[24,25] (engl. focus node) kelvollistamisessa vaadittavat ominaisuudet mahdollisine arvojoukkorajoitteineen. Jokaiselle muodolle on lisäksi mahdollista asettaa sen rikkomuksesta aiheutuvan *virheen*[26] vakavuusaste sekä ihmisluettava selite virheen syystä (*sh:severity*[27] ja *sh:message*[28], vastaavasti). Muotojen validoin-

[24]Yksi muoto voi kohdistua useisiin kohdesolmuihin.

[25]Kohdesolmu voi olla mikä tahansa datagraafin RDF-termi.

[26]Tässä työssä käytetään yleiskäyttöisestä termistä *raportoitava asia* keskenään vaihtokelpoisesti myös termiä virhe, vaikka raportoitavan asian aiheuttaneen rikkomuksen vakavuusaste voi olla aidon virheen sijaan esimerkiksi ”pelkkä” varoitus tai informaatioarvo.

[27]`http://www.w3.org/ns/shacl#severity`

[28]`http://www.w3.org/ns/shacl#message`

nin tuloksena syntyvä validointiraportti sisältää tiedon datan validoinnin onnistumisesta ja tätä tukeakseen myös validaattoritoteutuksen datagraafin validoinnin aikana havaitsemat, raportoitaviksi määritellyt asiat[29,30]. On huomioitavaa, että validoinnin katsotaan onnistuneen vain sellaisessa tapauksessa, jossa validointimoottori ei validoinnin seurauksena tuota yhtäkään raportoitavaa asiaa (riippumatta näiden vakavuusasteista), eikä se ilmoita tai ole keskeytynyt häiriöön[31]. Kuten SHACL-spesifikaatiossa vielä lisäksi todetaan, datagraafien kelvollisuuksista päättävien muotograafien voidaan ajatella tuottavan täsmäkuvauksia kaikista kelvollisista datagraafeista. Tällaisten kuvausten etuihin lukeutuu mahdollisuus hyödyntää niitä tavanomaisen validoinnin lisäksi esimerkiksi käyttöliittymien rakentamisissa, ohjelmakoodin automaattisessa tuottamisessa ja datajoukkojen integraatioissa. [63]

Listauksessa 3.6 on esitetty yksi mahdollinen tapa rajoittaa listauksessa 3.3 predikaattina käytetyn <naimisissa>-ominaisuuden subjekti ja objekti <Ihminen>-luokan yksilöihin SHACL-kielellä. Siinä riveillä 20–21 olevat lauseet määrittävät rajoitteessa <naimisissa_ihminen_rajoite> tarkasteltavat kohdesolmut datassa käytettyjen <naimisissa>-ominaisuuksien objekteiksi ja subjekteiksi, vastaavasti. Näiden kohdesolmujen arvojoukko voidaan SHACL-rajoitekielessä lisäksi rajata pelkkiin IRI-tunnisteisiin (rivi 22), mikä poissulkee sekä literaalit että nimettömät solmut[32]. Rivillä 23 sijaitseva rajoitekomponentti pakottaa rajoitteen kohdesolmujen vähintään yhdeksi tyypiksi <Ihminen>-luokan tai tämän alaluokan. Riveillä 24–33 päätetään rajoite määrittelemällä SPARQL-rajoitekomponentti[33], jonka käyttämät etuliitteet on määritelty entiteetissä <shacl_sparql_etuliitteet> (rivit 6–16). Rajoitekomponentti täydentää aiempia rajoituksia lisäämällä sellaisen poissulkevuusrajoitteen, joka estää hyväksymästä yksilöitä, jotka on tyypitetty sekä <Ihminen>-luokan (tai tämän alaluokan) ilmentymäksi että muuksi kuin <Ihminen>-luokan yläluokiksi.

Listauksessa 3.7 on esitetty kaksi virhettä sisältävä validointiraportti. Nämä virheet paljastuivat, kun listauksen 3.6 muotograafia ajettiin listauksen 3.3 datagraafia vastaan. Ensimmäinen virhe (rivit 7–15) on syntynyt siitä, että <naimisissa_ihminen_rajoite>-muotorajoitteen luokkarajoitekomponentti ei ole toteutunut. Tarkemmin asiaa tutkiessa havaitaankin, että yk-

[29]Nämä asiat ovat edellä mainittuja virheitä.

[30]Vähimmäisvaatimukset raportoitavalle asialle ovat tieto virheen sisältävästä kohdesolmusta, virheen vakavuusaste sekä rikottu rajoitekomponentti.

[31]Häiriö voi johtua esimerkiksi validoinnin aikana havaitusta rekursiivisesta muodosta tai laskentakoneen resurssien loppumisesta.

[32]SHACL-rajoitekielessä on toki olemassa rajoittava arvo kaikille kohdesolmujen tyyppiyhdistelmille.

[33]SHACL-rajoitekielessä on mahdollista hyödyntää SPARQL-rajoitekielisiä rajoitteita.

```
1  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3  @prefix sh: <http://www.w3.org/ns/shacl#> .
4  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5
6  <shacl_sparql_etuliitteet>
7    sh:declare [
8      sh:prefix "rdf" ;
9      sh:namespace
10   "http://www.w3.org/1999/02/22-rdf-syntax-ns#"^^xsd:anyURI ;
11   ] ;
12   sh:declare [
13     sh:prefix "rdfs" ;
14     sh:namespace
15   "http://www.w3.org/2000/01/rdf-schema#"^^xsd:anyURI ;
16   ] .
17
18 <naimisissa_ihminen_rajoite>
19   rdf:type sh:NodeShape ;
20   sh:targetObjectsOf <naimisissa> ;
21   sh:targetSubjectsOf <naimisissa> ;
22   sh:nodeKind sh:IRI ;
23   sh:class <Ihminen> ;
24   sh:sparql [
25     sh:prefixes <shacl_sparql_etuliitteet> ;
26     sh:select """SELECT $this
27 WHERE {
28   $this rdf:type ?type .
29   <Ihminen> rdfs:subClassOf* ?accepted_types .
30   FILTER (NOT EXISTS { ?type rdfs:subClassOf* <Ihminen> } &&
31       NOT EXISTS { ?accepted_types rdfs:subClassOf* ?type })
32 }"""
33   ] .
```

Listaus 3.6: Listauksen 3.3 <naimisissa>-ominaisuuden SHACL-rajoitekielellä esitetty määrittely- ja maalijoukkorajoite vahvistettuna muiden kuin <Ihminen>-tyypin SPARQL-poissulkevuusrajoitekomponentilla.

```
1  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
2  @prefix sh: <http://www.w3.org/ns/shacl#>.
3
4  [ rdf:type sh:ValidationReport ;
5    sh:conforms false ;
6    sh:result
7   [
8    rdf:type sh:ValidationResult ;
9    sh:resultSeverity sh:Violation ;
10   sh:value <koira_1> ;
11   sh:resultMessage "Value does not have class Ihminen" ;
12   sh:focusNode <koira_1> ;
13   sh:sourceShape <naimisissa_ihminen_rajoite> ;
14   sh:sourceConstraintComponent sh:ClassConstraintComponent
15  ] ;
16   sh:result
17  [
18   rdf:type sh:ValidationResult ;
19   sh:resultSeverity sh:Violation ;
20   sh:value <koira_1> ;
21   sh:sourceConstraint [] ;
22   sh:focusNode <koira_1> ;
23   sh:sourceShape <naimisissa_ihminen_rajoite> ;
24   sh:sourceConstraintComponent sh:SPARQLConstraintComponent
25  ]
26 ] .
```

Listaus 3.7: Listauksen 3.3 datagraafin validointiraportti, kun SHACL-muotorajoitteina on käytetty listauksen 3.6 muotograafia.

silö <koira_1> on tyyppiä <Koira>, joka on <Eläin>-luokan alaluokka, eikä näillä ole yläluokkayhteyttä <Ihminen>-luokkaan[34]. Toinen virhe (rivit 17–25) kohdistuu myös yksilöön <koira_1>, tällä kertaa <naimisissa_ihminen_rajoite>-muodon SPARQL-rajoitekomponentin laukaisemana[35]. Tätä virhettä analysoitaessa havaitaankin näiden molempien virheiden johtuvan samasta syystä ja rajoitekomponenttien olevan osittain päällekkäiset. Virheistä johtuen validointiraportin datagraafin rajoitteiden mukaisuuden ilmaisevan *sh:conforms*-predikaatin[36] (rivi 5) arvona on epätosi, eikä datagraafi täten noudata annettua muotograafia.

ShEx 2.0

ShEx versionumero 2.0 [88] on tätä kirjoitettaessa[37] uusin ShEx Community Group -yhteisöryhmän[38] jatkokehittämä versio alkuperäisestä ShEx-spesifikaatiosta [95], johon on lisätty merkittävänä uudistuksena muun muassa syklisiin rakenteisiin ja päättymättömään rekursioon pystyvän validoinnin määrittely[39]. ShEx 1.0 kuitenkin aikoinaan hävisi kilpailun SHACL-rajoitekielelle W3C:n viralliseksi RDF-rajoitekielisuositukseksi tätä varten perustetussa W3C:n RDF Data Shapes -työryhmässä[40], mutta on siitä huolimatta säilyttänyt merkityksensä erinäisissä sovelluksissa erilaisen validointifilosofiansa (skeemaperustaisuus verrattuna SHACL-kielen SPARQL-kyselykieleen vahva tukeutuminen) tuomien etujen ansiosta. Tästä viimeisimpänä esimerkkinä [71] Wikidata-palveluun[41] tuotettu integraatio.

ShEx- – kuten SHACL-rajoitekielessäkin – terminologian keskiössä ovat muodot ja solmurajoitteet, joiden loogisilla yhdistelmillä tässä tapauksessa muodostetaan erilaisia *muotolausekkeita* (engl. shape expression). Näiden kokoelmat taasen muodostavat käytettävän skeeman, jota voidaan puolestaan testata ShEx-mallissa haluttua RDF-datagraafia vastaan validointituloksia tuottaen. Tätä ei kuitenkaan voida tehdä ilman niin kutsuttua *muotokart-*

[34] Vaikka tämä olisi datan puolesta mahdollista toteuttaa, ei se kuitenkaan olisi erityisen mielekästä, sillä se ei kuvaisi reaalitodellisuutta kovinkaan hyvin.

[35] Rivin 21 tietoarvo on hieman kryptinen, sillä se viittaa anonyymiin solmuun. Tämä johtuu siitä, että listauksessa 3.6 riviltä 24 alkava *sh:sparql*-predikaatin objekti on nimetön solmu, eikä tällainen tieto voi siirtyä validoinnissa muuntumattomana ja täten informaatioarvonsa säilyttäen ilman aliluvussa 2.1 esitettyä skolemointia.

[36] `http://www.w3.org/ns/shacl#conforms`

[37] 25.6.2019

[38] `https://www.w3.org/community/shex/`

[39] Tässä työssä ensimmäiseen versioon viitataan tästedes termillä ShEx 1.0.

[40] `https://www.w3.org/2014/data-shapes/`

[41] `https://www.wikidata.org`

taa (engl. shape map) [87], joka määrittelee validointiprosessissa käypäisyystarkastuksen kohdesolmut sekä niihin kohdistuvat muotolausekkeet. Käytettävän muotokartan voi muodostaa itse vapaasti luetteloimalla (niin kutsuttu *kiintomuotokartta* [engl. fixed shape map]) tai hyödyntäen tätä varten kehitettyä *kyselymuotokarttaa* (engl. query shape map), jonka järjestelmä sisäisesti ennen validointia muuntaa ja tarvittaessa yhdistää yhtenäiseksi kiintomuotokartaksi. Validoinnin lopputuloksena validointimoottori muodostaa *tulosmuotokartan* (engl. result shape map), joka vastaa kohdesolmujen validoinnin tuloksia[42]. [11] Eriyttämällä muotokartan skeemasta ShEx-muotolausekekieli voi vastata useampiin käyttötarkoituksiin – esimerkkinä toiminee skeeman osalta analoginen XSD-kieli (XML Schema Definition Language) [47, 85], jonka luomisen jälkeen sitä käytettiin WSDL-protokollassa (Web Services Description Language) [25, 26]. [68]

Listauksessa 3.8 on esitetty ShExC-esitysmuodossa[43] (Shape Expressions Compact Syntax) luokan <Ihminen> skeema. Tämä skeema on rakennettu rekursiivisesti: riveillä 6–9 on <Ihminen>-luokan yksilön tunnistava <IhminenMuoto>-muotolauseke, joka rivillä 8 viittaa toiseen, riveillä 11–14 määriteltyyn <AliIhminenMuoto>-muotolausekkeeseen. <AliIhminenMuoto> vastaa <Ihminen>-tyypin alaluokkien kattamisesta; rivillä 12 päätetään etsintä, jos yläluokkaa merkitsevän rdfs:subClassOf-suhteen arvona on <Ihminen>, muuten tätä samaa muotolauseketta kutsutaan rekursiivisesti tämän yläluokalle (rivi 13). Määritelty skeema vastaa SPARQL-kyselykielen *ominaisuuspolkua*[44] (engl. property path) *?s rdf:type/rdfs:subClassOf* <Ihminen>* ja aiheuttaa virheen, jos asetellulle muuttujalle *?s* ei löydy tällaista polkua. Rivillä 4 on esitetty skeeman luojan oletusarvoinen, aloituskohdan osoittava ohje skeeman käyttäjälle.

Listauksessa 3.9 on esitetty kyselymuotokartta, joka asettaa <naimisissa>-ominaisuuden kaikki subjektit ja objektit (rivit 1 ja 2, vastaavasti) <IhminenMuoto>-muotolausekkeen kohdesolmuiksi. Listauksen 3.3 graafin kohdalla tämä vastaa yksilöitä <nainen_1> ja <koira_1>, ja kuvassa 3.3 on näytetty, miltä niiden tulosmuotokartta ShEx2 – Simple Online Validator -työkalulla[45] listauksen 3.8 skeemaa käyttäen näyttää. Kuten kuvasta on luettavissa, <nainen_1> läpäisee testin, mutta <koira_1> ei. Tarkemmin tulosta tutkimalla havaitaan validaattorin aloittaneen rekursiivisen tutkinnan <Koira>-luokan

[42]Moottori muodostaa kaikista validoiduista (myös kelvollisiksi todetuista) solmuista tuloksia, helpottaen näin tarkastamisprosessia.

[43]ShExC on ihmisluettavaksi ja -käytettäväksi tarkoitettu tiivis ShEx-esitysmuoto.

[44]Ominaisuuspolku on kahden solmun välinen mahdollinen polku graafissa [50].

[45]`https://rawgit.com/shexSpec/shex.js/master/packages/shex-webapp/doc/shex-simple.html`

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3
4  start = @<IhminenMuoto>
5
6  <IhminenMuoto> {
7     rdf:type [<Ihminen>] |
8     rdf:type @<AliIhminenMuoto>
9  }
10
11 <AliIhminenMuoto> {
12    rdfs:subClassOf [<Ihminen>] |
13    rdfs:subClassOf @<AliIhminenMuoto>
14 }
```

Listaus 3.8: Listauksen 3.3 <Ihminen>-luokan ShEx-muotoinen skeema.

yläluokista, mutta koska <Eläin>-luokalla ei <AliIhminenMuoto>-muotolausekkeen vaatimalla tavalla ole rdfs:subClassOf-suhdetta, on validointityökalu tulkinnut yksilön <koira_1> epävalidiksi suhteessa käytettyyn muotolausekkeeseen.

```
1  { FOCUS <naimisissa> _ }@START,
2  { _ <naimisissa> FOCUS }@<IhminenMuoto>
```

Listaus 3.9: <naimisissa>-ominaisuuden subjektit oletusmuotolausekkeen kohdesolmuiksi sekä saman ominaisuuden objektit <IhminenMuoto>-muotolausekkeen kohdesolmuiksi valitseva kyselymuotokartta.

✓<nainen_1>@START

✗<koira_1>@!<IhminenMuoto>

```
validating koira_1 as IhminenMuoto:
    validating Koira:
      validating Eläin:
          Missing property: http://www.w3.org/2000/01/rdf-schema#subClassOf
          Missing property: http://www.w3.org/2000/01/rdf-schema#subClassOf
```

Kuva 3.3: ShEx2 – Simple Online Validator -työkalun [94] tuottama tulosmuotokartta, kun listauksen 3.3 RDF-graafia validoidaan listauksen 3.8 skeemalla ja listauksen 3.9 kyselymuotokartalla.

Yhteenveto

Kuten edellä esiteltiin, on nykyisin käytössä olevilla RDF-rajoitekielillä (tai niiksi soveltuvilla kielillä) varsin suuria toiminnallisia ja ilmaisullisia eroja. Osa eroista selittyy historialla; esimerkiksi RDFS ja OWL ovat huomattavasti muita vanhempia teknologioita ja kehitetty ratkaisemaan vasta alkutaipaleellaan olleen semanttisen webin haasteita, kun taas SPARQL-kyselykieli on kehitetty RDF-tietokantojen ja -graafien käsittelyyn ja tiedon louhintaan. Semanttisen webin teknologiapinon kypsyessä ja linkitetyn datan määrän kasvaessa alkoi tarve virheitä tehokkaasti paikantaville rajoitekielille kasvamaan. Tähän tarpeeseen kehitettiin tässä aliluvussa aiemmin esitetyt SHACL- ja ShEx-rajoitekielet, joiden ominaisuudet tarkimmin vastaavat tämän työn aiheen kannalta oleellisimpia asioita, kuten taulukossa 3.3 esitetään. Kyseinen taulukko sisältää yhteenvedon eri rajoitekielten tai sellaisiksi soveltuvien kielten tärkeimmistä ominaisuuksista ja eroista[46]. On huomioitavaa, että jotkin tämän yhteenvedon päätelmistä ovat hieman tulkinnanvaraisia – esimerkiksi SPARQL-kyselykieli voi teknisesti kyllä sulkea entiteetin[47], mutta on tärkeää pitää mielessä, että tämä ja esimerkiksi monikertarajoitteiden mielivaltainen rajoittaminen eivät välttämättä skaalaannu kovinkaan hyvin. Labra Gayo ja kumppanit [68] tunnistavatkin kirjassaan ansiokkaasti, että tällaisten vain käypiä rakenteita hyväksyvien kyselyiden tuottaminen muille kuin yksinkertaisimmille tietomalleille on paitsi puuduttava, myös monimutkainen työ. Lisäksi tällaisten kyselyiden kirjoittaminen – puhumattakaan testaamisesta – voi olla vaikeaa aloittelevalle tietokanta-ammattilaiselle. Toisaalta SPARQL-kyselykieltä tuetaan laajasti ja se mahdollistaa erittäin ilmaisuvoimaisten kyselyiden teon, joilla voi vastata tavallisimpiin RDF-validointitarpeisiin. [68] SHACL-kielen SPARQL-laajennos antaakin sille muihin kieliin verrattuna selvän edun tässä suhteessa ja tätä ominaisuutta hyödynnettiin jo onnistuneesti listauksen 3.6 esimerkissä.

Kuten taulukosta 3.3 nähdään, ShEx täyttää SHACL-kielen kanssa suurimman osan keskeisistä rajoitekielten ominaisuuksista[48]. Labra Gayo ja kumppanit toteavatkin kirjassaan [68], että SHACL- ja ShEx-rajoitekielten ilmaisuvoima on riittävä ja samankaltainen yleisimpiin käyttötarkoituksiin. Seuraavassa on esitetty Labra Gayon ja kumppanien [68] tunnistama tiivistetty listaus SHACL- ja ShEx-kielten eroista:

- ShEx on skeemaperustainen, kun taas SHACL-kieli keskittyy RDF-

[46]Tämä yhteenveto ei ole täydellinen eikä sellaiseksi tarkoitettukaan, sillä kaikkien ominaisuuksien luettelointi tässä olisi lukijan kannalta liian vaikeaselkoista.

[47]Hyödyntämällä esimerkiksi COUNT-funktiota.

[48]Ominaisuuspolkujakin voidaan emuloida sisäkkäisillä muodoilla ja rekursiolla [68].

rajoitekieli / ominaisuus	**RDFS**	**OWL 2**	**SPARQL**	**SHACL**	**ShEx 2.0**
W3C-suositus	Kyllä	Kyllä	Kyllä	Kyllä	Ei
RDF-esitysmuoto	Kyllä	Kyllä	Ei	Kyllä	Kyllä
Ihmisluettava- ja käyttöinen syntaksi	Ei	Kyllä	Ei	Valmisteilla	Kyllä
UNA/CWA-oletukset	Ei[a]	Ei[a]	Ei/Ei sovellu	UNA 2.0[b]/Kyllä	Kyllä
Boolen operaattorit	Ei	Kyllä	Kyllä	Kyllä	Kyllä
Monikertarajoitteet	Ei	Kyllä	Kyllä	Kyllä	Kyllä
Ominaisuuspolut	Ei	Ei	Kyllä	Kyllä	Ei[c]
Käänteisominaisuus	Ei	Kyllä	Kyllä	Kyllä	Kyllä
Rekursio	Ei	Ei	Ei	Ei[49]	Kyllä
Päättelyjärjestelmä	Kyllä	Kyllä	Voi toteuttaa	Voi hyödyntää	Ei
Solmutyyppirajoitteet	Ei	Ei	Kyllä	Kyllä	Kyllä
Literaalitarkastelut[d]	Ei	Kyllä/Ei[d]	Kyllä	Kyllä	Kyllä
Laajennokset	Ei	Ei	Ei	SPARQL, JS	Semanttiset toiminnot[e]
Suljetut muodot	Ei	Ei	Ei	Kyllä	Kyllä
Validointiraportti	Ei	Ei	Mahdollinen	Kyllä	Kyllä

Taulukko 3.3: Yhteenveto eri rajoitekielten tai niiksi soveltuvien kielten ominaisuuksista ja eroista[46].

[a] Voidaan soveltaa. [b] UNA 2.0 vaatii erityisen suhteen ilmaisemaan kahden entiteetin välistä yhtäläisyyttä.
[c] Vain kyselymuotokartoissa. [d] VV13 & VV14 / VV15 [e] Kieliriippumaton apuohjelmointiympäristö.

graafien rajoitteiden määrittelyyn.

- ShEx-kielessä on formaali tuki rekursiolle ja syklisille tietomalleille, kun SHACL-kielen osalta se on määrittelemättä[49].
- SHACL tukee mielivaltaisia SPARQL-ominaisuuspolkuja, kun taas ShEx-kielessä on tuki vain saapuville ja lähteville kaarille.
- Kummatkin kielet mahdollistavat yleisen virheraportoinnin muototasolla. SHACL kuitenkin palauttaa edukseen yksinkertaisemmissa muodoissa tarkemman tiedon virheestä ja kontekstista. ShEx helpottaa validoinnin tarkastelua tuottamalla tulosmuotokartan, joka sisältää validoinnin tuloksen jokaisen solmun (myös kelvollisen) osalta. Tämä muotokartta ei kuitenkaan ole SHACL-validointiraportin tapaan RDF-muotoinen ja kiinteän sanaston alainen.
- ShEx-rajoitekielessä on kieliriippumaton laajennusmekanismi [88], kun taas SHACL-kielen ominaisuuksia voi tarvittaessa laajentaa JavaScript- (JS)[50] ja SPARQL-kielten toiminnoilla ([64] ja [63], vastaavasti).

Edellä mainittu ShEx-kielen kieliriippumaton laajennusmekanismi ansaitsee merkittävyytensä vuoksi hieman tarkemman selityksen. Mekanismin toiminta perustuu niin kutsuttuihin *semanttisiin toimintoihin* (engl. semantic actions), joiden avulla voidaan validoinnin aikana suorittaa millä tahansa toisella ohjelmointikielellä[51] operaatioita tai vaikka esimerkiksi ilmoittaa tapahtunut validointivirhe. Näitä toimintoja voidaan varsin vapaasti hyödyntää; esimerkkeinä toimivat RDF-tiedostojen muunto toiseen formaattiin ja skeeman muunto toiseksi skeemaksi[52]. Ainoana rajoitteena Labra Gayo ja kumppanit mainitsevat toimintojen soveltumattomuuden uusien, parametrisoitujen muotolausekkeiden tekoon, jota pidetään vielä myöhemmän ShEx-kielen kehitystyön tavoitteena. [68]

[49]Corman ja kumppanit ovat artikkelissaan [29] tosin esittäneet rekursiolle SHACL-kielessä formaalin, osittaisten ja *kerroksittaisten* (engl. stratification) muotojen sijoitteluun perustuvan semantiikan, joka tosin paljastuu NP-kovaksi (non-deterministic polynomial time). Sittemmin he ovat uudemmassa artikkelissaan [30] esittäneet ja kuvanneet hieman muunneltuun, niin kutsuttuun *tiukkaan kerroksellisuuteen* (engl. strict stratification) perustuvan algoritmin, joka heidän mukaansa takaa graafin validointiongelmalle laskennallisesti tehokkaan PTIME-luokan (polynomial time) aikavaativuuden.

[50]JavaScript on ECMA-262-standardiin (European Computer Manufacturers Association) [2] perustuva korkean tason komentosarjakieli.

[51]Toki sen täytyy olla järjestelmän käytettävissä.

[52]Nämä muunnot ovat samankaltaisista nimistään huolimatta hyvin erilaisia operaatioita.

Yhteenvetona voidaan todeta, että sovelluksen käyttötarkoitus vaikuttaa suurelta osin siihen, mikä linkitetyn datan rajoitekieli kannattaa kulloinkin valita toteutuksen perustaksi. Labra Gayo ja kumppanit muistuttavatkin kirjassaan [68], että eri käyttötarkoituksiin soveltuvia kieliä ja niiden ominaisuuksia on aina mahdollista yhdistellä käyttämällä tarvittaessa useampia niistä soveltuvin osin.

3.3.2 Olemassa olevat validaattorit

Kuten aliluvussa 3.1 käsiteltiin, validaattorit vastaavat validointiprosessissa rajoitteiden käsittelystä ja validointiraportin tuottamisesta. Aliluvussa 3.2 käsiteltiin rajoitteita sekä yleisessä että verkkomuotoisen datan (esimerkiksi linkitetyn datan) tapauksessa. Tämän lisäksi luvun 3 alussa todettiin rajoitteiden ja validoinnin jakaantuvan datan syntaktisten ja semanttisten ominaisuuksien perusteella kahtia. Lopuksi aliluvussa 3.3.1 tarkasteltiin linkitetyn datan validointikieliä ja -tapoja. Linkitetyn datan historian aikana on tuotettu useita erityyppisiä ja hyvin erilaisiin käyttötarkoituksiin soveltuvia validaattoreita. Seuraavaksi esitellään työn kannalta keskeisimmät, semanttisten ominaisuuksien perusteella validoivat toteutukset[53] kirjoittajan tietoon. Erityistä painoarvoa on annettu edelleen saatavilla oleville[54] ja tämän työn ohjelmalliseen toteutukseen (esitetty luvussa 5) vaikuttaneille, tunnetuille validaattoreille.

RDFUnit

RDFUnit[55] [66] (entiseltä nimeltään Databugger) on testiperustainen validaattori, joka muun muassa[56] tukee yleisimpiä RDFS- ja OWL-kielten aksioomia sekä SHACL-kielen rakenteita[57]. Näistä ensin mainituista aksioo-

[53]Kattavin ja päivitetyin listaus SHACL- ja ShEx-kielten osalta kirjoittajan tietoon on esitetty osoitteessa `https://github.com/validatingrdf/validatingrdf.github.io/wiki/Updated-list-of-implementations`.

[54]Kirjoittajan empiiristen havaintojen perusteella valitettavasti etenkin vanhojen tutkimusprojektien toteutuksilla on tapana kadota Internet-verkosta tai vain yksinkertaisesti lakata toimimasta.

[55]`http://aksw.org/Projects/RDFUnit.html`

[56]Muut ohjelman osittain tukemat skeemat ovat OSLC (Open Services for Lifestyle Collaboration) Resource Shape 2.0 [91] ja DC (Dublin Core) DSP (Description Set Profiles) [82], joiden jatkettu tuki tosin on lopetettu SHACL-kielen valmistumisen johdosta.

[57]Listaukset tällä hetkellä tuetuista aksioomista ja rakenteista ovat saatavilla osoitteissa `https://github.com/AKSW/RDFUnit/wiki/Overview` ja `https://github.com/AKSW/RDFUnit/issues/62`, vastaavasti.

mista ohjelma pystyy vieläpä skeeman perusteella automaattisesti generoimaan ohjelman testitapauksiksi niin kutsuttuja *datan laadun testimallineita* (engl. data quality test pattern, DQTP). Kaikki käytettävät testimallineet muunnetaan sisäisesti SPARQL-kyselyiksi erityisiä SPARQL-sapluunoita hyödyntäen[58], jolloin ohjelma voi tarvittaessa kohdistaa validoinnin suoraan datajoukon sisältämään SPARQL-palvelupisteeseen. Ohjelman tuottaman validointiraportin käyttäjä saa halutessaan HTML- tai RDF-muodossa. RDFUnit on yksi kehittyneimmistä avoimesti[59] saatavilla olevista validaattoreista.

TopBraid Composer

TBC (TopBraid Composer) [104] on TopQuadrant-yhtiön[60] kehittämä yksityisoikeudellinen, Java-perustainen [9] ja suhteellisen kehittynyt graafinen RDF- ja ontologiaeditori, jossa on kattava tuki datajoukkojen SPARQL-kyselyille ja validoinnille. Ohjelmalla voikin halutessaan validoida SHACL-, SPIN- ja SPARQL-kielillä, joille kaikille on tarjottu käyttöliittymässä oma näkymänsä. Näistä SHACL-[61] ja SPIN-kielten[62] toteutukset yhtiö on lisäksi julkaissut avoimena lähdekoodina. Muita TBC:n merkittäviä ominaisuuksia[63] tämän työn aihealueen kannalta ovat muun muassa:

- RDFS/OWL-aksioomien automaattinen muunto SHACL-kielisiksi
- SPARQL-käskyjen automaattinen generointi esimerkistä graafinäkymässä
- Sapluunat ja automaattinen tekstintäydennys
- Ohjelmaan yhdistetyn tietokannan vapaa muokkaaminen
- Skaalautuvien tietokantatoteutuksien käyttö datajoukoille
- Tuki erinäisille päättelykoneille
- SHACL-rajoitteiden automaattinen luonti datan perusteella

[58] Prosessi sisältää testimallineissa sijaitsevien muuttujien kiinnittämisen.
[59] `https://github.com/AKSW/RDFUnit`
[60] `https://www.topquadrant.com`
[61] `https://github.com/TopQuadrant/shacl/`
[62] `https://github.com/spinrdf/spinrdf`
[63] Huomautus: kaikki ohessa listatut ominaisuudet eivät ole käytettävissä ilmaiseksi saatavilla olevassa versiossa.

Stardog ICV

Stardog ICV (Integrity Constraint Validation) [27] on Stardog Union -yhtiön[64] kehittämän yksityisoikeudellisen Stardog-*tietämyskanta-alustan* (engl. knowledge graph platform) *eheysrajoitteista* (engl. integrity constraint) ja näiden validoinneista vastaava osa. Se tukee osittain tai kokonaan SHACL-, SPARQL-, SWRL- (Semantic Web Rule Language) [54] ja OWL 2 -rajoitekieliä oman Stardog Rules -syntaksinsa[65] lisäksi. [99] Järjestelmä muuttaa sisäisesti muunkieliset eheysrajoitteet SPARQL-kyselyiksi ja testaa annettuja rajoitteita [27]. Raportti[66] on listaus rikottuja eheysrajoitteita ja resursseja, joille ohjelma tosin pystyy erityisellä *explain*-komennolla tuottamaan käyttäjän ymmärrystä mahdollisesti lisäävän niin kutsutun *todistuspuun*[67] (engl. proof tree). *Fix*-komennolla ohjelma näyttää käyttäjälle korjausehdotuksen[68,69], jonka se tarvittaessa suorittaa yhtenä SPARQL UPDATE -operaatioiden [49] sarjana *execute*-valinnalla. [98]

Eräs mielenkiintoinen toiminto Stardog ICV:ssä on niin kutsuttu *vartiotila* (engl. guard mode), joka aktivoituna estää tietokantaan tehtävät sellaiset päivitykset, jotka transaktioperustaisen Stardog-tietämyskanta-alustan alaisessa tietokannassa rikkovat sille asetettuja eheysrajoitteita. [99] Tämä ominaisuus yhdistettynä mahdollisuuteen korjata virheitä puoliautomaattisesti tekevät Stardog ICV:stä varsin varteenotettavan validaattorin.

ELI Validator

ELI (European Legislation Identifier) Validator[70] on Euroopan unionin julkaisutoimiston[71] tuottama SHACL-perustainen[72] varmennepalvelu, joka on erityisesti sovitettu tarkastamaan julkaistujen, oikeudellisia tietoja standar-

[64] `https://www.stardog.com`

[65] Stardog Rules -säännöissä tavallisia SPARQL-osioita on aseteltu erityisiin *IF – THEN* -lohkoihin. [99]

[66] Stardog ICV:n SHACL-tuki mahdollistaa *report*-komennolla tuotettavien perusmuotoisten SHACL-validointiraporttien tuottamisen.

[67] Todistuspuut ovat näkymiä, joilla pyritään esittämään käyttäjälle datan, skeemojen ja sääntöjen rikkeet, väitteet ja päätellyt kolmikot ymmärrettävässä muodossa. [99]

[68] Useita ehdotuksia voi valita nähtäväkseen käyttämällä *limit*-parametria.

[69] Korjausehdotukset ovat ohjelman tuottamia, eikä menetelmästä ole saatavilla julkista dokumentaatiota.

[70] `https://publications.europa.eu/eli-validator/`

[71] `https://publications.europa.eu`

[72] ELI Validator käyttää SHACL-moottorinaan TopQuadrant-yhtiön avoimena lähdekoodina julkaisemaa TopBraid SHACL API (Application Programming Interface) -toteutusta.

doidulla tavalla kuvaavien ELI-metatietojen semanttinen oikeellisuus[73]. Sovelluksella on kuitenkin mahdollista validoida mitä tahansa SHACL-rajoitteita ja RDF-dataa. [37] Ohjelman selainperustainen graafinen käyttöliittymä näyttää kaikki validoidut solmut[74] ja soveltuu muun muassa seuraaviin käyttötapauksiin: [37]

- Sellaisten verkkosivujen validointiin, joihin on sisällytetty RDFa- tai JSON-LD-muotoista sisältöä
- RDF-dataa sisältävien tiedostojen validointiin
- Validointiraportin ja yksittäisten validointitulosten visualisointiin, joista raportoitavat asiat ryhmitellään vielä tarvittaessa tuloksen tuottaneen SHACL-muodon perusteella[75]
- Käytetyn muotograafitiedoston tutkimiseen
- Ohjelmalle syötetyn ELI-metatiedon muuntamiseen schema.org-yhteistoimintayhteisön[76] Legislation-laajennoksen[77] mukaiseksi metatiedoksi

RDFShape

RDFShape[78] [69] on avoimen lähdekoodin[79] selainperustainen linkitetyn datan validointi- ja kokeiluympäristö, joka soveltuu sekä SHACL- että ShEx-rajoitekieliseen validointiin. Validaattorin käyttämä, molempien kielten validointitoteutuksesta vastaava Shaclex-kirjasto[80] on varsin monipuolinen ja kattava, sillä se paketoi validointimoottorien lisäksi sovelluksen käyttöön muun muassa seuraavat merkittävät ja ohjelmassa hyödynnetyt ominaisuudet: [69]

- RDF-datan jäsentäminen ja muuntaminen toisiin sarjallistamismuotoihin

[73] Oikeellisuus tarkastetaan suhteessa ELI-ontologiaan tai tarkemmin sanottuna siihen perustuvaan SHACL-rajoitetiedostoon. [37]

[74] Ilmeisesti ohjelman käyttämässä SHACL-validointimoottorissa on tällainen tuki.

[75] Käyttöliittymä ei tekijöiden mukaan [37] kuitenkaan sovellu kaikkien raportoitavissa asioissa mahdollisesti olevien SHACL-rajoitteiden (kuten SHACL-SPARQL-rakenteiden) täydelliseen visualisointiin.

[76] `https://schema.org`

[77] `https://schema.org/Legislation`

[78] `http://rdfshape.weso.es`

[79] `https://github.com/labra/rdfshape`

[80] `https://github.com/labra/shaclex`

- SPARQL-toiminnallisuus (data- ja palvelupistekyselyt)[81]
- RDFS- ja OWL-päättelykoneet
- Muunnokset eri skeemaformaattien ja -kielien välillä[82]
- SHACL-raportoinnin yhteydessä kirjasto tuottaa todisteet kaikille käsitellyille solmuille (myös kelvollisille)

Graafisessa käyttöliittymässä voidaan lisäksi visualisoida ShEx-muotoja UML-tyyppisinä (Unified Modeling Language) [28] luokkakaavioina[83]. Eräs kiinnostava ja hyödyllinen erikoisominaisuus RDFShape-validaattorissa on se, että sen rajapintametodit[84] on toteutettu funktionaalisina monadeina ja ovat täten ketjutettavissa ja kytkettävissä RDF-datan käsittelyputkeen. Tämä pätee myös Shaclex-kirjaston validointimetodeihin. Nämä edellä mainitut ominaisuudet tekevät RDFShape-ohjelmasta varsin käytännöllisen työkalun ja sitä onkin käytetty useilla semanttisen webin kursseilla ja monissa oppimateriaaleissa. [69]

[81] Palvelupistekyselyistä palautuvaa dataa voidaan vieläpä suoraan käyttää muotorajoitteita vastaan.

[82] Kirjoittajan empiiristen havaintojen mukaan erityisesti SHACL- ja ShEx-kielten välisten muunnosten toiminnassa olisi parantamisen varaa.

[83] Visualisointi tehdään selaimen ymmärtämää SVG-grafiikkaa (Scalable Vector Graphics) [32] hyödyntäen.

[84] Tarjolla on sekä selainkäyttöliittymä että REST-rajapinta.

Luku 4

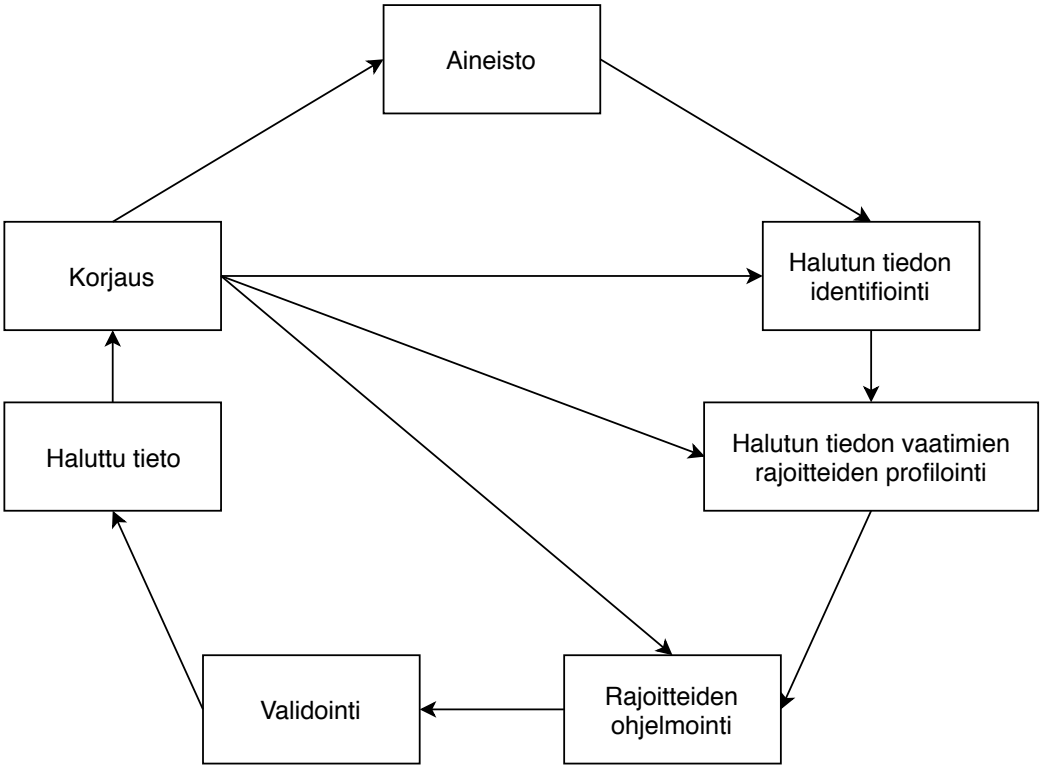
Menetelmä linkitetyn datan validoimiseksi ja korjaamiseksi

Tämän työn tutkimuskysymyksenä oli löytää ratkaisu ongelmaan, jossa validointituloksia tulkittaessa havaitaan virheitä datassa ja ne haluttaisiin heti korjata. Edellisissä luvuissa ei kuitenkaan käsitelty linkitetyn datan korjaamista niissä tilanteissa, joissa validointiraportti tuottaa hylätyn tuloksen. Tätä ongelmaa ei ole kirjallisuudessa käsitelty kattavasti kuten Paulheim kirjallisuuskatsauksessaan [84] havaitsee[1]. Ongelma on kuitenkin mahdollista ratkaista nykyteknologialla puoliautomaattisesti[2] seuraavalla tavalla: muokataan kuvan 3.1 validointiprosessia lisäämällä siihen tutkimuskysymyksessä haluttu korjaamisominaisuus, josta tulee prosessin seitsemäs ja viimeinen vaihe. Tämän jälkeen huomataan, että prosessia voidaan luonnollisesti laajentaa tekemällä siitä iteratiivinen: korjattu data vastaa ensimmäistä vaihetta eli uutta aineistoa, jolloin näiden välille muodostuu suhde. Toisaalta riippuen korjauksen vaatimasta tavasta, voi korjaus kohdistua validoinnissa käytetyn rajoitteen säätämiseen; tällöin suhde muodostuukin korjausvaiheesta suoraan kolmanteen vaiheeseen, sillä korjaus vaatii rajoitteen uudelleensuunnittelua ennen sen uudelleenohjelmointia. Puoliautomaattiseksi mahdollisen teknisen järjestelmän tekee se, että ohjelma pystyy tarvittaessa automaattisesti avus-

[1] Tarkemmin sanottuna Paulheim toteaa, että sellaisia yhdistelmämenetelmiä ei ole juurikaan esitetty, jotka samanaikaisesti yrittävät sekä parantaa tietämysgraafin *täydellisyyttä* (engl. completeness) että *oikeellisuutta* (engl. correctness), ja jos on, ovat ne keskittyneet vain jonkin tietyn ominaisuuden, kuten esimerkiksi pelkkien tyyppi- tai suhdeväittämien tai literaaliarvojen parantamiseen. Lisäksi holistisia ratkaisuja, jotka olisivat useammista näkökulmista parantaneet tietämysgraafia ei Paulheim havainnut ollenkaan katsausta tehdessään.

[2] Täysin automaattinen ratkaisu vaatisi sitä aliluvussa 3.1 mainittua täydellistä HMI-rajapintaa, jota ei vielä valitettavasti ole kehitetty.

tamaan käyttäjää korjauksissa. Tämä on mahdollista esimerkiksi siten, että ohjelma ehdottaa ja tarjoaa ratkaisuvaihtoehtoja käyttäjälle siitä, miten virhe voitaisiin korjata, mahdollisesti toistaen saman korjauksen vielä kaikille vastaaville virheille[3]. Näitä korjauksia voidaan toteuttaa suoraan ensimmäiseen ja neljänteen vaiheeseen, jotka edustavat varsinaista dataa sovelluksessa. Lopuksi on vielä ajateltava sellaista tilannetta, jossa haluttu tieto tarkentuu saadun validointiraportin avulla. Tämänkaltaista tilannetta varten tarvitaan suhde toiseen vaiheeseen. Uusi, edellä kehitetyn mallin mukainen validointiprosessi on esitetty kuvassa 4.1.



Kuva 4.1: Validoinnin ja korjaamisen iteratiivinen validointiprosessi. Ensimmäinen vaihe on kuvattu ylimpänä.

[3] Tarkempi listaus ja kuvaus kaikista aidosti kehittyneeltä korjaus- ja ehdotusjärjestelmältä vaadittavista ominaisuuksista (esimerkiksi oikeiden *linkki- ja tyyppiennusteiden* [engl. link prediction ja type prediction, vastaavasti] tuottaminen) on tämän työn laajuuden ulkopuolella. Aiheesta kiinnostunutta lukijaa suositellaan tutustumaan Paulheimin perusteellisen kirjallisuuskatsauksen [84] lisäksi esimerkiksi Melon väitöskirjaan [78], jossa on suhteellisen vastikään tutkittu ja kehitetty aiheen tiimoilta erityisesti suurille tietämysgraafeille sopivia ABox- (Assertion Box) ja TBox-perustaisia (Terminology Box) (puoli)automaattisia jalostusmenetelmiä.

Kuvan 4.1 korjaamisvaiheesta lähtevät neljä suhdetta esittävät kaikki vaihtoehtoisia, toisensa poissulkevia vaiheistuksia, kun virheitä ja niiden syitä tarkastellaan analyyttisesti. On toki mahdollista tehdä useampia korjauksia dataan ja/tai rajoitteisiin yhdellä korjauskerralla, mutta nämä voidaan ajatella erillisinä työvaiheina iteratiivisessa prosessissa.

Esimerkiksi taulukossa 3.1 esitetyn validointiraportin tapauksessa käyttäjä voi päivitetyn, iteratiivisen validointiprosessin mukaisesti joko korjata hylätyt alkiot hyväksytyiksi[4], hyväksyä ohjelman ehdottomat (puoliautomaattiset) muutokset dataan ja/tai rajoitteisiin, tai laventaa rajoitteessa käytettyä kaavaa 3.1[5] ja ajaa syöte (yhä) uudestaan aliluvussa 3.2 mainitulla kuvitteellisella validaattorilla saaden näin lopulta validin syötteen ja virheettömän validointiraportin. Tässä tapauksessa siirtyminen korjauksesta suoraan toisen vaiheeseen ei ole luonnollinen; tämä siirtymä on tarkoitettu ennemminkin hyväksyttyjen rajoitteiden jälkeen edelleen datassa olevien puutteiden korjaamiseen. Toki useampien rajoitteiden ollessa samaan aikaan tarkasteltavana on mahdollista, että jokin toinen rajoite aiheuttaa tällaisen siirtymän prosessissa, mutta tämäkin voidaan toissa kappaleessa esitetyn tapauksen tavoin tulkita iteratiivisessa prosessissa omana ilmentymänään.

Tässä luvussa aiemmin mainittua termiä ”puoliautomaattinen” voidaan havainnollistaa seuraavasti: sovellus lähettää dataa validaattorille, joka vastaa validointiraportilla[6]. Sovelluksen on mahdollista yhdistellä saamiansa tietoja rajoitteista sekä virheistä ja muodostaa analyysi siitä, miten esimerkiksi yksittäinen rajoite ei datassa toteutunut. Tämä vaatii tietoa rajoitteen kohteesta, käytetystä aineistosta sekä rajoitteen käyttämistä ominaisuuksista; esimerkiksi rajoitteen, joka vaatii kaikille SKOS-sanaston (Simple Knowledge Organization System) [79] mukaisille subjekteille, jotka ovat luokan skos:Concept yksilöitä, predikaatille skos:prefLabel arvon suomen kielellä, rikkominen voisi ehdottaa ohjelman toimesta subjektina olevan resurssin poistoa tai oletusarvoisen predikaatin skos:prefLabel arvon[7] toistamis-

[4]Esimerkiksi poistamalla nämä alkiot.

[5]Esimerkiksi pienentämällä y-rajoitteen alarajaa yhdellä ja poistamalla x-rajoite kokonaan.

[6]Tämä saattaa aiheuttaa ongelmia lähetettäessä dataa, joka sisältää anonyymejä solmuja, sillä ne eivät säilytä identiteettiään esiintyessään eri dokumenteissa. Kaikki mahdolliset anonyymit solmut on kuitenkin mahdollista muuntaa IRI-muotoisiksi, jolloin niiden säilyminen samana validointiprosessin aikana voidaan taata [31].

[7]Esimerkiksi kielikoodittoman arvon, jos tällainen on olemassa.

ta tämän predikaatin suomenkieliseksi arvoksi[8]. Toisaalta taas esimerkiksi joihinkin rajoitekieliin mahdollisesti määritetyt predikaattien lukumäärärajoitteiden virheiden korjausehdotukset voidaan esittää käyttäjälle pyyntöinä joko täyttää tai poistaa predikaatteja valinnan mukaan. Tällaisessa tilanteessa järjestelmä voi automaattisesti tarvittaessa päättää (jos predikaatilla on ylen määrin arvoja) minkä arvon se jättää[9]. Yleistä ratkaisua ei kuitenkaan ole olemassa sovelluksen tietojen puitteissa, minkä takia vuorovaikutus käyttäjän kanssa on välttämätöntä ja kuvan 4.1 toteuttava järjestelmä puoliautomaattinen parhaimmassakin tapauksessa.

Teknisinä huomioina lisättäköön ensinnäkin, että aineiston sijaitessa SPARQL-palvelupisteessä voidaan ehdotettu korjaus tarvittaessa suorittaa järjestelmän toimesta suoraan kyseiseen palvelupisteeseen käyttäen tähän tarkoitukseen sopivaa SPARQL-päivityspyyntöä. Toisekseen, prosessin vaiheisiin 3–4 vaikuttavat kiinteästi aineistossa käytetyt ontologiat, ja käyttäjän tuleekin olla varsin hyvin perehtynyt niihin (datan lisäksi) rajoitteita muodostaakseen, ellei käytettyjen sanastojen mukana tule niiden rajoiteskeemoja valitussa rajoitekielisessä muodossa. Esimerkiksi SHACL-kielessä määriteltyjen muotojen rajoiteskeema aineistona olevien muotojen validoimiseksi on saatavilla osoitteessa `https://w3.org/ns/shacl-shacl#`[10], mutta vastaavaa skeemaa ei SHACL-kielellä esitettynä ole kirjoittajan tietoon saatavilla esimerkiksi RDFS:n osalta. Tämä tarkoittaa sitä, että tällöin esimerkiksi rajoitteena ymmärrettävät, aineistograafissa olevat RDFS-sanaston avulla ilmaistut arvo- ja lähtöjoukkomäärittelyt tulee itse ohjelmoida SHACL-kielellä (ja kohdistuen vieläpä suoraan näihin RDFS-lausekolmikoihin), ellei järjestelmässä muuten ole tukea tällaisten rajoitteiden ”omakieliseen” validointiin.

[8]Tässä tapauksessa aidosti hyvän ratkaisun löytäminen automaattisesti voi olla mahdotonta, mutta Suominen ja Mader esittävät artikkelissaan [102] juuri tähän tilanteeseen hyvän ehdotuksen: luonnollisen kielen tekniikoita käyttäviä järjestelmiä voisi hyödyntää kääntämään subjektin suositeltava nimike toiselle kielelle monikielisissä ontologioissa.

[9]Esimerkiksi Skosify-muunnostyökalu [101] valitsee itse käyttämänsä heuristiikan perusteella jätetyn skos:prefLabel-arvon, jos näitä on useita samalla subjektilla jollekin kielikoodille. Käytetty heuristiikka on oletusarvoisesti Skosify-työkalussa lyhin mahdollinen arvo [101], joka on hyvä approksimaatio tälle ongelmalle.

[10]Huomautus: kyseisessä osoitteessa julkaistu skeema on tätä kirjoitettaessa (18.7.2019) luonnos eikä sisällä kaikkia SHACL-kielen syntaktisia sääntöjä.

Luku 5

ROTEVA-rajoitevalidaattori

Työn tavoitteissa asetettuihin haasteisiin on vastattu toteuttamalla ohjelmallinen esimerkkitoteutus, joka pyrkii vastaamaan työn laajuudessa mahdollisimman tarkkaan[1] luvussa 4 esitettyä iteratiivista validointi- ja korjausprosessia. Tämä toteutus on nimetty työn teemaan sopivasti ROTEVAksi (RajOiTEVAlidaattori).

Luvun rakenne on seuraavanlainen: aliluvussa 5.1 esitetään validaattorin kokonaisarkkitehtuuri. Sovelluksen ajamiseen vaadittava käyttöympäristö ja ajo-ohjeet käydään läpi aliluvussa 5.2. Aliluvussa 5.3 syvennytään validaattorin tarjoamaan ohjelmalliseen rajapintaan ja sen tuottaman validointiraportin rakenteeseen, jotka molemmat tulevat käyttöön aliluvussa 5.4 esitetyssä graafisen käyttöliittymän tarkemmassa käyttö- ja rakennekuvauksessa.

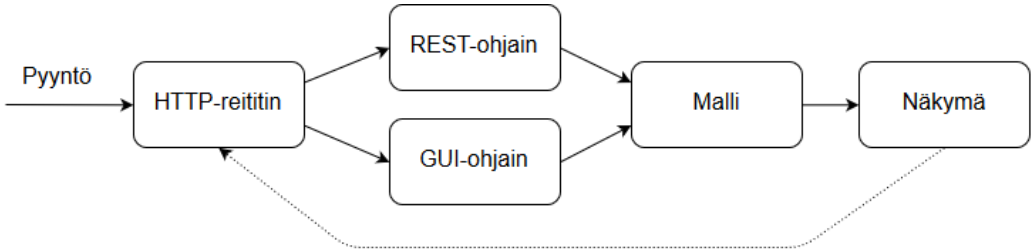
5.1 Arkkitehtuuri

ROTEVAn arkkitehtuuri perustuu Play Framework -ohjelmistokehyksen[2] käyttämään MVC-malliin (Model-View-Controller) [105]. MVC-mallissa on tapana eristää omiksi komponenteikseen *malli* (M), *näkymä* (V) ja *ohjain* (C), jolloin mikä tahansa niistä voidaan tarvittaessa korvata toisella toteutuksella. Toisaalta tämä helpottaa myös ohjelmakoodin ylläpitoa, kun osien välillä on selkeästi määritellyt rajapinnat. Ohjelmistokehys mahdollistaa kevyiden, tilattomien verkkosovellusten tekemisen, jota on hyödynnetty tässä luvussa esiteltävän sovelluksen tarkemmassa suunnittelussa ja toteutuksessa.

[1] Luvussa 4 esitetyn iteratiivisen validointi- ja korjausprosessin täydellinen toteutus on tämän työn laajuuden ulkopuolella.

[2] `https://www.playframework.com`

Kuvassa 5.1 on esitetty ROTEVAn MVC-malliin perustuva arkkitehtuuri. Ohjelmalle tuleva pyyntö ohjautuu ensin ohjelmistokehyksen HTTP-reitittimelle, joka puolestaan ohjaa pyynnön joko ohjelmointirajapinnan REST-ohjaimen tai graafisen käyttöliittymän GUI-ohjaimen (Graphical User Interface) käsiteltäväksi. Nämä ohjaimet on toteutettu työn aikana vastaamaan palvelulle asetettuja vaatimuksia; REST-ohjain (esitetty tarkemmin aliluvussa 5.3) vastaa ROTEVAn ohjelmallisesta rajapinnasta ja GUI-ohjain (esitetty tarkemmin aliluvussa 5.4) graafisessa käyttöliittymässä esitetystä näkymästä. Molemmat ohjaimet tuottavat ensin MVC:n mukaisen mallin, jonka perusteella käyttäjälle näytettävä näkymä tuotetaan. Katkoviivaisella nuolella on esitetty graafisen käyttöliittymän näkymästä lähetettävät pyynnöt takaisin palvelimelle validoinnin suorittamiseksi ja uuden, iteratiivisen näkymän tuottamiseksi[3].



Kuva 5.1: ROTEVAn MVC-malliin perustuva arkkitehtuuri.

Valittu arkkitehtuuri mahdollistaa keskitetyn ja eristetyn ohjelman hallinnan. Toteutusta varten luvussa 4 esitetyn menetelmän rajoitekieleksi valikoitui aliluvussa 3.3.1 esitetty SHACL-rajoitekieli, sillä se vastaa kaikkiin luvussa 1 esitettyihin validointivaatimuksiin[4], tuottaa samaisen aliluvun mukaan tarkimman validointiraportin ja on vakiintunut saaden virallisen W3C:n suosituksen ja useita esimerkkitoteutuksia[5], joista voitiin valita työhön sopivin. Tämän lisäksi Tomaszukin toteuttamassa suorituskykytestauksessa [103] SHACL suoriutui järjestelmällisesti ShEx-kieltä nopeammin jo suhteellisen pienellä aineistolla, jolla oli vaikutusta valintaan.

[3]Graafisen käyttöliittymän validoinnista vastaava osuus on käyttäjän kanssa vuorovaikutteinen yhden Internet-sivun sovellus.

[4]Itse asiassa jo SHACL-kielen vaadittujen käyttötapausten listaus [100] enemmän kuin ylittää työn ohjelmallisen osan validointikieleltä vaatimat ominaisuudet.

[5]Valmiita SHACL-kielen validaattoritoteutuksia ovat muun muassa Corese SHACL, Netage SHACL Engine, RDFUnit ja TopBraid SHACL API. [70]

5.2 Käyttöympäristö

ROTEVAn toteutus on – kuten mikä tahansa tietokoneohjelma – ajettava jossakin ympäristössä. Tätä ennen ohjelmakoodin on kuitenkin oltava käännettynä ajettavaksi tavukoodiksi. Tässä alivussa käydään läpi tähän vaadittavat toimenpiteet, ja se rakentuu seuraavalla tavalla: aliluvussa 5.2.1 määritellään ROTEVA-rajoitevalidaattorin palvelinympäristölle asettamat vähimmäisvaatimukset ja tarvittavat asetukset. Ohjeet ohjelman lataamiseksi kerrotaan yksityiskohtaisesti aliluvussa 5.2.2. Aliluvussa 5.2.3 kuvataan ohjelman koostaminen ajettavaksi paketiksi, jonka ajoon ja käyttöön annetaan ohjeet aliluvussa 5.2.4.

5.2.1 Palvelinympäristö

ROTEVAa voidaan ajaa millä tahansa palvelimella, jolla on Java versionumero 8 (tai korkeampi) asennettuna, sillä se on rakennettu Java 8 -yhteensopivan Play Framework -ohjelmistokehyksen päälle. Palvelimella ajettavan sovelluksen käyttö vaatii lisäksi palomuurin ja muiden verkkoasetusten asettamista oikein REST-rajapinnan ja verkkosovelluksen saavutettavuuden hallitsemiseksi; esimerkiksi kehitysympäristössä ROTEVA kuuntelee oletuksena saapuvaa HTTP-liikennettä portissa 9000[6]. Suositeltavana vähimmäisvaatimuksena ROTEVAn käytettäväksi on syytä varata vähintään kaksi gigatavua muistia, sillä nykyasetuksilla ROTEVA lukee koko annetun datagraafin muistiin, ja keskusmuistin loppuminen kesken kaataa ohjelman[7].

5.2.2 Ohjelman lataaminen

ROTEVAn avoimena lähdekoodina saatavissa oleva ohjelmakoodi on ladattavissa Aalto-yliopiston ylläpitämän GitLab-verkkoversionhallintajärjestelmän[8] säilöstä, jota hallinnoi Semanttisen laskennan tutkimusryhmä SeCo[9] osoitteessa

https://version.aalto.fi/gitlab/seco/Roteva

[6]Tarkemmat tiedot liikenteen ohjaamisesta on esitetty aliluvussa 5.2.4.

[7]Tämä toiminallisuus on helposti vaihdettavissa massamuistiperustaiseen *app/api/-rest/RestController.java*-tiedostosta, jonka jälkeen ohjelma on tosin käännettävä uudestaan.

[8]`https://about.gitlab.com`

[9]`https://seco.cs.aalto.fi`

josta ohjelmakoodin voi ladata suoraan muodoissa **zip**, **tar.gz**, **tar.bz2** tai **tar** lisäämällä edellä olevan osoitteen perään

/-/archive/master/Roteva-master.$TIEDOSTOMUOTO

jossa $TIEDOSTOMUOTO on jokin yllä olevista neljästä eri arkistomuodon tiedostopäätteistä. Lisäksi koko versiohistoria on kloonattavissa minkä tahansa Git-perustaisen[10] asiakasohjelmiston avulla verkko-osoitteesta

https://version.aalto.fi/gitlab/seco/Roteva.git

jossa annettu osoite on komennon *git clone* pakollinen säilö-parametri.

5.2.3 Ohjelman kokoaminen

Ladattu (ja tarvittaessa purettu) lähdekoodi on seuraavaksi käännettävä, jotta koodin ajaminen onnistuu. Koska ROTEVAn taustalla oleva ohjelmistokehys on toteutettu *sbt*-koontijärjestelmää[11] (Scala Build Tool) hyödyntäen, on ohjelmiston kääntäminen yksinkertaista: lähdekoodin päähakemistossa sijaitsevassa *build.sbt*-tiedostossa on Scala-ohjelmointikielellä [83] esitetyt asetukset sbt:lle ohjelman koontia varten ja *project*-hakemistossa sijaitsevat loput tarvittavat määritelmät projektin koontiin. Päähakemistossa sijaitsevat tiedostot *activator* ja *activator.bat*, jotka vastaavasti ovat *NIX- ja Windows-käyttöjärjestelmiä käyttävien tietokoneiden komentorivitulkkien kanssa yhteensopivia, ajettavia komentojonotiedostoja, kutsuvat päähakemistossa sijaitsevaa ajettavaa *activator-launch-1.3.7.jar*-Java-arkistotiedostoa. Tämä *activator*-komentojonotiedoston käyttämä ohjelma sisältää sbt:n, jolloin sitä ei tarvitse erikseen asentaa[12], ja ROTEVA on koottavissa zip-paketiksi suorittamalla vain komento *activator dist*[13]. Vaihtoehtoisesti voidaan käyttää sbt:n natiivipakkainta[14], jolla voidaan luoda erityyppisiä, ajovalmiita arkistotiedostoja useille eri alustoille ja ohjelmistoille. Koontipaketti on nyt valmis siirrettäväksi palvelimelle. On myös mahdollista ajaa komento *activator clean stage*, joka luo ajovalmiin paketin *target/universal/stage*-hakemistoon.

[10] `https://git-scm.com`

[11] `https://www.scala-sbt.org`

[12] On kuitenkin mahdollista asentaa erikseen sbt tietokoneelle, jolloin komennon *activator* tilalla voidaan tästedes käyttää suoraan komentoa *sbt*.

[13] Valmis zip-tiedosto on käytettävissä tämän jälkeen tiedostopolussa *target/universal/roteva-1.0.zip*.

[14] `http://sbt-native-packager.readthedocs.io/en/v1.3.4/index.html`

5.2.4 Ohjelman ajaminen

Ennen tuotantoversion ajoa on tärkeää luoda ennalta arvaamaton salausavain[15] ohjelman ajamiseksi turvallisesti[16]. Eräs mahdollisuus[17] on käyttää tähän komentoa *activator playGenerateSecret*, joka antaa käyttäjälle tarpeeksi pitkän ja monimutkaisen salausavaimen. Tämän jälkeen ohjelma voidaan ajaa komennolla

$POLKU/bin/roteva -Dplay.crypto.secret=$SALAUSAVAIN

jossa $POLKU on polku aliluvun 5.2.3 ohjeiden mukaan koottuun (ja tarvittaessa purettuun) hakemistoon[18] ja $SALAUSAVAIN on esimerkiksi edellä mainitulla tavalla luotu merkkijono. Ohjelma käynnistyy ja ilmoittaa oletustulostusvirtaan kuuntelemansa portin numeron ja osoitteen. Nämä voidaan tarvittaessa vaihtaa käyttäen ajoparametreja *-Dhttp.port=$PORT* ja *-Dhttp.address=$ADDRESS*, joissa $PORT on ohjelman kuuntelema TCP/IP-pinon (Transmission Control Protocol, Internet Protocol) [44] portti HTTP-liikenteelle ja $ADDRESS ohjelman kuuntelema DNS-nimi (Domain Name System) [80] tai IP-osoite [34], vastaavasti.

Kehitysversiona ROTEVAa voi ajaa huomattavasti yksinkertaisemmin kuin tuotantoversiona, sillä silloin ajamiseen riittää komento *activator ~run*. Tällöin muutokset ohjelmakoodiin havaitaan automaattisesti ja ROTEVAn lähdekoodi käännetään välittömästi uudestaan, mikä nopeuttaa muutosten näkymistä käytännössä. Kehityksen helpottamiseksi voidaan myös ajaa sbt-komento *activator "eclipse with-source"*, jolloin sbt generoi ROTEVAsta Eclipse IDE (Integrated Development Environment) -kehitysympäristön[19]

[15]Salausavaimen haltuunsa saanut pystyy kirjautumaan järjestelmään minä vain käyttäjänä hän haluaa [74].

[16]Periaatteessa ROTEVAn käyttöä ei haittaa salausavaimen turvattomuus, sillä siinä ei säilytetä istuntoja, mutta mahdollisen kirjautumisen lisääminen tähän sovellukseen aiheuttaisi ongelmia asian suhteen. Lisäksi Play Framework -ohjelmistokehys estää käynnistämisen ja antaa virheilmoituksen, jos salausavainta ei vaihda ja ohjelman tuotantoversiota yritetään ajaa.

[17]Toinen vaihtoehto on ajaa komento *activator playUpdateSecret*, joka korvaa kovakoodatun arvon *conf/application.conf*-tiedostossa. Käyttäjän on myös itse mahdollista korvata salausavain tässä samassa tiedostossa joko suoraan tai luomalla toisen tiedoston, joka ensin sisällyttää (*include "application"*) aiemman konfiguraatiotiedoston, ja sitten yliajaa *play.crypto.secret*-muuttujan arvon. Tällöin ohjelman yhdeksi ajoparametriksi asetetaan *-Dconfig.file=oma-konfiguraatiotiedosto*. [74]

[18]Esimerkiksi *target/universal/stage*.

[19]`https://www.eclipse.org/eclipseide/`

sujuvaan käyttöön[20] vaadittavat projekti- ja ympäristötiedostot (*.project* ja *.classpath*, vastaavasti). Ohjelma on nyt ajokomennon jälkeen valmis käytettäväksi.

5.3 Ohjelmallinen rajapinta

Kuvassa 5.1 esitetty ROTEVAn MVC-mallin REST-ohjain vastaa sovelluksen ohjelmallisesta rajapinnasta. Kuten kuvasta ilmenee, tämä ohjaimen rajapinta[21] vastaa mallin ja tästä luodun näkymän tuottamisesta. Rajapinnan tarjoama toiminnallisuus on saavutettavissa lähettämällä pyyntö alilukujen 5.2.1 ja 5.2.4 ohjeiden mukaan konfiguroituun, ohjelman kuuntelemaan ja REST-polulla */api/rest/* täydennettyyn osoitteeseen. ROTEVA hyväksyy sekä JSON-koodatun että HTML-lomakkeesta[22] lähetetyn datan. ROTEVAn SHACL-moottoriksi päätettiin valita TopBraid SHACL API -toteutus[23], joka ohjelmointiajankohdan aikaan ainoana moottorina läpäisi kaikki SHACL-rajoitekielelle asetetut yksikkötestit [70]. Lisäksi kyseisen moottorin käyttö aliluvussa 3.3.2 esitetyssä ELI Validator -varmennepalvelussa osoitti sen ominaisuuksien olevan riittävät tämän työn tarpeisiin. Työ laajentaa moottoria käärimällä sen validointimetodin REST-rajapinnan avulla käytettäväksi.

Kuvassa 5.2 on esitetty yleinen ja täten myös käytetyn SHACL-toteutuksen tuottaman validointiraportin rakenne. Kuvassa resurssit on kuvattu pyöristetyillä suorakulmioilla ja literaalit suorakulmioilla. Yksisuuntaiset nuolet kuvaavat suhdetta subjektista objektiin. Muuttuja n on määritelty luonnolliseksi luvuksi. Nuolen päällä oleva kirjoitus kertoo esitetyn suhteen predikaatin[24]. Taulukossa 5.1 on esitetty sallitut arvot ja määritelmät kullekin kuvassa 5.2 esitetylle predikaatille sekä kaikille ROTEVAn validointiraportissa mahdollisesti esiintyville käytetyn SHACL-validaattoritoteutuksen tukemille vapaaehtoisille predikaateille.

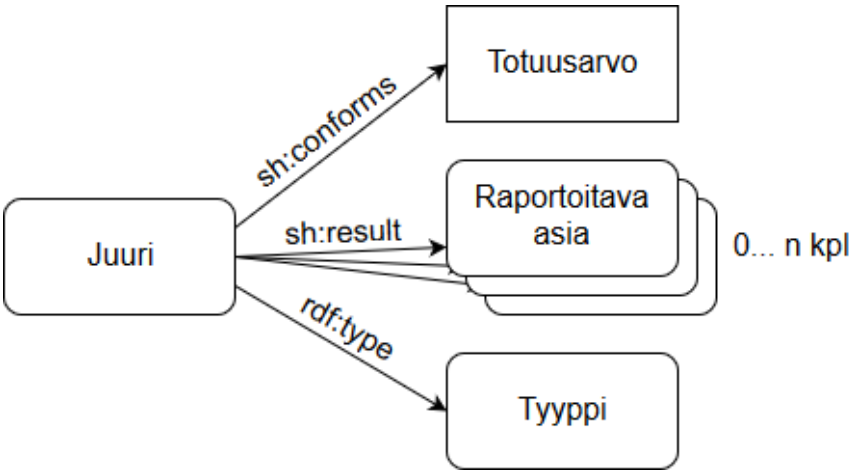
[20]Komento tuottaa kehitysympäristön käyttöön projektitietojen lisäksi sovelluksessa käytettyjen ohjelmistokirjastojen saatavilla olevat lähdekoodit.

[21]Esitetty liitteessä A.

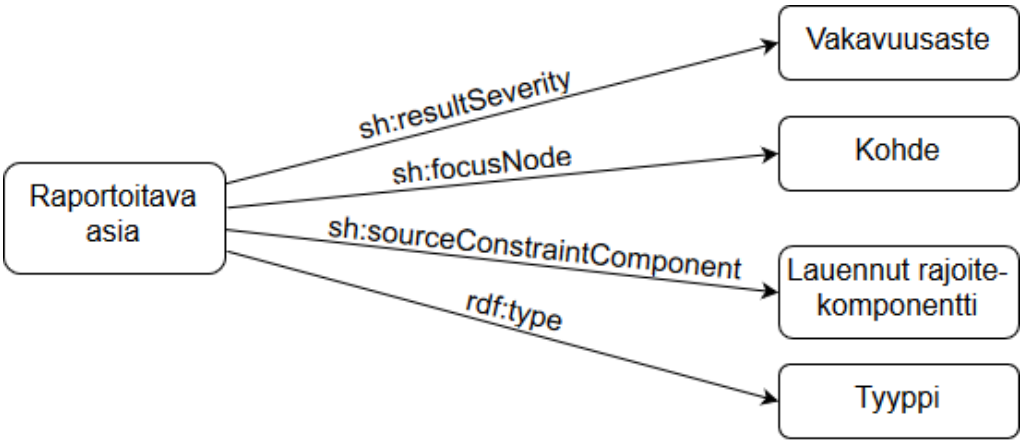
[22]Tällöin HTTP-pyynnön tulee sisältää *Content-type*-otsikko, jossa on määritelty tietotyypiksi HTML-lomakkeissa käytetty *application/x-www-form-urlencoded-* tai *multipart/form-data*-koodaus.

[23]`https://github.com/TopQuadrant/shacl/tree/shacl-1.2.1`

[24]Predikaattien ja näiden arvojen kuvaamiseen on käytetty seuraavia nimiavaruuksia:
rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>,
rdfs: <http://www.w3.org/2000/01/rdf-schema#>,
sh: <http://www.w3.org/ns/shacl#> ja
xsd: <http://www.w3.org/2001/XMLSchema#>.



(a) Rakenteen ylin osa.



(b) Raportoitavan asian pakolliset ominaisuudet.

Kuva 5.2: Yleinen ja täten myös ohjelmallisessa työssä käytetyn TopBraid SHACL API -validaattoritoteuksen tuottaman validointiraportin rakenne[24].

Predikaatti	Määritelmä	Sallitut arvot
	Validointiraportti	
sh:conforms	Tosi, jos data noudattaa muotoja, muuten epätosi.[a]	xsd:boolean-literaali
sh:result	Raportoitava asia.[b]	sh:Validation-Result-yksilö
rdf:type	Validointiraportin tyyppi.[a]	sh:Validation-Report
	Raportoitava asia	
sh:resultSeverity	Raportoitavan asian vakavuusaste.[a]	IRI
sh:focusNode	Kohdesolmu, jonka validointi tuotti tämän raportoitavan asian.[a]	IRI, anonyymi solmu
sh:source-Constraint-Component	Rajoitekomponentti, jonka laukeaminen tuotti tämän raportoitavan asian.[a]	sh:Constraint-Component-yksilö
sh:source-Constraint	Rajoite, jonka laukeaminen tuotti tämän raportoitavan asian.[a]	IRI
rdf:type	Raportoitavan asian tyyppi.[a]	sh:Validation-Result
sh:resultPath	Raportoitavan asian tuottanut predikaattipolku.[a]	rdfs:Resource-yksilö
sh:value	Raportoitavan asian aiheuttanut arvo.[a]	RDF-termi
sh:sourceShape	Muoto, jota vastaan kohdesolmu validoitiin.[a]	sh:Shape-yksilö
sh:resultMessage	Ihmisluettava viesti, joka selittää syyn tälle raportoitavalle asialle.[b]	xsd:string- tai rdf:langString-literaali

Taulukko 5.1: ROTEVAn validointiraportissa käytettävien predikaattien määritelmät sekä niiden sallitut arvot[24].

[a] Ei toistettava. [b] Toistettava.

Listauksessa 5.1 on esitetty esimerkkisyöte, jonka lähettäminen REST-rajapinnan *validate*-metodille[25] tuottaa vastauksena listauksen 5.2 mukaisen tulosteen. Kuten listauksen 5.2 tulosteesta nähdään, on SHACL-rajoitekielen toteuttavan validaattorin vastaus SHACL-raportin rakennetta kuvaavan kuvan 5.2 mukainen. Tulosteessa on määritelty useampia suhteita kuin mitä kuvassa 5.2(b) on esitetty; tämä onkin sallittua, sillä kuten taulukossa 5.1 on luetteloitu, sallii SHACL-spesifikaatio pakollisten ominaisuuksien lisäksi muun muassa predikaattien[24] *sh:focusNode*, *sh:resultPath*, *sh:value*, *sh:sourceShape*, *sh:detail*[26] ja *sh:resultMessage* määrityksen, joista vielä kahta viimeistä on mahdollista toistaa [63]. Lisäksi validointiraportti voi sisältää[27] mielivaltaista provenanssitietoa ja sillä voi olla predikaatti *sh:shapesGraphWellFormed*[28], joka ilmaisee validaattorille annetun muotograafin oikeamuotoisuuden [63]. Listauksen 5.2 tuloste on samalla yksi mahdollinen näkymä listauksen 5.1 syötteelle. Muita näkymiä voidaan tuottaa joko asettamalla *to*-parametrin tai *Accept*-otsakkeen johonkin REST-ohjaimen sisäisesti käyttämän RDF-kirjoittimen[29] tukemaan formaattiin, kuten liitteessä A ohjeistetaan.

5.4 Graafinen käyttöliittymä

ROTEVAn intuitiivinen käyttö vaatii graafisen käyttöliittymän, jotta käyttäjää voidaan luvussa 1 annettujen työn tavoitteiden mukaisesti avustaa rajoitteiden muodostamisessa. Validoinnin tulokset voidaan visualisoida käyttäjälle käyttöliittymän avulla havainnollisemmin, mitä onkin hyödynnetty luvussa 4 esitetyn iteratiivisen validointi- ja korjausprosessin toteutuksessa. Työn edetessä tämän prosessin automatisointi avustettuine korjausehdotuksineen osoittautui kuitenkin liian suuritöiseksi työn laajuuteen nähden. Tästä syystä graafisen käyttöliittymän toteutuksessa keskitytäänkin iteratiivisen validointi- ja korjausprosessin mahdollisimman sulavaan toimintaan ja vuoro-

[25]Kun parametria *to* ei ole asetettu, tulostaa ohjelma oletuksena JSON-LD-koodattua dataa.

[26]Linkittää raportoitavan asian toisen raportoitavan asian kanssa tuottaakseen lisäinformaatiota ja mahdollistaa esimerkiksi aliluvussa 3.3.2 esitetyn ELI Validator -varmennepalvelun kaikkien validoimien kohdesolmujen listaamisen. Tämä ei kuitenkaan ole työssä käytetyn moottoriversion (1.2.1) tukema ominaisuus käytetyllä validointimetodilla, eikä sitä täten ole listattu taulukossa 5.1.

[27]Koko SHACL-sanasto on esitetty osoitteessa `https://www.w3.org/ns/shacl.ttl`.

[28]Tämä ei ole käytetyn SHACL-moottorin tukema ominaisuus, minkä takia sitä ei ole luetteloitu taulukossa 5.1.

[29]Apache Jena RIOT (RDF I/O Technology).

```
1  {
2  "from": "turtle",
3  "data":
4    "@base <http://seco.cs.aalto.fi/roteva/> .
5     @prefix sh: <http://www.w3.org/ns/shacl#>.
6
7     <Muoto> sh:targetNode <oikein>, <väärin> ;
8             sh:property [
9               sh:path <vainIRIsuhde> ;
10              sh:nodeKind sh:IRI ;
11            ] .
12    <väärin> <vainIRIsuhde> \"kielletty suhde\" .
13    <oikein> <vainIRIsuhde> <väärin> .
14 }
```

Listaus 5.1: ROTEVAlle annettava JSON-koodattu esimerkkisyöte. Rivinvaihdot (”\n”) on lukemisen helpottamiseksi korvattu data-kentässä aidoilla rivinvaihdoilla.

vaikutukseen saman sovelluksen sisällä, vastaten näin mahdollisimman hyvin työn tavoitteisiin.

Tämän aliluvun rakenne on seuraavanlainen: aliluvussa 5.4.1 käydään läpi käyttöliittymän päänäkymä sekä sen tekninen ja arkkitehtoninen rakenne. Aineiston lataaminen, esittäminen ja käsittely järjestelmässä esitetään aliluvussa 5.4.2, jota seuraa prosessin seuraava tekninen vaihe eli rajoitteiden ohjelmoinnin avustettu muodostaminen ja sovitus dataan aliluvussa 5.4.3. Aliluvussa 5.4.4 käydään läpi validointi ja validoinnin tuloksena saadun validointiraportin graafinen esittäminen ohjelmassa, minkä jälkeen aliluvussa 5.4.5 opastetaan lopuksi, kuinka ohjelman aikana luotu data tallennetaan ja vastataan iteratiiviseen korjaamiseen liittyviin haasteisiin esittämällä käyttöliittymässä toteutettu muotojen ja datan vuorovaikutus ja käyttö validoinnissa havaittujen virheiden ja niissä osallisina olevien kolmikoiden tai niiden osien välillä, täydentäen näin luvussa 4 esitetyn iteratiivisen validointi- ja korjausprosessin.

5.4.1 Näkymän tuottaminen

ROTEVAn MVC-mallin kuvassa 5.1 esitetty GUI-ohjain[30] vastaa sovelluksen graafisesta käyttöliittymästä. Kuten kuvasta ilmenee, tämä ohjain vastaa mallin ja tästä luodun näkymän tuottamisesta. Lisäksi näkymästä on lähe-

[30]Määritelty tiedostossa *app/controllers/GUIController.java*.

```
{
"@graph" : [{
   "@id" : "_:b0",
   "@type" : "sh:ValidationReport",
   "sh:conforms" : false,
     "sh:result" : {
       "@id" : "_:b1"
   }}, {
   "@id" : "_:b1",
   "@type" : "sh:ValidationResult",
   "sh:focusNode" : {
     "@id" : "väärin"
   },
   "sh:resultMessage" :
"Value does not have node kind sh:IRI",
   "sh:resultPath" : {
     "@id" : "vainIRIsuhde"
   },
   "sh:resultSeverity" : {
     "@id" : "sh:Violation"
   },
   "sh:sourceConstraintComponent" : {
     "@id" : "sh:NodeKindConstraintComponent"
   },
   "sh:sourceShape" : {
     "@id" : "_:b2"
   },
   "sh:value" : "kielletty suhde"
   }],
"@context" : {
   "@base" : "http://seco.cs.aalto.fi/roteva/",
   "sh" : "http://www.w3.org/ns/shacl#"
}}
```

Listaus 5.2: ROTEVAn listauksen 5.1 syötteelle palauttama vastaus, kun syöte on lähetetty ROTEVAn REST-rajapinnan validate-metodille.

tettävissä uusia AJAX-pyyntöjä (Asynchronous JavaScript and XML) [48] HTTP-reitittimelle, edeten näin ohjelman suorituksessa. Ohjaimen tarjoama toiminnallisuus on saavutettavissa avaamalla selaimella[31] ohjelmistokehyksen kuuntelema osoite ja portti[32] polulla */validate&repair* täydennettynä.

Kuvassa 5.3 on esitetty käyttöliittymän validoinnista ja korjaamisesta vastaavan HTML-sivun osa[33,34], kun käyttäjä on validoinut dataa. Kuvan yläosassa on lueteltu datassa määritetyt etuliitteet nimiavaruuksineen. Tämän alla on esitetty niin kutsutussa *Datajoukko-taulukossa* käyttäjän järjestelmään syöttämä data, josta on valittu tarkasteltavaksi muoto *ex:ClosedShapeExampleShape* (esitetty kuvassa vaaleanpunaisella taustavärillä). Taulukon alla sijaitseva tyhjä tekstikenttä on tarkoitettu vapaamuotoiseen datan massalataamiseen. Lopuksi kuvassa näkyvä vihreä *Validate*-painike vastaa asynkronisen validointikutsun lähettämisestä validointimoottorille, jonka palauttama validointiraportti on esitetty jäsennettynä kuvan alaosassa. Kuten kuvan validointiraportista nähdään, käyttäjän data ei ole muotojen mukaista, sillä validaattori on raportoinut datan sisältävän seitsemän virhettä ja yhden varoitus-tason raportoitavan asian (esitetty kuvassa vaaleanpunaisella taustavärillä). Niin kutsutussa *Raportti-taulukossa* on esitetty graafisesti sovelluksen havaitsemat virheet, joista *"Zu viele Zeichen"@de*-huomion tuottama raportoitava asia (esitetty kuvassa harmaalla taustavärillä) on purettu selittäviin tekijöihinsä. Validointiraportin sisältämät raportoitavat asiat on jaoteltu vakavuusasteen ja raportoidun viestin mukaan, millä kuvan tapauksessa pyritään antamaan suhteellisen järkevä yhteenveto datassa havaituista virheistä.

Teknisesti käyttöliittymän sivut koostuvat useista ROTEVAn käyttämistä ohjelmistokirjastoista ja tiedostoista[35], jotka selain tulkitsee käyttäjälle ennalta määriteltyjen sääntöjen puitteissa. Selaimessa ajettavalla dynaamisella JavaScript-komentosarjakielellä on toteutettu edellä kuvattu vuorovaikutteinen sivu, jolla on mahdollista validoida ja korjata käyttäjän syöttämää RDF-dataa. JavaScript-perustaisella jQuery-kirjastolla[36] on tässä tärkeä tehtävä: se vastaa näkymän vuorovaikutuksesta mahdollistaen verkkosivun DOM-puun (Document Object Model) [60] tehokkaan tapahtumakuuntelun ja käsittelyn.

[31] ROTEVAn verkkosovelluksen ainoa virallisesti tukema verkkoselain on Mozilla Firefox versionumerosta 60.0 eteenpäin, mutta sovelluksen tulisi kyllä toimia millä tahansa muullakin ECMAScript 2017 -yhteensopivalla [1] selaimella.

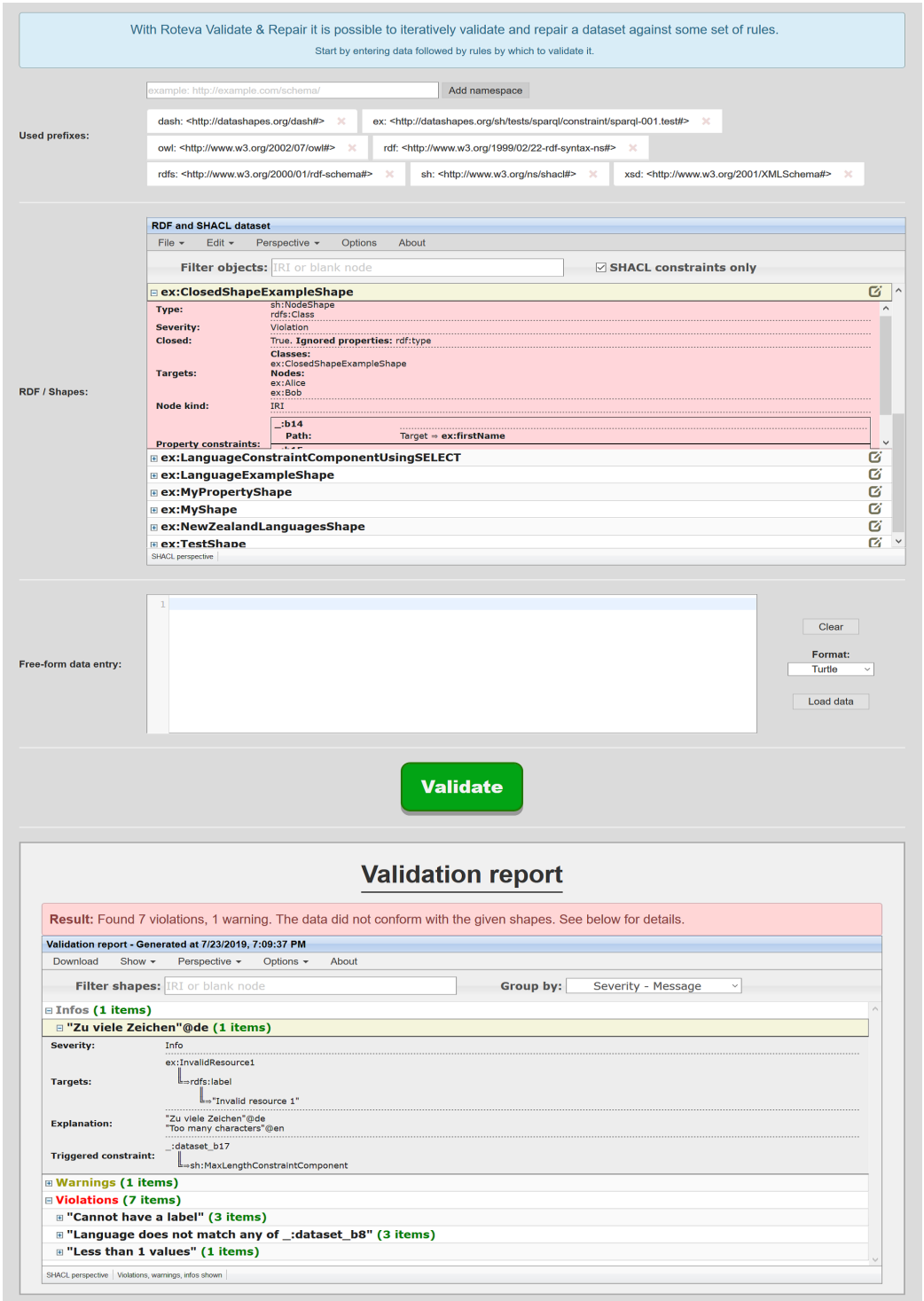
[32] Konfigurointi selitetty aliluvussa 5.2.4.

[33] Kuvan ulkopuolelle jäävät ohjelman nimike, navigaatiopalkki sekä alatunniste.

[34] Muut osuudet ovat etusivu, käyttäjädokumentaatio ja sovelluksen selostesivu.

[35] Tarkka listaus on esitetty tiedostossa *app/views/main.scala.html*.

[36] `https://jquery.com`



Kuva 5.3: Ruutukaappaus ROTEVAn graafisesta käyttöliittymästä.

ROTEVAn toiminnallinen JavaScript-ohjelmakoodi on kirjoitettu tiedostoon *public/javascript/main.js*. Näkymän ulkoasun toteutuksessa on hyödynnetty suosittua Bootstrap-kirjastoa[37,38], jonka lisäksi sivun ulkoasun tarkempi räätälöinti on kirjoitettu esikäännettävään, LESS-muotoiseen[39] (Leaner Style Sheets) *app/assets/stylesheets/main.less*-tiedostoon[40]. Yhdessä nämä kolme selainten yleisesti tukemaa tekniikkaa (HTML, JavaScript & CSS) tuottavat GUI-ohjaimen näkymän[41]. Käyttöliittymä on optimoitu 1280 pikseliä leveille 720p HD -näytöille (high-definition)[42].

5.4.2 Datan syöttö, esittäminen ja manipulaatio

Luvussa 4 esitetyn validointiprosessin ensimmäinen vaihe on aineisto. Aineiston syöttö onkin ensimmäinen vaihe graafisen käyttöliittymän käyttösekvenssissä. Kuvassa 5.4 on esitetty käytössä olevat tavat siirtää suuria datamassoja sovellukseen. Ensimmäinen tapa on ladata tiedosto (sen sisältö) sovellukseen. Tämä tapahtuu valitsemalla kuvassa ylhäällä näkyvän taulukon päävalikosta toiminto *File* → *Open File*, minkä jälkeen käyttäjälle avautuu näkymään uusi ikkuna, jossa pyydetään täyttämään tarvittavat tiedot ladattavasta tiedostosta (tiedostopolku, merkistökoodaus ja sarjallistamismuoto)[43]. Käyttäjän on syytä täydentää nämä parametrit avautuvaan ikkunaan oikein[44], sillä esimerkiksi väärällä merkistökoodauksella luettu tiedosto voi latautua väärin sovellukseen ja aiheuttaa ennalta arvaamattomia ongelmia myöhemmin. Toinen tapa on käyttää kuvan alareunassa näkyvää vapaamuotoisen tekstidatan kenttää ja liittää (tai kirjoittaa) siihen RDF-muotoista dataa jossakin

[37] `https://getbootstrap.com`

[38] Tämän lisäksi ROTEVAn näkymässä käytettävät kirjastokomponentit saattavat käyttää omia tyylimäärittelytiedostojaan.

[39] `http://lesscss.org`

[40] Play Framework -ohjelmistokehys kääntää automaattisesti *app/assets*-hakemistossa sijaitsevat tiedostot, kunhan sopivat lisäosat on asetettu käyttöön ohjeiden [75] mukaan *project/plugins.sbt*-tiedostossa. Tässä tapauksessa LESS-tiedosto käännetään selaimen suoraan ymmärtämäksi standardinmukaiseksi CSS-kieleksi (Cascading Style Sheets) [10].

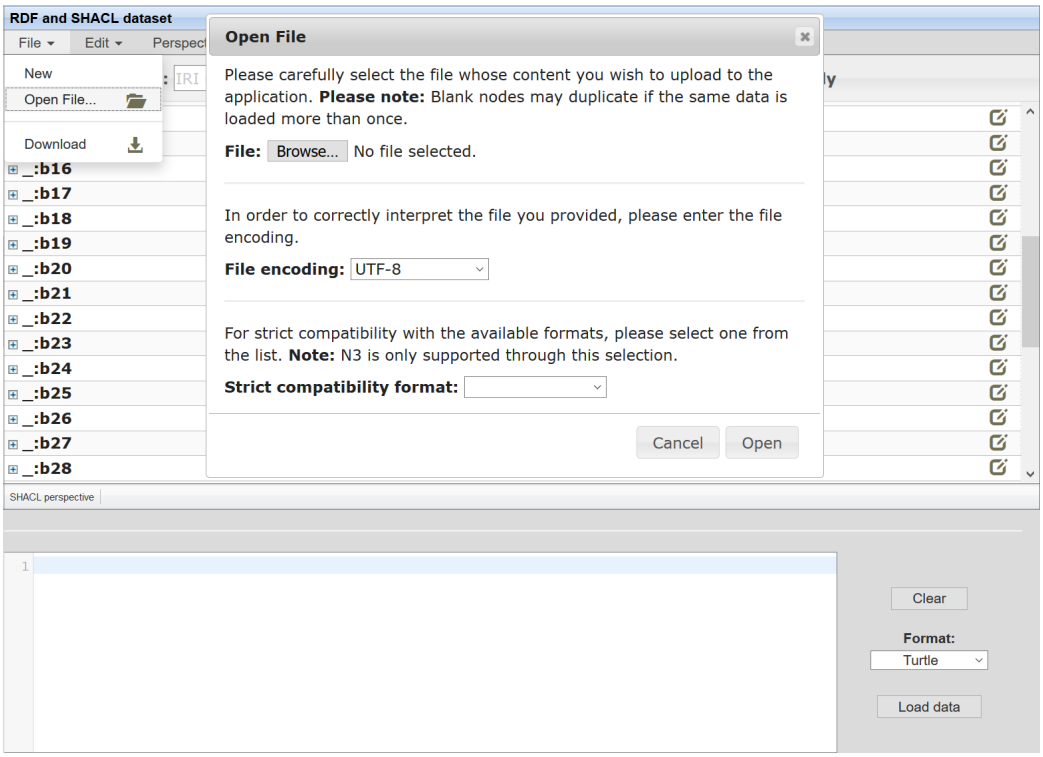
[41] Teknisesti ottaen esimerkiksi pelkistetty tekstiselain voi näyttää pelkän HTTP-pyynnön vastauksena tulevan datan ilman siihen sisältyvien ohjelmaosioiden ajoa, jota tarvitaan vuorovaikutteisuuden toteuttamiseksi. ROTEVAn graafisen käyttöliittymän käyttö vaatiikin tästä syystä dynaamisen JavaScript-ohjelmakoodin suoritusoikeutta selaimessa, minkä voitaneen olettaa olevan oletusarvoista ja hyväksyttävää.

[42] Myös muut 1280 pikseliä leveät näytöt käyvät yhtä hyvin. Erikokoisten näyttöjen kohdalla suositellaan käytettäväksi selaimen lähentämis- tai loitontamistoiminnallisuuksia.

[43] *File*, *File encoding* ja *Strict compatibility format*, vastaavasti.

[44] Jättämällä *Strict compatibility format* tyhjäksi järjestelmä yrittää lukea ”avoimesti” päätellen dataa, mikä ei aina onnistu oikein.

sovelluksen ymmärtämässä sarjallistamismuodossa[45]. Painamalla painiketta *Open* tai *Load data* (ensimmäinen ja toinen tapa, vastaavasti) sovellus aloittaa valitulla menetelmällä syötetyn datan prosessoinnin ja lukemisen sisään. Datan mahdollisesti sisältämät nimiavaruudet etuliitteineen tallentuvat järjestelmän käyttämiin nimiavaruuksiin (esitetty kuvan 5.3 kohdassa *Used prefixes*) uudelleenkäyttöä ja näyttöä varten. On huomioitavaa, että graafinen käyttöliittymä ei tue erittäin suurten tiedostojen tai tekstimassojen lataamista, sillä se heikentää käyttökokemusta[46].



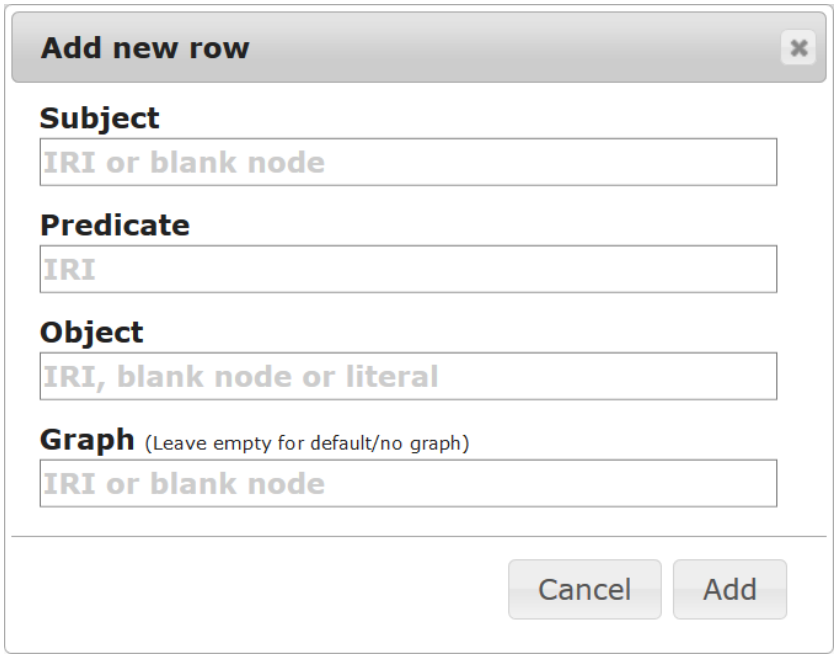
Kuva 5.4: Lähiruutukaappaus datan massalataamisen vaihtoehdoista.

Pienempien datamäärien lisäys on mahdollista joko valitsemalla kuvan 5.4 yläosan Datajoukko-taulukon päävalikosta toiminto *Edit* → *Add a Row*, jolloin käyttäjälle avautuu kuvan 5.5 mukainen yhden kolmikon lisäysikkuna, tai valitsemalla taulukon päävalikosta toiminto *Edit* → *Add a Shape*, jolla voidaan lisätä kokonainen muoto kerralla (esitetty tarkemmin aliluvussa 5.4.3).

[45]Turtle, TriG, N-Triple, N-Quad ja Notation3 [15] ovat sovelluksen tukemat *Format*-alasvetovalikossa määritellyt sarjallistamismuodot.

[46]Selaimen JavaScript-tulkki voi ylikuormittua suuren kuorman alla, jolloin ohjelma ei vastaa.

Järjestelmään tallennettuja nimiavaruuksia on mahdollista käyttää kaikilla pienempien datamäärien syöttötavoilla, mikä mahdollistaa IRI-tunnisteiden lyhennettyjen kirjoitusmuotojen käytön sekä syötettäessä että esitettäessä syötettyä dataa. Ohjelma validoi käyttäjän syötteen ja estää epäkelvon datan syöttämisen järjestelmään. Esimerkiksi kuvan 5.5 syötemenetelmällä tämä tarkoittaa punaisten virheilmoitusten näyttämistä virheellisten kenttien kohdalla, jotta käyttäjä voi helposti ja nopeasti tunnistaa ja korjata virheellisesti syöttämänsä datan.



Kuva 5.5: Yhden kolmikon lisäysikkuna.

Datan esittäminen ja manipulaatio on toteutettu ROTEVAn graafisessa käyttöliittymässä vuorovaikutteisella SlickGrid-taulukolla[47] (edellä esitelty Datajoukko-taulukko), joka on ohjelmakoodissa sovitettu yhteen RDF-kirjasto N3.js:n[48] kanssa. Taulukon päävalikosta on edellä esiteltyjen toimintojen lisäksi mahdollista vaihtaa datan esitysmuodoksi joko perinteinen RDF- tai muotodatan esittämisen suhteen kehittyneempi SHACL-näkymä[49] (valinnat *Perspective* → *RDF* ja *Perspective* → *SHACL*, vastaavasti) sekä taulukon

[47] `https://slickgrid.net`
[48] `https://rdf.js.org/N3.js/`
[49] Esitetty tarkemmin aliluvussa 5.4.3.

yleisasetuksia[50] toiminnolla *Options*. Kuvassa 5.6 esitetyssä RDF-näkymässä käyttäjän on mahdollista tarkastella kaikkia järjestelmään syötettyjä kolmikoita tarkemmin hyödyntäen ohjelmaan rakennettuja selaus-[51], suodatus-[52] ja datan uudelleenjärjestelemisominaisuuksia[53]. Sovelluksessa valittuna oleva rivi[54] on korostettu kellertävällä värillä ja kulloinkin aktiivisena oleva solu on korostettu kiinteällä reunaviivalla. Käyttäjä voi vaihtaa aktiivista solua nuolinäppäimillä (tai hiirellä) ja painamalla syöttönäppäintä (kaksoisnapsauttamalla), jolloin näkymään aukeaa kuvassa näkyvä muokkausikkuna, jossa käyttäjä voi vapaasti muokata aktiivisen solun arvoa. Avautuva ikkuna tarjoaa lisäksi tavan muuttaa kaikkien aineiston kolmikoiden vastaavat[55] arvot samaan, muokattuun arvoon. Tämä tapahtuu asettamalla ruksi kohtaan *Change all references (if applicable)* ennen muutoksen (muutoksien) tallentamista. Valittu rivi voidaan poistaa valitsemalla päävalikosta toiminto *Edit* → *Delete* ja aineiston kaikki kolmikot valitsemalla toiminto *File* → *New*. On huomioitavaa, että tässä näkymässä on suhteellisen helppoa vahingossa rikkoa RDF-tietomallin eheys, sillä mikään ei estä käyttäjää luomasta esimerkiksi päättymätöntä listaa. Tämä on seurausta siitä, että järjestelmän kaikki kolmikot (ja täten myös esimerkiksi kolmikoista rakennettavat listat) ovat atomisia ja toisistaan riippumattomia ”paljaassa” RDF-näkymässä. Liitteessä B on esitetty molempia näkymiä koskevat sekä koko sovelluksen helppokäyttöisyyttä (navigointi, muokkaus) lisäävien ei-standardien pikanäppäimien[56] luettelo.

5.4.3 Rajoitteiden sovittaminen dataan

Kun käyttäjä on syöttänyt datansa aliluvussa 5.4.2 esitetyin tavoin järjestelmään, voidaan muotoja alkaa rakentamaan[57]. Valitsemalla Datajoukko-

[50]Valittavia asetuksia ovat muun muassa taulukossa näytettävät sarakkeet, implisiittisten datatyypitysten käyttö datan esittämisessä sekä resurssien täydellisten IRI-polkujen näyttö näkymässä.

[51]Näkymän alla sijaitsevat sivutukseen liittyvät valinnat.

[52]Kunkin sarakkeen otsikon alla on siihen sarakkeeseen liitetty suodatin.

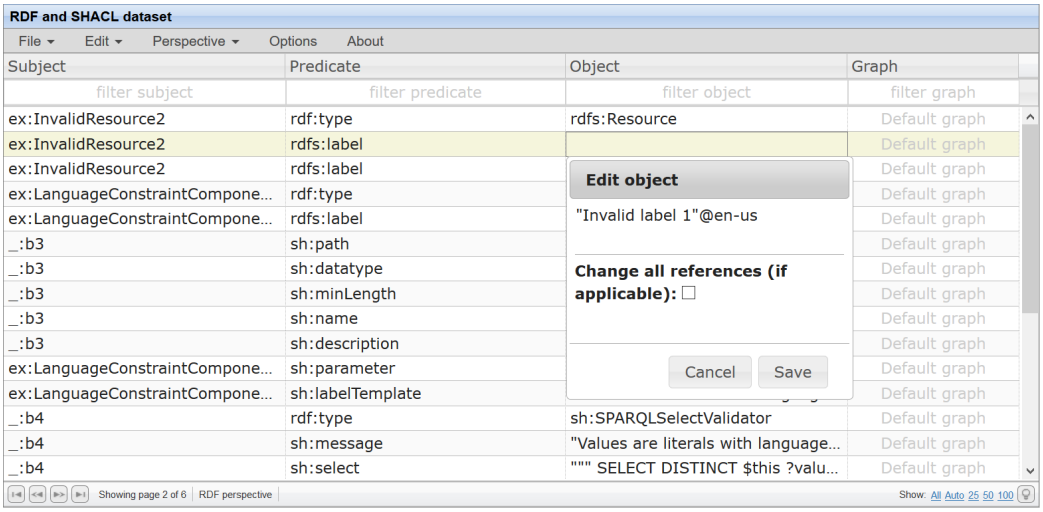
[53]Napsauttamalla sarakkeen otsikkoa kolmikot järjestetään tämän sarakkeen mukaan. Sarakkeita on myös mahdollista venyttää ja niiden paikkaa vaihtaa.

[54]Useita peräkkäisiä rivejä on mahdollista valita painamalla vaihtonäppäintä ja näppäimistöltä nuolinäppäintä ylös (tai alas). Hiirellä tämä onnistuu napsauttamalla ykköspainiketta toisen rivin kohdalla vaihtonäppäimen pohjaan painamisen jälkeen. Hiirellä voidaan myös valita/poisvalita rivejä tarkemmin kontrollinäppäin + ykköspainike -yhdistelmällä.

[55]Vastaava tarkoittaa tässä tapauksessa saman arvon omaavaa IRI-tunnistetta, anonyymiä solmua tai literaalia missä tahansa kolmikon osassa.

[56]Ei-standardi pikanäppäin on sellainen pikakomento, joka ei ole selaimen tai sovelluksessa käytettyjen ohjelmakirjastojen ennalta määrittelemä.

[57]Tarkkaan ottaen muotoja voidaan alkaa rakentamaan ilman muuta aineistoa.



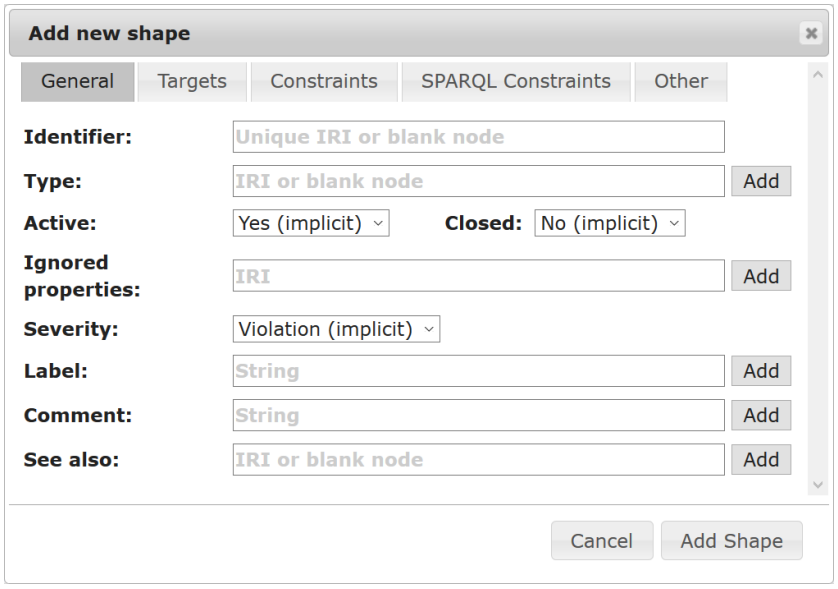
Kuva 5.6: RDF-näkymä esimerkkidatalla sekä näkymän tarjoama datan välittömän muokkaamisen ikkuna.

taulukon päävalikosta toiminto *Edit* → *Add a Shape* avautuu näkyville kuvan 5.7 mukainen ikkuna, jossa pyydetään käyttäjää ensin täyttämään kyseiselle, lisättävälle muodolle yksilöivä tunniste, jota seuraavat kuvan mukaisessa järjestyksessä ylhäältä alaspäin muut yleiset määritteet, nimellisesti tyyppimääritykset, onko muoto aktiivinen (käytössä validoinnissa), suljettu (vain ennalta määritellyt ominaisuudet sallittuja), sulkemisessa huomiotta jätetyt ominaisuudet, vakavuusaste, nimikkeet, kommentit ja katso myös -resurssit. Nämä määritykset muodostavat yhdessä ensimmäisen välilehden (*General*). Seuraavissa välilehdissä sijaitsevat muodon kohdesolmumääritykset (*Targets*), muodon kohdesolmuja koskevat tavanomaiset[58] rajoitemääritykset (*Constraints*), muodon kohdesolmuja koskevat SPARQL-rajoitemääritykset (*SPARQL Constraints*) ja viimeisessä välilehdessä (*Other*) muut näihin edellisiin välilehtiin sopimattomat ominaisuudet.

Kaikki käyttäjän syöttämät ominaisuuksien arvot tarkastetaan SHACL-spesifikaatiossa määriteltyjen maalijoukkojen sallittuja datatyyppejä vastaan ennen muodon hyväksymistä[59]. Lisäksi näkymä huolehtii mahdollisista ominaisuuksien vaatimista RDF-listojen ja anonyymien solmujen tuottamisis-

[58]Tavanomaisella tarkoitetaan tässä kaikkia SHACL-ydinkielen (engl. SHACL Core) rajoitekomponentteja, joita kaikkien SHACL-validaattoritoteutusten on tuettava [63].

[59]Tämä ei kuitenkaan välttämättä tarkoita niiden validoimista parhaalla mahdollisella tavalla; esimerkiksi ominaisuus `http://www.w3.org/ns/shacl#select` vaatii toimiakseen arvokseen syntaktisesti oikein muodostetun SPARQL SELECT -kyselyn, mikä tahansa merkkijono ei kelpaa.



Kuva 5.7: SHACL-muotorajoitteiden avustettu täyttäminen.

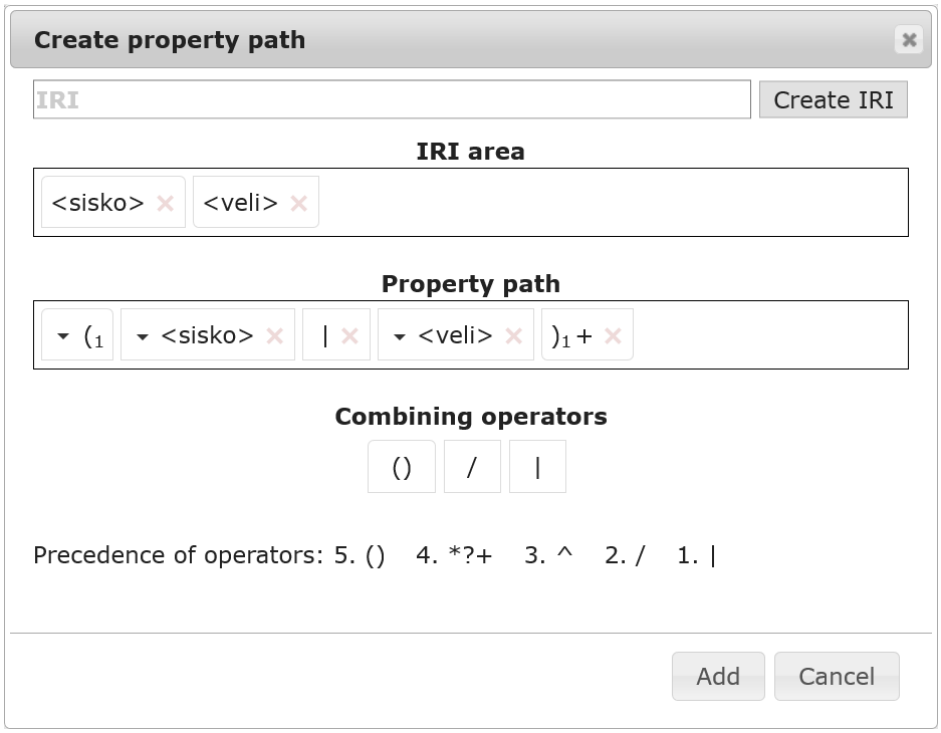
ta; esimerkiksi niin kutsutuissa *ominaisuusrajoitemuodoissa* (engl. property shape) ominaisuuspolun määrittävä *sh:path*-predikaatin[60,61] arvo voi olla liitteen C mukaisesti yksittäinen IRI-tunniste, anonyymi solmu tai tämä voi monimutkaisemman ominaisuuspolun tapauksessa olla lista[62]. Sovelluksen helppokäyttöisyyttä on tästä syystä tämän suhteellisen monimutkaisen predikaatin kohdalla edelleen tehostettu: SHACL-muotoisen ominaisuuspolun muokkaamisen ja luonnin edesauttamiseksi järjestelmään on toteutettu vielä erillinen, kuvassa 5.8 esitetty näkymä, jossa käyttäjän on mahdollista intuitiivisesti rakentaa haluamansa ominaisuuspolku käyttäen hänelle mahdollisesti tutumpaa ja yksinkertaisempaa SPARQL-ominaisuuspolkunotaatiota. Tallennetut polut näytetään myös tällä SPARQL-muotoisella notaatiotavalla, joista on tarvittaessa poistettu käyttäjän syöttämät mahdolliset, tarpeettomat (implisiittiset) sulut.[63]

[60] `http://www.w3.org/ns/shacl#path`

[61] Ominaisuusrajoitemuodon tulee sisältää täsmälleen yksi arvo predikaatille sh:path ollakseen validi SHACL-ilmentymä.

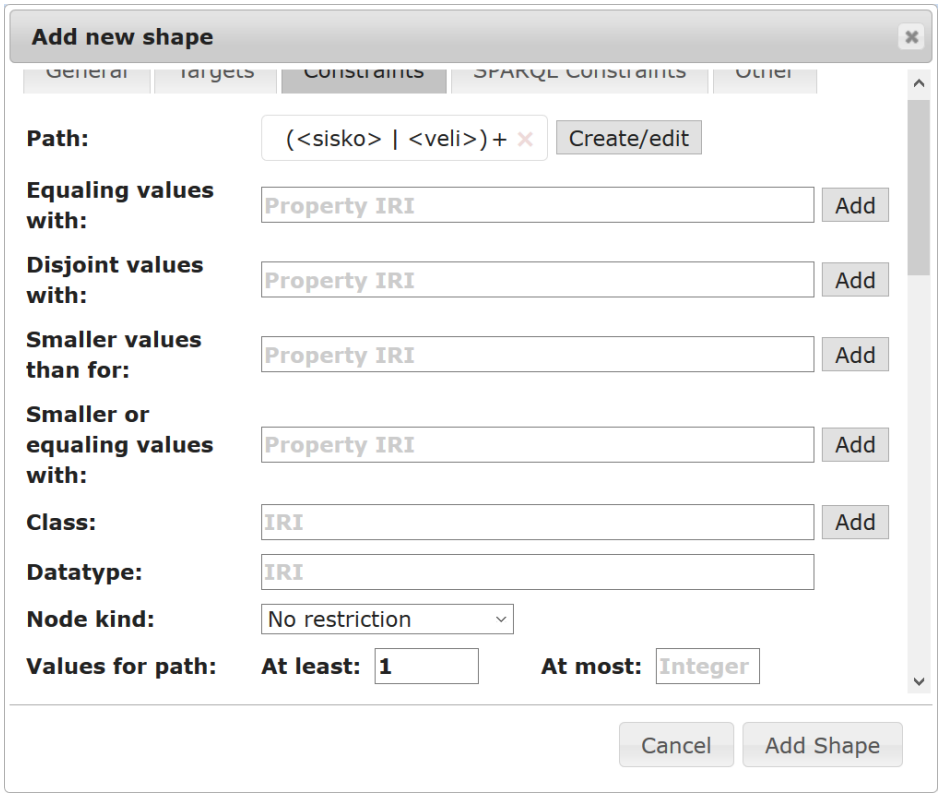
[62] Listankin ensimmäinen solu on yleensä anonyymi solmu, ja sovelluksen käyttämä SHACL-toteutus vaatiikin näiden olevan tällaisia sh:path-predikaatin tapauksessa.

[63] Oheiset toiminnot toimivat myös toiseen suuntaan; esimerkiksi muokatessa muotoa toiminnolla *Edit* → *Edit Shape* tai syötettäessä RDF-dataa järjestelmään ohjelma osaa (SHACL-näkymässä) automaattisesti jäsentää ja näyttää SHACL-muotoiset ominaisuuspolut SPARQL-muotoisina ominaisuuspolkuina.



Kuva 5.8: Näkymä SPARQL-ominaisuuspolun mukaisten SHACL-ominaisuuspolkujen avustetuksi luomiseksi ja muokkaamiseksi.

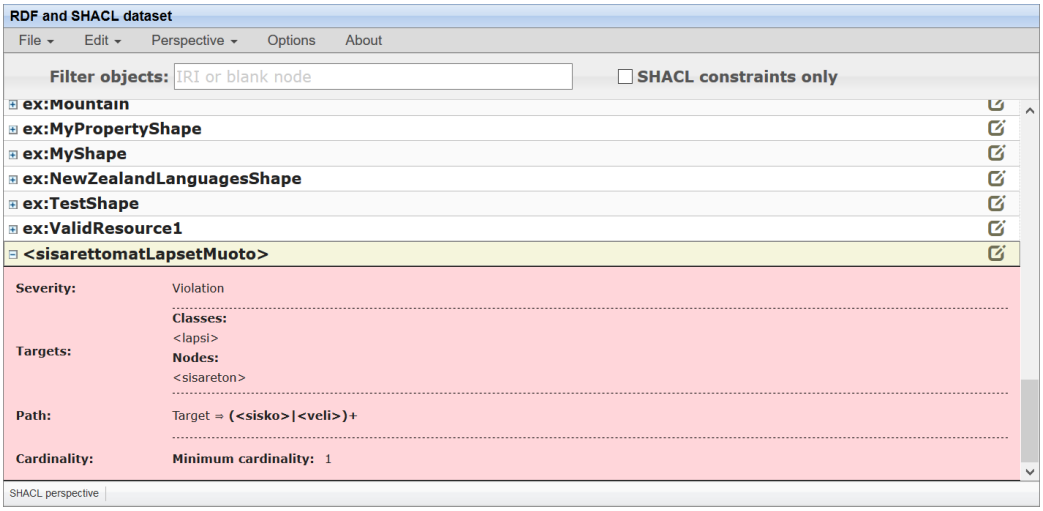
Kuvassa 5.8 on tuotettu yksinkertainen ominaisuuspolku, joka varmistaa, että kohdesolmulla on vähintään yhden kaaren mittainen <sisko>- tai <veli>-polku. Tällaisten ominaisuuspolkujen vähimmäismääräksi on mahdollista asettaa kuvassa 5.9 esitetyllä tavalla yksi, ja jos kohdesolmuiksi on asetettu <sisareton>-yksilö sekä <lapsi>-luokan ilmentymät (ja muodon tunnisteeksi <sisarettomatLapsetMuoto>), on kyseisen muodon RDF-data jo suhteellisen monimutkainen käsittää ja käsitellä kolmikoina RDF-näkymässä (pelkkä sh:path-määritys käsittää tällöin seitsemän kolmikkoa). Tästä syystä järjestelmään on kehitetty kuvassa 5.10 esitetty erityinen SHACL-näkymä muotojen näyttämiseen. Kuvassa on esitetty edellä olevan esimerkin mukaan luodun muodon <sisarettomatLapsetMuoto> tiedot avattuna taulukon alalaidassa. Muodon vakavuusasteeksi (*Severity*) on määräytynyt oletusarvoisesti virhe (Violation). Muodon kohdesolmut (*Targets*) vastaavat edellä muodolle lisättyjä määritelmiä. Ominaisuuspolku (*Path*) sekä ilmentymien sallittu määrä (*Cardinality*) vastaavat kuvassa 5.9 esitettyjä arvoja. Muodon <sisarettomatLapsetMuoto> tiedot on korostettu vaaleanpunaisella värillä, joka ilmaisee



Kuva 5.9: Kuvan 5.8 kultakin kohdesolmulta vaadittavan ominaisuuspolun minimipolkumäärän asettaminen yhteen.

muodon olevan vakavuusasteeltaan virhe (*sh:Violation*[64]). Muita sovelluksessa käytettyjä korostusvärejä ovat keltainen varoitukselle (*sh:Warning*[65]) ja harmaa tiedotuksille (*sh:Info*[66]) sekä muille, ei-muodoiksi laskettaville subjekteille[67]. Näkymässä muoto voidaan avata (sulkea) napsauttamalla hiirellä pluskuvaketta (miinuskuvaketta) ja sitä voidaan alkaa muokkaamaan kynäkuvakkeesta, joka vastaa päävalikon toimintoa *Edit* → *Shape* aktiivisena olevalle muodolle[68]. On huomioitavaa, että muodon tunnistetta ei voida vaihtaa tässä näkymässä – se on tehtävä RDF-näkymän kautta[69].

[64] `http://www.w3.org/ns/shacl#Violation`
[65] `http://www.w3.org/ns/shacl#Warning`
[66] `http://www.w3.org/ns/shacl#Info`
[67] Itse määritellyt muotorajoitteiden vakavuusasteet noudattavat harmaata sävyä.
[68] Näkymässä on lisäksi mahdollista poistaa näkyvistä muut kuin SHACL-rajoitteiksi laskettavat tietueet asettamalla suodatusrivillä ruksi kohtaan *SHACL constraints only*.
[69] Tarkemmin sanottuna muodon muokkausikkuna *Edit* → *Edit Shape* ei mahdollista tunnisteen vaihtamista, eikä SHACL-näkymässä ole muuta tapaa muokata dataa.



Kuva 5.10: SHACL-näkymä esimerkkidatalla sekä (osittain) kuvassa 5.9 määritetty <sisarettomatLapsetMuoto>-tietue avattuna.

5.4.4 Datan validointi ja tulosten esittäminen

Kun käyttäjä on ohjelmoinut rajoitteensa, voi hän käynnistää validoinnin painamalla kuvassa 5.3 näkyvää Validate-painiketta, jolloin ohjelma lähettää käyttäjän syöttämän datan AJAX-pyynnöllä N-Quads-sarjallistamisformaatissa[70] [23] samalla palvelimella tarjottavan REST-rajapinnan validate-metodille[71]. Datan mahdollisesti sisältämät anonyymit solmut skolemoidaan aliluvussa 2.1 kuvatulla tavalla ennen lähetystä siitä syystä, että anonyymejä solmuja ei muuten ole mahdollista luotettavasti tunnistaa ja takaisinmallintaa vastauksesta. Ainoat poikkeukset skolemointiin aiheutuvat *sh:alternativePath*-[72] ja sh:path-predikaattien anonyymeistä, listamuotoisista arvoista, joiden SHACL-spesifikaatio ja täten käytetty SHACL-moottori vaativat olevan aidosti anonyymejä. Vastauksena saadun validointiraportin solmujen skolemointi puretaan, ja anonyymien solmujen tunnisteet muo-

[70]N-Quads-sarjallistamisformaatin etuna muihin formaatteihin verrattuna on se, että se säilyttää datassa käytetyt graafit ja palauttaa tulokset normalisoidussa muodossa (TriG-formaatin mahdollistamat lyhennetyt muodot aiheuttavat pienimuotoisia vaikeuksia raportin vapaamuotoisen sh:resultMessage-kentän mahdollisesti sisältämien IRI-tunnisteiden tehokkaaseen tunnistamiseen).

[71]Jos ohjelmallinen rajapinta halutaan jossain vaiheessa erottaa täysin omaksi komponentikseen ja tarjota toisessa osoitteessa, tulee *public/javascript/main.js*-tiedostossa käytetyn AJAX-pyynnön URL-datalähde vaihtaa.

[72]`http://www.w3.org/ns/shacl#alternativePath`

dostetaan käyttäen kaavaa *_:dataset_$ID*, jossa $ID vastaa alkuperäisessä datajoukossa olleen anonyymin solmun yksilöivää tunnistetta. Tämän jälkeen raportti tulkitaan ohjelman toimesta ja näytetään käyttäjälle kuvassa esitetyllä tavalla. On huomioitavaa, että kohdesolmujen täysin oikeellinen solmutyyppivalidointi (muodoissa mahdollisesti käytetyn *sh:nodeKind*-predikaatin[73] laukaisemana) ei ole mahdollista skolemoinnin aiheuttaman solmutyyppimuunnoksen takia[74].

Kuvassa 5.11 on esitetty lähiruutukaappaus sovelluksen visualisoimasta validointiraportista, kun syötteenä on käytetty listausta 5.1. Kuten kuvasta nähdään, on raportin tulos hylätty (löytyi yksi virhe). Raportti on – samaan tapaan kuin aliluvussa 5.4.2 käsitelty aineistokin – toteutettu vuorovaikutteisella SlickGrid-taulukolla (aliluvussa 5.4.1 mainittu Raportti-taulukko). Tämän taulukon SHACL-näkymä poikkeaa kuitenkin hieman Datajoukko-taulukon SHACL-näkymästä, sillä muotojen sijaan näytettävät oliot ovat raportin tuloksia (raportoitavia asioita). Muita eroavaisuuksia ovat Raportti-taulukon otsikossa sijaitseva raportin luontiaika, mahdollisuus suodattaa tuloksia päävalikon *Show*-alasvetovalikon avulla (esimerkiksi näytä tuloslistauksessa vain sh:Violation-tason virheet) ja SHACL-näkymän tapauksessa vaihtoehtoiset tavat ryhmitellä tuloksia (*Group by* -valinta). Tulokset on kuvassa ryhmitelty validointiraportista saatujen vakavuusaste-, muoto- ja viestitietojen mukaan. Eriasteiset vakavuudet on värikoodattu käyttöliittymässä Datajoukko-taulukon kanssa vastaavasti; vaaleanpunainen väri indikoi virhettä, keltainen varoitusta ja harmaa tiedotusta. Tällöin kaikissa ryhmittelyissä tiedon vakavuuden erottaa välittömästi raporttia selatessa, vaikka eriasteiset virheet sijoittuisivatkin samaan ryhmään[75].

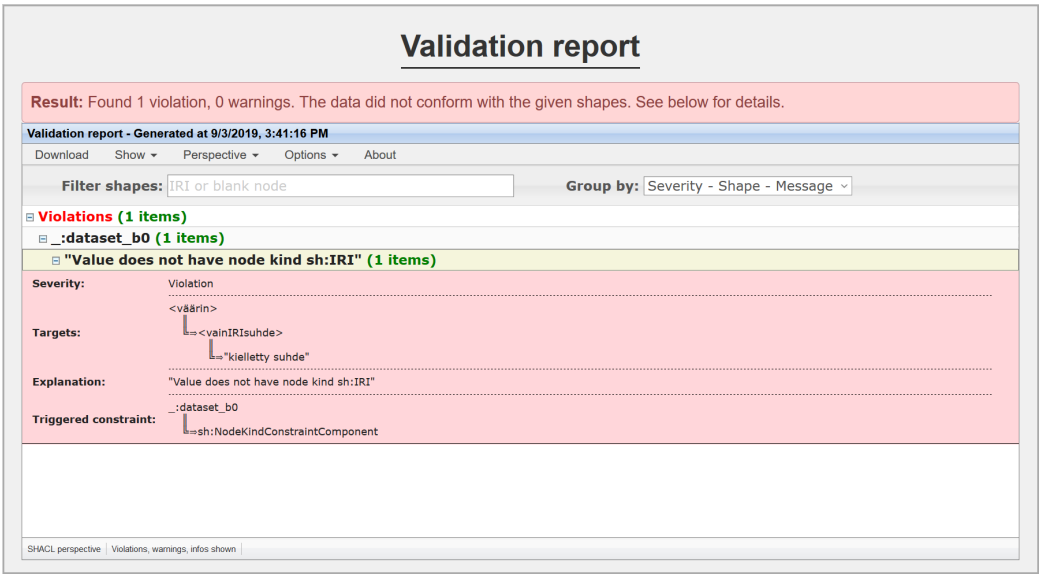
Kuvan 5.11 tuloksia tarkemmin tarkasteltaessa havaitaan, että virhe on kohdesolmun <väärin> (*Targets*, ylin) predikaatin <vainIRIsuhde> (*Targets*, keskimmäinen) literaaliarvossa ”kielletty suhde” (*Targets*, alin). Virheen tekstimuotoinen selitys on kohdassa *Explanation*, joka yleensä[76] tarjoaa vihjeen mahdollisesta ongelman lähteestä datassa. Nämä tiedot yhdistettynä lauenneen rajoitteen (*Triggered constraint*) polkuun tuottavat tarvittavat tiedot ongelman paikallistamiseksi ja korjaamiseksi (korjaaminen esitetty tarkemmin aliluvussa 5.4.5). Tässä tapauksessa tunnisteen _:dataset_b0 saaneella anonyymillä ominaisuusmuotorajoitteella (tämän määrittely alkaa listauksen 5.1 riviltä 8) oleva *sh:NodeKindConstraintComponent*-rajoitekomponent-

[73] `http://www.w3.org/ns/shacl#nodeKind`

[74] Tämä puute koskettaa kuitenkin vain graafista käyttöliittymää, sillä REST-rajapinnan validate-metodi suorittaa validoinnin ilman minkäänlaista skolemointia.

[75] Toki tämä vaatii raportoitavien asioiden avaamista käyttöliittymässä ensin.

[76] On mahdollista, että rajoitteeseen on epäselvästi kirjattu syy rajoitteen laukeamiseen.



Kuva 5.11: ROTEVAn graafisen käyttöliittymän validointiraportti, kun syötteenä on käytetty listausta 5.1. Raportoivat asiat on ryhmitelty asian vakavuusasteen, aiheuttaneen muodon ja viestin mukaan.

ti[77] (määritetty listauksen 5.1 rivillä 10) on lauennut <väärin>-kohdesolmun kohdalla (tämä oli kohdesolmu <Muoto>-rajoitteelle, mutta tästä ei ole enää tietoa raportissa, sillä virheen aiheuttanut *lähdemuoto* [engl. source shape] oli tälle rajoitteelle määritelty ominaisuusrajoitemuoto _:dataset_b0[78]). On huomioitavaa, että kuvan 5.11 anonyymien solmujen tunnisteet eivät vastaa listauksen 5.2 validointiraportin anonyymien solmujen tunnisteita. Kuvan tapauksessa tunniste _:dataset_b0 on kuitenkin kiinnitetty, ja pysyy aina samana eri ajokertojen välillä. Tämä on graafisessa käyttöliittymässä tehdyn skolemoinnin ansiota; käyttäjä voi luottaa _:dataset_b0-tunnisteen vastaavan _:b0-tunnistetta Datajoukko-taulukossa.

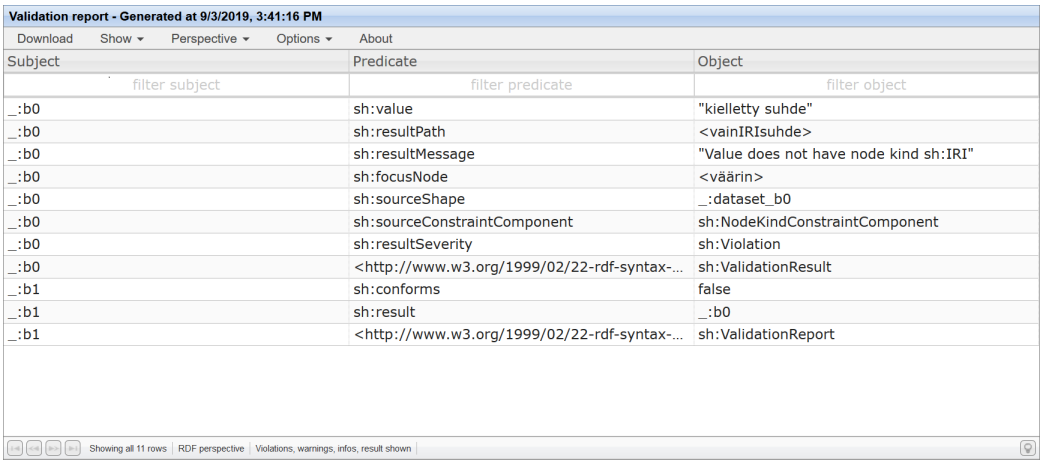
Tehokäyttäjien on mahdollista tutkia dataa myös perinteisessä RDF-näkymässä (esitetty kuvassa 5.12). Tässä näkymässä on mahdollista asettaa näkyville raportin ylin taso[79] (kuvassa _:b1-tunniste) valitsemalla päävalikosta toiminto *Show* → *Result*. Tämä RDF-näkymä toimii samalla tavalla kuin

[77] `http://www.w3.org/ns/shacl#NodeKindConstraintComponent`

[78] <Muoto>-rajoitteen puuttumista validointiraportista voi perustellusti pitää pienenä puutteena sovelluksessa. Asian voisi mahdollisesti korjata sh:detail-predikaattimäärittelyllä, mutta tälle ei tällä hetkellä ole tukea ROTEVAssa.

[79] Ylin taso sisältää tiedon raportin tuloksesta sekä mahdolliset viittaukset rajoitteiden laukaisemiin raportoitaviin asioihin, kuten kuvassa 5.2(a) esitetään.

aliluvussa 5.4.2 esitetty Datajoukko-taulukon RDF-näkymäkin lukuun ottamatta sitä perustavanlaatuista seikkaa, että validointiraportin dataa ei ole mahdollista muokata. On lisäksi huomioitava, että tässä näkymässä nähtävä sh:conforms-predikaatin arvo ei suoraan vastaa validointiraportissa (taulukon yläpuolella, esillä kuvassa 5.11) näytettävää tulosselitettä. Tämä johtuu siitä, että työn tekijän mielestä on loogisempaa ilmoittaa datan olevan rajoitteiden mukaista silloin, kun data ei sisällä yhtäkään sh:Violation-tason virhettä, kun taas SHACL-spesifikaatio vaatii tulosjoukon olevan tyhjä, jolloin edes validoinnin kannalta sinänsä merkityksettömiä sh:Info-tason ilmoituksia ei sallita.



Kuva 5.12: Validointiraportin RDF-näkymä. Huomaa näytettävän raportin ylin taso datassa.

5.4.5 Virheiden korjaaminen ja korjatun aineiston tallentaminen

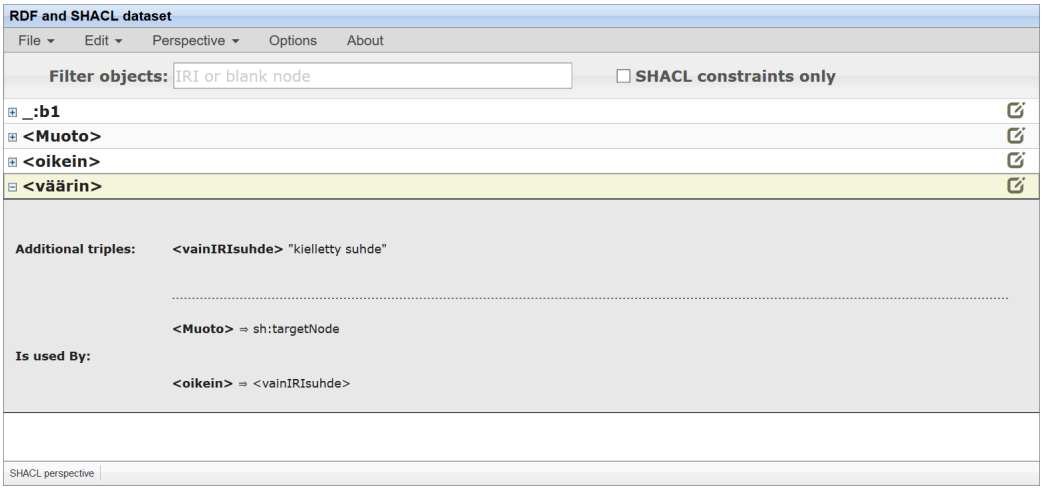
Luvussa 4 esitetty iteratiivinen, yhdistetty validointi- ja korjausprosessi on ROTEVAssa toteutettu seuraavasti: Raportti-taulukossa käyttäjän on mahdollista napsauttaa virheen kohdesolmua, jolloin ohjelma avaa kyseisen solmun Datajoukko-taulukon SHACL-näkymässä ja ohjaa selaimen näyttämään tämän[80]. On myös mahdollista napsauttaa lauennutta rajoitetta tai sen ominaisuuksia – joskus virhe onkin aliluvun 5.4.3 mukaan tuotetussa muodossa, jolloin puolestaan rajoitetta tulee muokata. Datajoukko-taulukossa tehdyt

[80]Tämä siis olettaen, että solmu esiintyy subjektina jossakin aineiston kolmikossa.

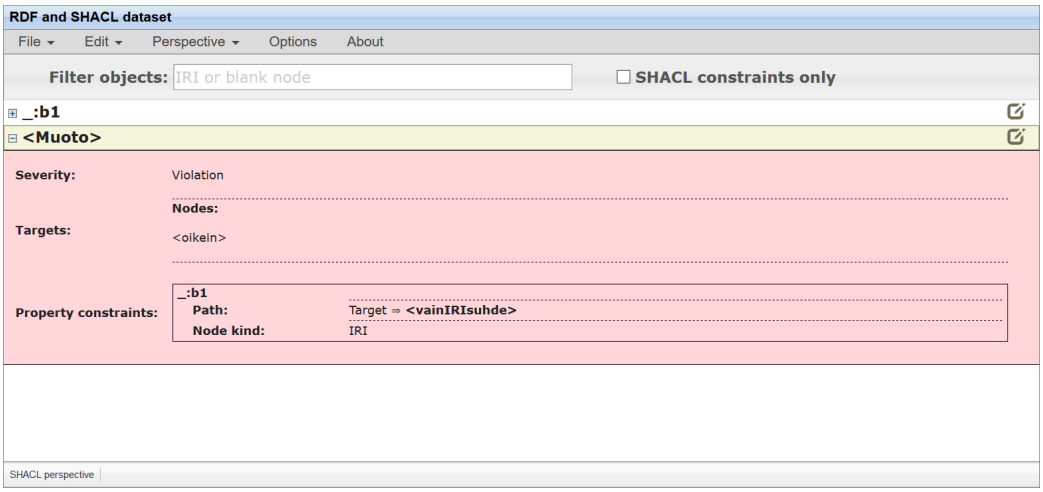
muokkaukset muuttavat Raportti-taulukon otsikon ilmaisemaan sitä seikkaa, että raportti ei ole enää synkronissa (ajan tasalla) Datajoukko-taulukon kanssa. Raportin saa taas synkroniin validoimalla datan uudestaan painamalla aliluvun 5.4.4 ohjeiden mukaisesti Validate-painiketta. Tämän jälkeen käyttäjä voi tarkastella uudestaan raporttia, jossa toivottavasti ei ole enää virheitä jäljellä[81], tai vaihtoehtoisesti, jos tarkoituksena on tämän jälkeen lisätä vielä uusia muotoja, jatkaa tätä työtä.

Kuvassa 5.13 on esitetty kokonaisuudessaan edellä esitelty prosessi käyttäen lähtötilanteena kuvan 5.11 validointiraporttia. Ensin kuvassa 5.13(a) käyttäjä on napsauttanut alkuperäisen kuvan kohdesolmua <väärin>. Ohjelma on ohjannut selaimen näyttämään kyseisen solmun muotonäkymässä (mahdollinen ruksi *SHACL constraints only* -kentästä on poistettu, jotta se on näytettävissä) ja avannut tämän. Käyttäjä voi korjata datan joko suoraan tässä näkymässä (käytännössä poistamalla kyseisen solmun tai muokkaamalla <Muoto>-rajoitetta, johon on mahdollista navigoida esimerkiksi kuvan 5.13 *Is used by* -tiedon kautta tai suoraan Datajoukko-taulukosta) tai vaihtaa RDF-näkymään ja tehdä tarvittavat muutokset siellä vapaamuotoisemmin kolmikkotasolla. Käyttäjä on päättänyt poistaa virheen aiheuttaneen solmun <väärin>. Tämän operaation lopputulos on nähtävissä kuvissa 5.13(b) ja 5.13(c). Näistä ensimmäisessä on esillä poisto-operaation seuraus Datajoukko-taulukossa: sekä <väärin>- että <oikein>-solmut ovat hävinneet näkyvistä. Tämä on seurausta siitä, että <oikein>-solmulla ei ollut muita predikaatteja asetettuina kuin suhde <väärin>-solmuun, joka puolestaan on odotetusti poistunut näkyvistä. Kuvassa 5.13(c) on puolestaan esitetty muutoksen seuraus Raportti-taulukon otsikkopalkissa, johon on lihavoidulla ja punaisella tekstillä lisätty tilailmaus *UNSYNCHRONIZED*, joka ilmentää raportin tietojen olevan vanhentuneita suhteessa Datajoukko-taulukon dataan. Tila saadaan poistettua hyväksymällä tehdyt muutokset edellä kuvatusti eli validoimalla aineisto uudestaan painamalla Validate-painiketta, jolloin ohjelma tuottaa uuden, datan suhteen synkronoidun validointiraportin. Lopputuloksena saadun virheettömän raportin tulostieto on esitetty kuvassa 5.13(d) – uudelleenvalidoinnissa datassa ei ole enää virheitä, sillä <oikein>-kohdesolmulla ei ole enää yhtäkään siltä testattavaa <vainIRISuhde>-predikaattia, joka olisi määritetyn rajoitteen vastainen (_:b1-ominaisuusrajoitemuodossa ei ole määritetty mitään monikertavaatimuksia <Muoto>-rajoitteen kohdesolmuilta vaadittavalle predikaattipolulle, jolloin mikä tahansa predikaattipoluton kohdesolmu läpäisee tämän).

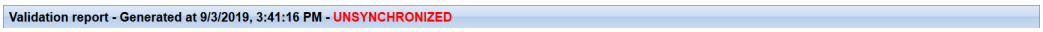
[81] Jos on, prosessia toistetaan kunnes ei enää ole.



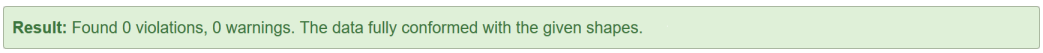
(a) Kuvan 5.11 <väärin>-kohdesolmun napsautuksen jälkeinen näkymä.



(b) Käyttäjän poistettua <väärin>-kohdesolmun myös <oikein>-solmu häviää näkyvistä, sillä se oli subjektina vain <väärin>-solmuun liitettynä. <Muoto>-rajoitteeseen jää jäljelle <oikein>-kohdesolmumääritys.



(c) Käyttäjän poistettua <väärin>-solmun Raportti-taulukon otsikko muuntuu heijastamaan sitä faktaa, että se on vanhentunut.



(d) Uudelleenvalidoidun raportin tulos on vihreä heijastaen sitä, että Datajoukko-taulukon data on validia. Tyhjä, synkronoitu taulukko on jätetty merkityksettömänä kuvasta pois.

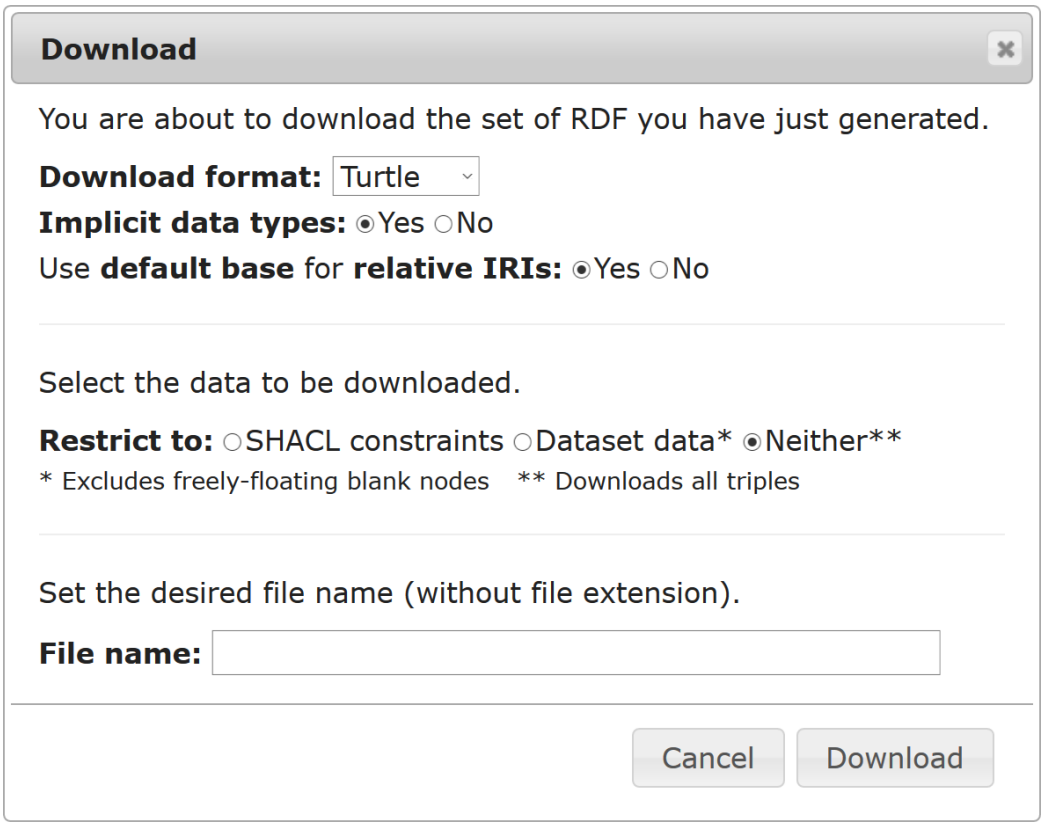
Kuva 5.13: Kuvan 5.11 virheen korjaus epäsynkronivaiheineen.

Muutostietojen pysyvä tallentaminen on toteutettu datan lataamisella (Datajoukko-taulukon päävalikon toiminto *File* → *Download*) ja se on syytä tehdä sen jälkeen, kun käyttäjä on varmistunut datansa oikeellisuudesta validaattorin avulla[82]. Kuvassa 5.14 on dataa ladattaessa esitettävä ikkuna, jossa käyttäjä voi asettaa lataukselleen haluamansa parametrit. Valittavana on muun muassa valittavien kolmikoiden sarjallistamismuoto (*Download format*), käytetäänkö *suhteellisille tunnisteille* (engl. relative IRI) *kantatunnistetta* (engl. default base IRI)[83] (*Use default base for relative IRIs*) vai ei, ladattavan tiedoston nimi (*File name*) ja se, että haluaako käyttäjä ladata kaikki kolmikot (*Restrict to* → *Neither*), pelkän aineiston ilman SHACL-rajoitteita ja kiinnittymättömiä (kelluvia) anonyymejä solmuja (*Restrict to* → *Dataset data*), vai vain pelkät SHACL-rajoitteet ilman aineistoa (*Restrict to* → *SHACL constraints*)[84]. Käyttäjän valittua haluamansa parametrit ja napsautettua näkymässä *Download*-painiketta, tuottaa ohjelma valitusta datasta parametrien mukaisen (ja nimisen) ladattavan tiedoston ja tarjoaa sitä käyttäjälle tallennettavaksi ja täten muissa sovelluksissa edelleenkäytettäväksi.

[82]Validointiraportin voi ladata valitsemalla Raportti-taulukon päävalikosta toiminnon *Download*.

[83]ROTEVAn kantatunniste on <http://seco.cs.aalto.fi/roteva/>, mutta se on helposti vaihdettavissa ohjelmakoodista.

[84]On huomioitavaa, että sarjallistamisformaatit Turtle ja N-Triples eivät tue kolmikoille mahdollisesti asetettuja graafimäärityksiä, jolloin nämä tiedot häviävät lopullisesta tiedostosta käytettäessä näitä formaatteja.



Kuva 5.14: Muotojen ja datan lataus valittavine parametreineen.

Luku 6

ROTEVAn soveltaminen koeaineistoihin

Koeaineistot muodostavat vertailukelpoisen tavan analysoida työn aikana kehitetyn sovelluksen rajoja, soveltuvuutta ja toimintaa. Aineistojen valinta tulee kuitenkin tehdä tarkkaan, jotta saadaan totuudenmukainen kuva järjestelmän suoritus- ja mukautumiskyvystä erilaisissa olosuhteissa. Aineistojen käyttämien sanastojen, metatietoskeemojen ja ontologioiden noudattama tietomalli semanttisine määritelmineen ja *eheysehtoineen* (engl. integrity condition) luo perustan vaadittaville eheysrajoitteille; esimerkiksi SKOS-tietomallissa skos:prefLabel-predikaatille sallitaan enintään yksi arvo subjektia kohden kullekin kielikoodille (SKOS-tietomallin eheysehto S14) [79]. Tällaisten yleisten rajoitteiden lisäksi aineistoilla on usein juuri kyseiseen aineistoon kiinnitettyjä ja sen datalta vaadittavia rajoitteita. Tällaisten tuottaminen vaatii yleensä perehtymistä käytettävään aineistoon.

Tämä luku on kaksijakoinen: aliluvussa 6.1 esitetään SKOS-muotoinen Metatietosanasto ja yritetään muodostaa sille sopivia rajoitteita, jotka voidaan ohjelmoida ja validoida ROTEVAn graafisessa käyttöliittymässä. Ohessa tehdään havaintoja ohjelman suoriutumisesta työn tavoitteisiin nähden (helppokäyttöisyys). Vastaava prosessi tehdään aliluvussa 6.2 toiselle koeaineistolle, nimellisesti toiseen maailmansotaan keskittyvälle Sotasammolle, joka soveltaa edellisestä eriävästi ICOM-järjestön[1] (International Council of Museums) tapahtumaperustaista CIDOC CRM -tietomallia (International Documentation Committee, Conceptual Reference Model) [4] sotatapahtumien kuvailussa, tuoden arvokasta vertailupohjaa sovelluksen ja myös koko SHACL-kielen soveltuvuudesta erilaisia ontologioita hyödyntäville aineistoille.

[1] `https://icom.museum/en/`

6.1 Metatietosanasto

Metatietosanasto[2] (MTS) on Kansalliskirjaston[3] tuottama ja kehittämä sanasto, jonka omassa kuvauksessa [57] todetaan, että

> Metatietosanasto on aineistojen kuvailussa tarvittavien termien ja ilmausten kokonaisuus, joka sisältää sekä ISBD-kuvailusääntöjen että RDA-kuvailuohjeiden mukaista terminologiaa. Sanaston käyttö vakiinnuttaa kuvailevan metatiedon tuottamista ja yhtenäistää kuvailua tiedonhaun tuloksellisuuden edistämiseksi.

Toisaalta kuvauksessa [57] sanotaan myös, että

> Käsitemallin mukaiset kategoriat ja niihin kuuluvat kuvailuelementit ovat Metatietosanastossa ryhmiä, joihin yksi tai useampi käsite kuuluu. Jokainen käsite on sisällytetty johonkin ryhmään.

Yhdessä nämä tiedot antavat vihjeitä mahdollisista aineistokohtaisista rajoitteista. Esimerkiksi olisi mahdollista tarkastaa, että kaikki sanaston käsitteet kuuluvat yhteen tai useampaan ryhmään, tai tutkia onko sanastossa käytetty muuta kuin ISBD-kuvailusääntöjen (International Standard Bibliographic Description) [97] tai RDA-kuvailuohjeiden[4] (Resource Description and Access) mukaista terminologiaa.

Metatietosanaston kokoluokka (~38 000 kolmikkoa) asettaa ROTEVAn heti ensimmäisen haasteen eteen: ROTEVAn nykyversiolla ei kuvan 6.1 mukaisesti ole mahdollista ladata yli 10 000 rivin tiedostoja sille asetetun, ehkä hieman yksinkertaisen (riviperustaisille sarjallistamismuodoille tarkoitetun) tiedostokokorajoitteen vuoksi[5]. Ongelma on kuitenkin kierrättävissä esimerkiksi Turtle-muodossa poistamalla ylimääräiset, datan ihmisluettavuutta selkiyttävät kolmikoiden väliset rivinvaihdot lähtötiedostosta, kuten kuva 6.2 osoittaa. Tämän jälkeen havaitaan graafisen käyttöliittymän käytettävyyteen liittyvä ongelma, joka ilmenee siten, että monet taulukkonäkymää muuttavat SHACL-näkymän operaatiot tapahtuvat käyttökokemusta haittaavalla viiveellä. On ilmeistä, että nämä toiminnot tarvitsisivat sovelluksessa lisäoptimointia näin suurelle aineistolle, mikä on tosin työn muiden osioiden suuritöisyyksien takia päätetty jättää työn laajuuden ulkopuolelle[6].

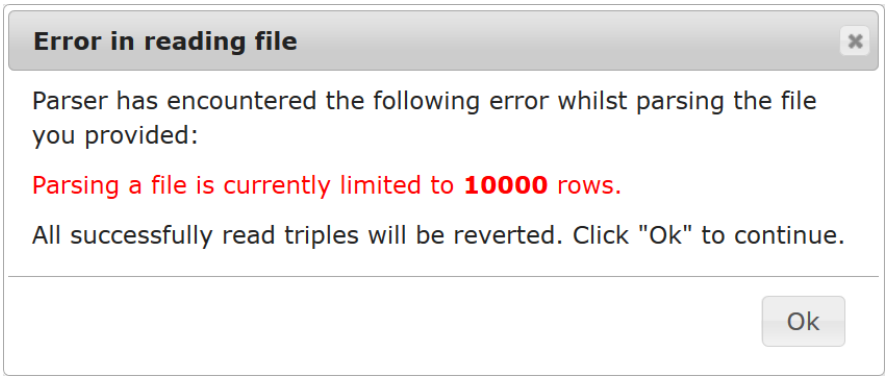
[2] Ladattavissa ja selailtavissa osoitteessa `http://finto.fi/mts/fi/`.

[3] `https://www.kansalliskirjasto.fi`

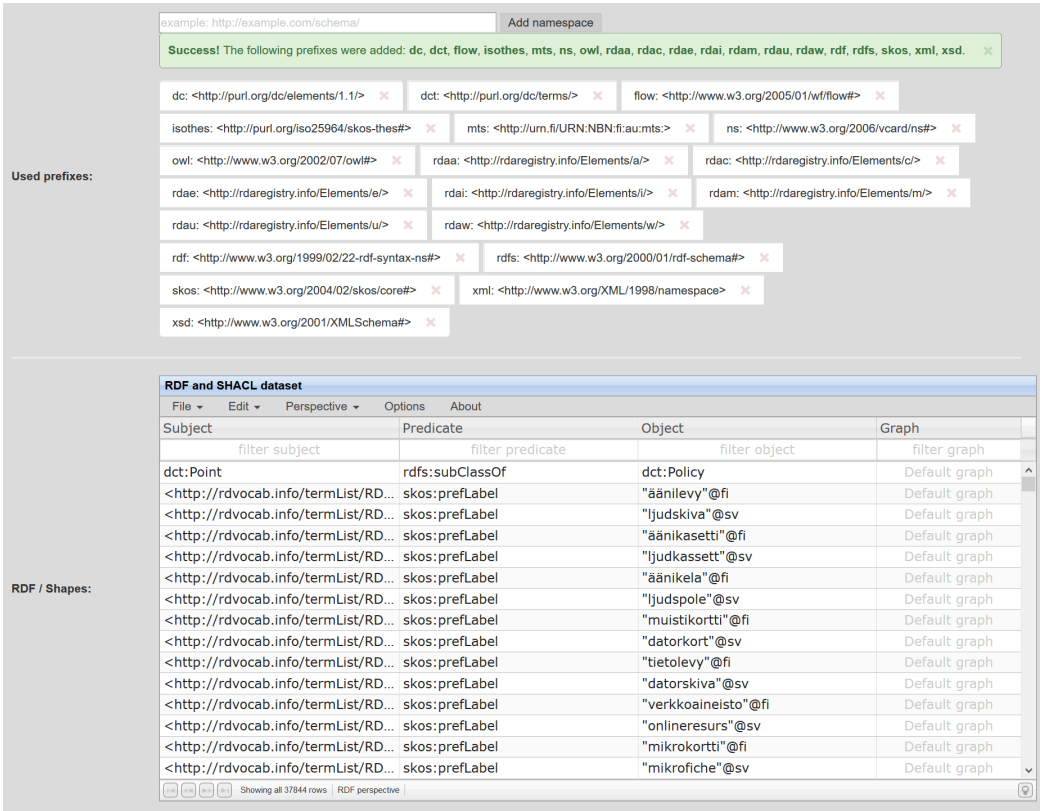
[4] Julkaistu osoitteessa `http://access.rdatoolkit.org` (vaatii kirjautumisen).

[5] Tämä rajoite oli aiemmin asetettu aliluvussa 5.4.2 esitetyistä syistä.

[6] Myös tiedostokokorajoite tulisi samalla muotoilla uudestaan kolmikkoperustaiseksi.

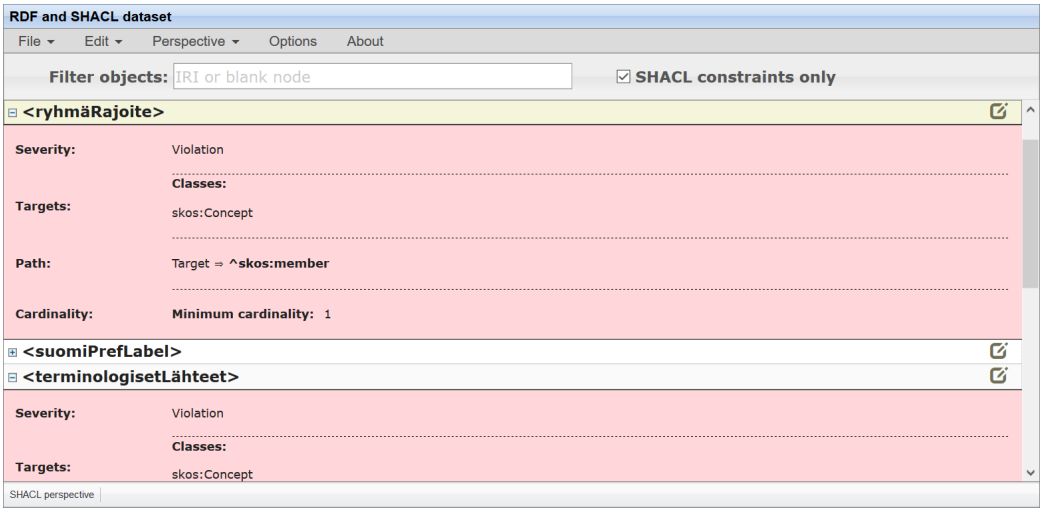


Kuva 6.1: ROTEVA-sovelluksessa tiedoston koolle on asetettu 10 000 rivin rajoite.



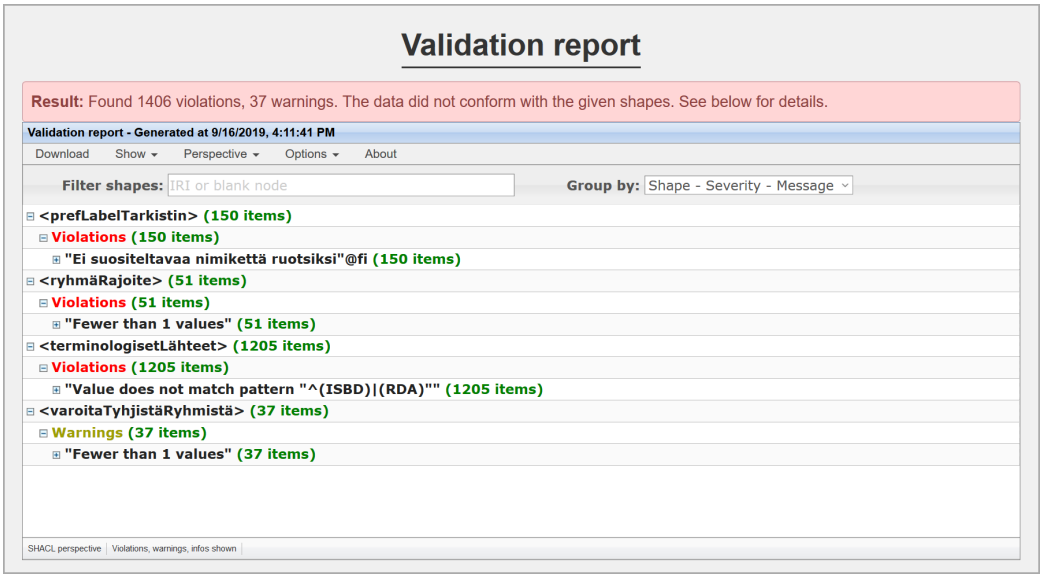
Kuva 6.2: Koko Metatietosanasto saadaan onnistuneesti ladattua ROTEVA-sovellukseen pienellä lähtöaineiston muokkausoperaatiolla.

Alun vaikeuksista huolimatta käyttäjän on mahdollista työn tavoitteen mukaisesti tuottaa sovelluksella haluamansa rajoitteet sekä tarvittaessa muokata niitä. Kuvassa 6.3 on esitetty aiemmin Metatietosanastolle ehdotettu käsitteen ryhmään kuulumisen vaativa rajoite. Lisäksi sanastolle on käyttöliittymän avulla toteutettu useampi kuvassa näkyvä rajoite, joista esimerkkeinä mainittakoon <terminologisetLähteet> (vastaa muiden kuin ISBD-kuvailusääntöjen tai RDA-kuvailuohjeiden mukaisten käsitteiden raportoimisesta), <suomiPrefLabel> (jokaiselle käsitteelle ja kokoelmalle on olemassa suomenkielinen suositeltava nimike), <uniikitArvotPrefLabel> (SKOS-tietomallin eheysehto S14) sekä <varoitaTyhjistäRyhmistä> (tulosta varoitus kokoelmista, jotka ovat tyhjiä). Kaikkien tässä työssä kehitettyjen Metatietosanasto-testiaineistoon kohdistuvien muotojen täydet rajoitemääritykset on esitetty Turtle-esitysmuodossa liitteessä D.



Kuva 6.3: Metatietosanaston kuvauksen perusteella muodostettujen eheysehtojen paikkansapitävyyden tarkastamista varten kehitetty <ryhmäRajoite>-muoto, joka tarkastaa jokaisen sanaston käsitteen osalta, että käsite kuuluu johonkin ryhmään.

Kuvassa 6.4 on esitetty ROTEVAn tulostama validointiraportti, kun lähtöaineistona on Metatietosanasto ja rajoitteina liitteen D muodot. Kuten kuvasta nähdään, on raportoituja virheitä varsin paljon (1406 kappaletta), mikä on suuremmilta osin seurausta <terminologisetLähteet>-rajoitteesta (1205 raportoitavaa asiaa). Näitä lukuja vertaillessa on suhteellisen ilmeistä, että koeaineistossa on yleisesti käytetty myös muita kuin ISBD-kuvailusääntöjen tai RDA-kuvailuohjeiden mukaista terminologiaa. Raportin tarkempi tarkas-



Kuva 6.4: Metatietosanastosta löydetyt puutteet tai lisätutkimuksia vaativat raportoidut asiat. Huomaa tuloslistauksessa kuvassa 6.3 esitetty <ryhmäRajoite>-muoto – Metatietosanasto ei ole kuvauksensa mukainen.

telu paljastaakin, että erityisiä YSO-termejä (*dc:source*[7] ”Yso-termi”@fi) on käytetty yleisesti (1107 kertaa) aineistossa ja että tämän osalta rajoitetta olisi tarpeen säätää hieman[8]. Lisäksi validointiraportista selviää, että 150 käsitettä tai ryhmää on ilman ruotsinkielistä suositeltavaa nimikettä (<ruotsiPrefLabel>-muoto, vastaava kuin <suomiPrefLabel>, sillä Metatietosanaston tulisi määritelmänsä mukaisesti olla suomen- ja ruotsinkielinen [57]), 51 käsitettä ei kuulu mihinkään ryhmään Metatietosanaston kuvauksen vastaisesti ja 37 ryhmää on tyhjiä. Toisaalta on positiivista huomata, että datassa ei ole yhtäkään <uniikitArvotPrefLabel>- tai <suomiPrefLabel>-rikettä.

Validointiraportti paljastaa siis jo[9] edellä kehitettyjen muotojen perusteella Metatietosanastossa olevan selviä kehityskohteita. Käytettyjä muotoja olisi myös tarpeen hieman päivittää: esimerkiksi <terminologisetLähteet>-rajoitteen vakavuusasteen voisi perustellusti vaihtaa sh:Info-tasoon edellä havaitun YSO-termien huomioon ottamisen lisäksi. On kuitenkin huomioitava,

[7] `http://purl.org/dc/elements/1.1/source`

[8] On lisäksi huomioitavaa, että myös Raportti-taulukossa tällaiset suuret tulosjoukot heikentävät käyttökokemusta.

[9] On selvää, että tämän raportin tuottamisessa käytetyt, liitteessä D esitetyt muodot eivät vastaa kaikkia mahdollisia Metatietosanaston sisäisiä laatuvaateita.

että SKOS-sanastojen julkaisupalvelu Fintossa[10] julkaistu Metatietosanasto on mitä ilmeisimmin aktiivisen kehityksen alaisena (sen viimeisin muokkauspäivä on tätä kirjoitettaessa[11] 7.9.2019 [57]), jolloin tällaista kehitysversiota vasten tehdyt kelvollistamiset eivät välttämättä anna aivan totuudenmukaista kuvaa sanaston laadusta, sillä voihan olla, että oheiset puutteet ovat jo sanaston kehittäjien tiedossa ja seuraavaksi korjauslistalla.

6.2 Sotasampo: talvi- ja jatkosodan tapahtumat

Marraskuussa 2015 julkaistu Sotasampo on toiseen maailmansotaan liittyvien erinäisten hajautettujen data-aineistojen monimuotoinen avoimen linkitetyn datan julkaisupalvelu. Palvelussa on kaksi osaa: sovelluskehittäjille suunnattu Sotasampo-datapalvelu[12] sekä loppukäyttäjille suunnattu, edellisen osan varaan rakennettu semanttinen Sotasampo.fi-portaali[13]. Sotasammossa käytetyt laajat, toisiinsa kytketyt tietoaineistot on kuvattu CIDOC CRM -standardin mukaisina tapahtumina, joka sotatapahtumien lisäksi soveltuu erinomaisesti myös muiden Sotasammon sisältötyyppien kuten sotapäiväkirjojen, kuvien ja lehtiartikkelien kuvaamiseen. [56]

Kuvassa 6.5 on esitetty Sotasammossa käytetty sotatapahtumien malli. Kuten kuvasta nähdään, on tapahtuma (*wsc:Event*) yhdistävä tekijä eri ontologioiden (tietoaineistojen) välillä CIDOC CRM -mallin mukaisesti. Tapahtumilla on useita mahdollisia ominaisuuksia, joiden kuvaamiseen käytetään CIDOC CRM:n ohella SKOS- ja Dublin Core -tietomalleja. Aliluvussa 6.1 esitellyn Metatietosanaston käsittely osoitti ROTEVAssa puutteita, minkä vuoksi tässä aliluvussa keskitytään Sotasammon kaikkien tietoaineistojen sijaan pelkästään talvi- ja jatkosodan tapahtumagraafiin[14], jonka koko on suhteellisen maltilliset ~20 000 kolmikkoa. Tämän kokoisen tiedoston ROTEVA-sovellus pystyy suhteellisen helposti käsittelemään[15] ja ohjelman toimintojen

[10] `http://finto.fi`

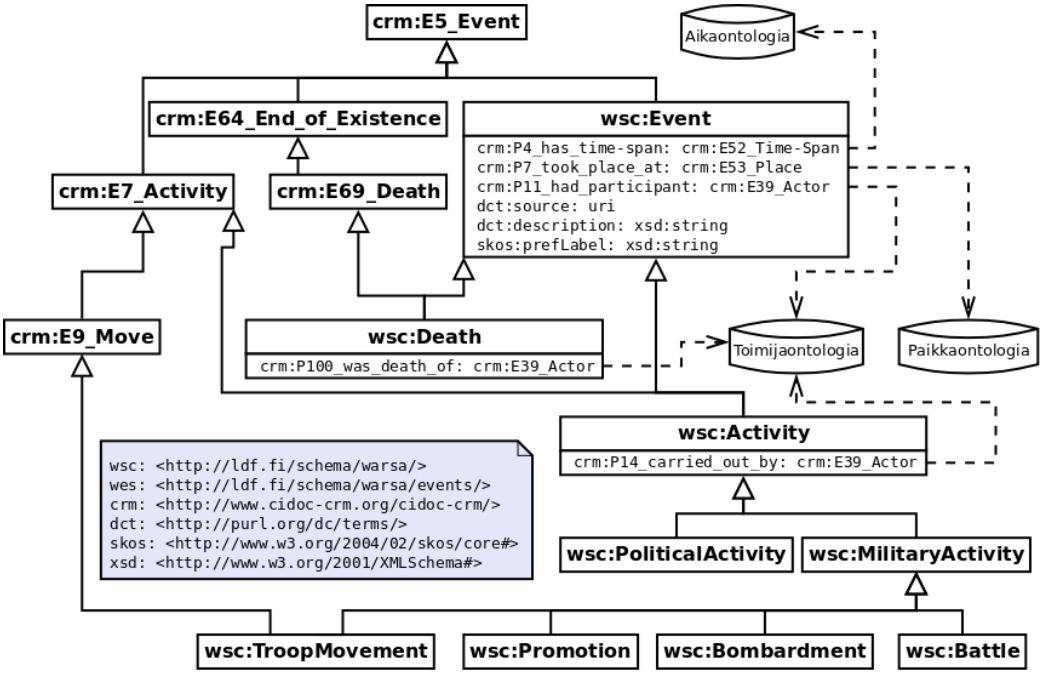
[11] 17.9.2019

[12] Julkaistu SeCo-tutkimusryhmän koordinoimassa *elävän laboratorion* (engl. living laboratory) Linked Data Finland -palvelussa. Data-aineisto kuvauksineen on saatavilla osoitteessa `http://www.ldf.fi/dataset/warsa` ja aineiston koneluettava SPARQL-palvelupiste on käytettävissä osoitteessa `http://ldf.fi/warsa/sparql`.

[13] Julkaistu ja saavutettavissa osoitteessa `https://www.sotasampo.fi`.

[14] Graafi on ladattavissa osoitteesta `http://ldf.fi/warsa/data?graph=http://ldf.fi/warsa/events`.

[15] Koko tiedoston syöttäminen sovellukseen vaatii toki 10 000 rivin kokorajoitteen vuoksi samanlaisen käsittelyn kuin Metatietosanastokin.



Kuva 6.5: Heinon pro gradu -tutkielmassaan kuvaama, Sotasammossa käytetty CIDOC CRM:ää soveltava ja laajentava sotatapahtumien malli [52].

suorittamisessa esiintyvät viiveet paljastuvatkin huomattavasti Metatietosanaston vastaavia pienemmiksi. On oletettavaa, että alkuperäinen 10 000 rivin (kolmikon) rajoite tuottaisi sulavasti toimivan käyttäjäkokemuksen.

Tutkimalla lähtöaineistoksi valittua talvi- ja jatkosotagraafia sekä kuvassa 6.5 esitettyä aineiston tapahtumamallia (josta tosin valittu graafi ei kata kuin pienen osan) päädyttiin ohjelmalla tuottamaan liitteessä E esitetyt rajoitemuodot. Luokan *ws:Event*[16] ilmentymille asetettiin yksilölliset kielitunnukset varmistava rajoite <prefLabelTarkistin> (SKOS-tietomallin eheysehto S14), käyttäjää informoidaan data-aineistosta havaituista muista kuin YSA-tunnisteisista[17] (Yleinen suomalainen asiasanasto) [58] *skos:related*-ominaisuuden[18] arvoista, Suomen sodissa käyttämien lentokonetyyppien numerointi tarkastetaan <lentokoneMuoto>-muodon avulla vastaamaan sovittua konventiota[19], <alkuEnnenLoppua>-rajoite varmistaa, ettei taistelun alkuhet-

[16] `http://ldf.fi/schema/warsa/Event`

[17] Kaikki YSA-tunnisteet alkavat merkkijonolla `http://www.yso.fi/onto/ysa/`.

[18] `http://www.w3.org/2004/02/skos/core#related`

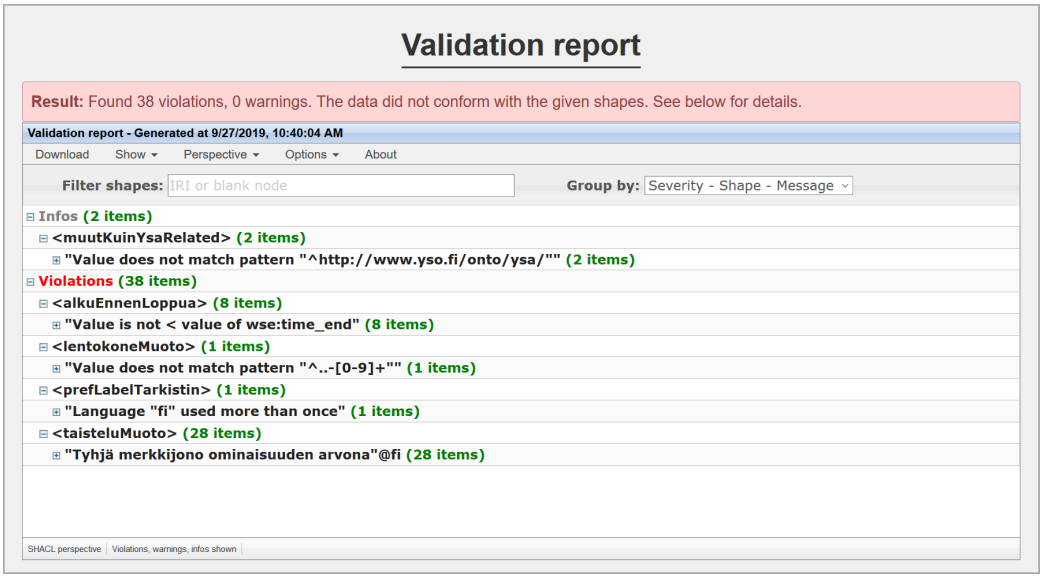
[19] Vuodesta 1927 on konetyypit Suomessa ilmaistu kirjainparilla ja juoksevalla numeroinnilla [109].

ki (wse:time_start[20]) ole tapahtunut ennen loppuhetkeä (wse:time_end[21]) käyttäen SHACL-kielen mukaista pienempi kuin *sh:lessThan*-vertailuoperaattoria[22], joka toimii suoraan *xsd:time*-tyypitetyille[23] literaaleille (näiden tyypitys on vielä erikseen varmistettu <aikaMuoto>-rajoitteessa) ja lopuksi <taisteluMuoto>-rajoite toimii kohdistavana ja aloittavana muotona taistelutapahtumille, nimellisesti aiemmin mainitulle <lentokoneMuoto>-rajoitteelle sekä <tyhjäMerkkijonoRajoite>- ja <toteuttajaEiOllutOsanottaja>-SPARQL-rajoitekomponenteille, joista ensimmäinen varmistaa, että kohdesolmulla ei ole tyhjää literaaliarvoa minkään sen ominaisuuden arvona ja jälkimmäinen eheysrajoitteen, jonka mukaan taistelun toteuttaja on myös sen osanottajien joukossa.

On valitettavaa, että monia kuvassa 6.5 esitettyjä mielenkiintoisia väitteitä ja suhteita (esimerkiksi erilaiset arvojoukkorajoitteet toimija- ja aikaontologioihin) ei kyetty varmentamaan lähtöaineistograafien puuttuessa, minkä takia näille ei nähty tässä vaiheessa syytä rakentaa vastaavia muotojakaan. Toisekseen, <toteuttajaEiOllutOsanottaja>-rajoitetta muodostaessa havaittiin, että ROTEVAn käyttöliittymässä ei vielä ole avustettua tukea SPARQL-rajoitekomponentin *sh:prefixes*-ominaisuudelle[24], minkä vuoksi tämä tulee itse kirjoittaa (tai vaihtoehtoisesti käyttäjän tulee käyttää vain absoluuttisia IRI-tunnisteita SPARQL-kyselylausekkeissa).

Kuvassa 6.6 on esitetty ROTEVAn tulostama validointiraportti, kun aineistona on Sotasammon talvi- ja jatkosotagraafi ja rajoitteina liitteen E muodot. Kuten kuvasta nähdään, on data suhteellisen kelpoista (38 raportoitua virhettä). Aineiston virheistä on kuvassa esitetty ensimmäisenä aakkostuksen mukaisesti muodon <alkuEnnenLoppua>-rikkeet (8 kappaletta), jotka avattaessa paljastuvat välittömiksi tapahtumiksi: ominaisuudet wse:time_start ja wse:time_end on näissä määritelty tapahtuvan samalla hetkellä, jolloin SHACL-moottori tulkitsee ne muodon sh:lessThan-rajoitekomponentin laukaisemana virheiksi. Kyseiset tapahtumat eivät välttämättä kuitenkaan ole aidosti virheellisiä, sillä tällainen tapa mallintaa tapahtumia voi olla perusteltu esimerkiksi julistusten (kaikki virheinä raportoidut tapahtumat ovat ilmavoittoja) tapauksessa, joiden voidaan ajatella olevan välittömiä. Verrattaessa raportoituja ilmavoittoja muihin datassa oleviin ilmavoittoihin voidaan kuitenkin päätellä, että näiden kahdeksan kohdalla kyse on poikkeuksista datassa, sillä muissa ilmavoitoissa on eri alku- ja loppuajanjaksot. Teknisesti olisi

[20] `http://ldf.fi/schema/warsa/events/time_start`
[21] `http://ldf.fi/schema/warsa/events/time_end`
[22] `http://www.w3.org/ns/shacl#lessThan`
[23] `http://www.w3.org/2001/XMLSchema#time`
[24] `http://www.w3.org/ns/shacl#prefixes`



Kuva 6.6: Sotasampo-koeaineistosta löydetyt puutteet tai lisätutkimuksia vaativat raportoidut asiat.

toki mahdollista, että sotatapahtuma olisi kestänyt tasan yhden tai useamman päivän, mutta datagraafia tarkasteltaessa havaitaan, että kyseisillä tapahtumilla on niiden ajanjaksosta kertovan *crm:P4_has_time_span*-predikaatin[25] arvosta jäsennettävissä sama päivä. Yhden lentokoneen tyyppi ei ole <lentokoneMuoto>-rajoitteessa vaadittua muotoa: arvo ”f66119ee-967a-46c2-9ad2-d7f37b3afc2f” viittaisi mahdollisesti anonyymiin solmuun (sen nimeen) eli lähtöaineistossa olevaan anomaliaan (poikkeukseen). Samaten yhdessä tapahtumassa (*we:event_1025*[26]) on englanninkielinen skos:prefLabel asetettu suomenkieliseen arvoon jostakin syystä, jolloin S14-eheysehtoa valvova <prefLabelTarkistin>-rajoite on lauennut. Lopulta on havaittu 28 <taisteluMuoto>-rajoitteen rikettä, jotka kaikki tosin viittaavat <tyhjäMerkkijonoRajoite>-komponentin rikkeisiin – yhtäkään <toteuttajaEiOllutOsanottaja>-rikettä ei havaittu. Edellä esiteltyjen virheiden lisäksi datasta paljastuu muodon <muutKuinYsaRelated> sh:Info-tason raportoitavina asioina kaksi muista poikkeavaa ei-YSA-tunnisteista skos:related-suhdetta DBpedian entiteetteihin, molemmat samasta *we:event_94*-tapahtumasta[27].

[25] `http://www.cidoc-crm.org/cidoc-crm/P4_has_time-span`
[26] `http://ldf.fi/warsa/events/event_1025`
[27] `http://ldf.fi/warsa/events/event_94`

Luku 7

Tulosten arviointi

Valmiin sovelluksen testaaminen luonnollisia koeaineistoja vastaan paljastaa sen soveltuvuuden reaalimaailman kohteisiin. Testaaminen vaatii onnistuakseen kuitenkin objektiivisia (sekä kvantitatiivisia että kvalitatiivisia) mittareita. Luvussa 2.2 esiteltiin tällaiset nykykirjallisuuden mukaiset, tietoaineistojen arvioinneissa käytetyt laatu-ulottuvuudet. Toisaalta testauksessa käytettyjen data-aineistojen tulisi olla yhteismitallisia eli käytännössä vakioituja. Tässä työssä tavoitteena oli kuitenkin aidosti löytää uusia epäjohdonmukaisuuksia jo verkossa julkaistuista ja aktiivisessa käytössä olevista aineistoista. Osa kirjallisuudessa käytetyistä laatu-ulottuvuuksista (esimerkiksi luottamus, dynaamisuus ja tavoitettavuus) osoittautui lisäksi vaikeiksi, ellei mahdottomaksi kirjoittajan formalisoida sellaiseen muotoon, jossa näitä olisi voitu luotettavasti raportoida[1]. Näistä syistä tuloksia arvioidaan tässä luvussa vain suhteessa työn tavoitteisiin ja koeaineistoissa havaittuihin puutteisiin nähden.

Tämä luku on jaettu kahtia seuraavasti: aliluvussa 7.1 analysoidaan luvussa 6 esitettyjen koeaineistojen validoinneissa havaittuja virheitä, pohditaan jatkotoimenpiteitä sekä arvioidaan koeaineistojen validoinnin onnistuminen yleisesti. Aliluvussa 7.2 käydään puolestaan läpi koeaineistojen käsittelyssä havaitut käyttöliittymän ominaisuudet ja puutteet. Näitä ominaisuuksia ja puutteita peilataan suhteessa työn tavoitteisiin nähden: miten hyvin työn aikana kehitetty validointimenetelmä soveltuu käytettäväksi, kuinka helppokäyttöinen sovellus on ja onko ROTEVAn laajamittaiselle käytölle esteitä?

[1]Lisäksi validoinnissa käytettävien rajoitteiden kehittäjän tulisi tuntea mallinnettava data-aineisto yhtä hyvin kuin aineiston tekijän, jotta siinä esiintyviin, aineistolle ominaisiin virheisiin voidaan ylipäätään puuttua. Data-aineistoon tutustuminen näin syvällisesti ei luonnollisesti ollut mahdollista kirjottajalle tämän työn laajuus huomioon ottaen.

7.1 Koeaineistoissa havaitut puutteet

Kuten luvussa 6 havaittiin, paljastui jo käytössä olevista ja julkaistuista koeaineistoina käytetyistä data-aineistoista virheitä. Tämä ei luonnollisestikaan ole mitenkään poikkeuksellista ei-kriittisille sovelluksille[2] – tärkeämpää lienee saada data ylipäätään jaettua ja julkaistua, jolloin datassa mahdollisesti piilevät virheet – toki niin kiusallisia kuin ne ovatkin – ovat toissijaisia[3]. Kuitenkin semanttisten tekniikoiden kehittyessä on luonnollista pyrkiä kohti täydellisempää dataa – etenkin kun tässä työssäkin esitetyt menetelmät antavat siihen mahdollisuuden.

Metatietosanastossa (käsitelty aliluvussa 6.1) havaittiin ohjelmalla tuotetuilla muodoilla (esitetty liitteessä D) kuvan 6.4 mukaiset puutteet. Kuten aliluvussa todettiin, kaikki havaitut virheet (esimerkiksi <terminologisetLähteet>-muodon tuottamat sellaiset) eivät aidosti ole virheitä, vaan ne vaatisivat työssä kehitetyn iteratiivisen menetelmän mukaisesti muodon uudelleenohjelmoimisen. Osa puutteista on kuitenkin luonteeltaan vakavampia – esimerkkinä toimii <ryhmäRajoite>-muoto, jonka raportoimat asiat ovat suoraan sanaston kuvauksen[4] vastaisia. Tällaisten virheiden tyyppi vastaa Laition diplomityössään [72] esittelemää ja kuvassa 2.1 esitettyä virheellisen datan laatutyyppiä. Toisaalta taas työssä kehitetty <varoitaTyhjistäRyhmistä>-muoto paljastaa saman laatu-ulottuvuusjaottelun mukaan puutteellisesta syystä epätäydellisen datan virheitä. Näiden virhetyyppien mukaisten virheiden – kuten muidenkaan tässä työssä tuotettujen muotojen tunnistamien virheiden – korjaamista kirjoittajan toimesta ei kuitenkaan nähty mielekkäänä, sillä datalähtöisinä virheinä niiden korjaamisen voidaan ajatella olevan sanaston ylläpitäjän vastuulla. Lisäksi kuten aliluvussa 6.1 mainittiin, on Metatietosanasto mitä ilmeisimmin jatkuvan kehityksen alainen sanasto ja osa virheistä saattaakin olla jo tätä kirjoitettaessa[5] korjattu tai korjauslistalla.

Sotasammon talvi- ja jatkosotien tapahtumagraafista (esitelty aliluvussa 6.2) työssä kehitetyillä muodoilla (rajoitemääritykset esitetty liitteessä E) löydetyt puutteet[6] on esitetty kuvassa 6.6. Kuten raportoitujen asioiden määrästä (38 kappaletta) voidaan päätellä, on aineisto suhteellisen kelpoista, ainakin suhteessa validoinnissa käytettyihin rajoitteisiin. Erityisenä piirteenä tapah-

[2]Kirjoittaja olettaa tässä, että oheiset tietokannat eivät ole elämää ylläpitäviä.

[3]Toki kuitenkin hyödynnettävyyden rajoissa – esimerkiksi väärin mallinnettu data ei välttämättä ole enää loppukäyttäjän kannalta lainkaan hyödyllistä.

[4]Kirjoittaja olettaa tässä, että Metatietosanaston tekstimuotoinen kuvaus (katkelma siitä esitetty aliluvussa 6.1) on yhtäpitävä sen tietomallin skeeman kanssa.

[5]13.10.2019

[6]Oikeammin tiivistetty listaus virheistä.

tumagraafissa voidaan pitää <alkuEnnenLoppua>-muodon virheitä, jotka voidaan kuvan 2.1 mukaisesti tulkita epätarkan datan laatutyyppivirheiksi. Muut virheet vaikuttavat tyypillisiltä yksittäisiltä datan syöttö- tai lukuvaiheessa[7] syntyneiltä datavirheiltä <taisteluMuoto>-virheitä (28 kappaletta) lukuun ottamatta. Nämä virheartefaktit lienevät jonkin systemaattisen prosessin tuottamia. On kuitenkin huomioitavaa, että virheiden vähäinen määrä ei kuitenkaan välttämättä tarkoita datan olevan erityisen kelvollista, sillä tässä validoinnissa ei ole lainkaan mukana sellaisten tähän graafiin kytkeytyneiden objektien validointia, jotka sijaitsevat jossain toisessa graafikontekstissa. Tällainen kattavampi koko graafiperheen validointi ei ollut mahdollista koeaineiston rajoittuessa yhteen datagraafiin. Tämänkään aineiston kohdalla ei – datalähtöisistä virheistä johtuen – kirjoittaja nähnyt mielekkääksi suorittaa jatko-operaatioita aineistolle tai siihen sovelletuille rajoitemäärityksille. Sen sijaan työn aikana havaituista virheistä raportoitiin graafin julkaisseelle taholle, jonka vastuulla aineistoon tehtävät korjaukset ovat.

7.2 Sovelluksen käyttökelpoisuus

Huolimatta luvussa 6 havaituista tehokkuusongelmista sovelluksella on mahdollista työn tavoitteen mukaisesti validoida ja korjata linkitetyn datan tietoaineistoja[8,9], mistä todisteina ovat Metatietosanaston ja Sotasammon tapahtumagraafin (kuvat 6.4 ja 6.6, vastaavasti) validointiraportit. Nämä raportit sisältävät vaaditusti ohjelman koeaineistoille kehitettyjen muotojen perusteella raportoimat asiat. On huomioitavaa, että osa työssä toteutetuista muodoista on suhteettoman monimutkaisia (ne vaativat edistyneemmille käyttäjille suunnattujen SPARQL-rajoitekomponenttien hyödyntämistä), mikä on seurausta valitusta SHACL-rajoitekielestä, jonka ydinkieleen ei sisäsyntyisesti kuulu esimerkiksi mahdollisuutta ilmaista kielikoodillisen arvon puuttumista kohdesolmulta (<ruotsiPrefLabel>- ja <suomiPrefLabel>-rajoitteet). SHACL-kielen eduksi on kuitenkin todettava, että se täyttää ja jopa ylittää työssä valittavalle rajoitekielelle asetetut validointivaatimukset.

ROTEVAn ohjelmallinen rajapinta vastaa työlle asetettuja tavoitteita. Rajapinnan toiminnallisuutta olisi kuitenkin helppo parantaa monella tapaa. Yksi

[7]Ei ROTEVAn sisäisessä, vaan koeaineistoa alun perin tuotettaessa tapahtuneessa tapahtumassa.

[8]Rajoittuen toki hieman pienehkönlaisiin, alle 40 000 kolmikon aineistoihin.

[9]Koeaineistojen korjaamista ei kuitenkaan nähty aliluvussa 7.1 esitetyistä syistä tässä työssä mielekkäänä, vaikka aineiston korjaaminen ja uudelleenvalidoiminen olisi ollut täysin mahdollista aliluvussa 5.4.5 esitetyin menetelmin.

ilmeisimmistä keinoista olisi tukea erillisten muoto- ja datagraafien prosessointia – periaatteessa tämä on jo käyttöliittymän osalta toteutettu aliluvussa 5.4.5 esitetyin datan tallennustavoin (lähetystavoin). Lisäksi työssä käytetyssä SHACL-moottorissa on jo tuki tämänkaltaiselle operaatiolle. Aliluvussa 5.4.4 mainittu skolemoinnin aiheuttama ongelma anonyymien solmujen oikeellisesta solmutyyppivalidoinnista (joka tosin koskee vain käyttöliittymää) on hieman hankalampi korjata; se vaatisi käyttöliittymässä käytetyn takaisinmallinnuksen mahdollistamiseksi erityisen parametrin tai metodin lisäämistä ohjelmalliseen rajapintaan, mikä mahdollistaisi vastaavan skolemointioperaation suorittamisen palvelimella.

Työn aikana toteutetun linkitetyn datan yhdistetyn validointi- ja korjauskäyttöliittymän helppokäyttöisyyttä on – kirjoittajan ilmeisen jääviyden vuoksi – vaikea objektiivisesti tarkastella. Helppokäyttöisyyden arviointi vaatisi perinpohjaisen käyttäjätutkimuksen tekemistä, mutta tästä päätettiin luopua työn ohjelmallisen osion osoittauduttua työn laajuuteen nähden liian suureksi. Siitä huolimatta sovellusta testatessa tuli ilmeiseksi, että esimerkiksi muotojen laatimisessa tulisi olla valmiita (esitäytettyjä) muotoprofiileja erilaisille muototyypeille. Lisäksi sovelluksen täysimittaiseksi tuotteistamiseksi erityisesti automaattisesti täydentyvän tekstinsyötön lisääminen olisi erittäin hyödyllinen ja helppokäyttöisyyttä aidosti lisäävä toimi. Käyttöliittymän tehokkuusongelmat tulisi myös ratkaista optimoimalla toteutusta. Toisaalta on kuitenkin sanottava, että aliluvussa 5.4.3 esitetty tavallisille käyttäjille mahdollisesti tuntemattomien ja vaikeasti muodostettavien SHACL-ominaisuuspolkujen avustettu tuottaminen, muokkaaminen ja näyttäminen helppokäyttöisempinä SPARQL-ominaisuuspolkuina onnistui työssä hyvin. Samaten navigointi sovelluksessa toimii tarkoituksenmukaisesti – ainakin pienen harjoittelun jälkeen. Liitteessä B esitetyt pikanäppäimet mahdollistavat tehokäyttäjille sovelluksen helpomman ja nopeamman käytön. Helppokäyttöisyyden tehostamiseksi muotojen SPARQL-tuki voisi silti olla parempi, kuten myös avustetun täytön avulla tuotettujen muotojen sisäiset käypäisyystarkastukset, sillä tällä hetkellä käyttäjän on mahdollista tuottaa rikkinäisiä muotoja monin eri tavoin, joskus vieläpä ohjelman suorittamisen keskeyttävällä tavalla.

On huomioitavaa, että vaikka koeaineistojen testauksissa ei erikseen mainittu luvussa 4 esitetyn iteratiivisen validointi- ja korjausmenetelmän hyödyntämistä, käytettiin sitä kuitenkin laajalti luonnollisena osana näille muodostettujen muotojen tuottamisessa[10]. Menetelmä havaittiin toimivaksi ja käy-

[10]Tästä syystä koeaineistojen lopullisten validointiraporttien (esitetty kuvissa 6.4 ja 6.6) muodoissa (esitetty liitteissä D ja E, vastaavasti) ei enää ollut tehtävänä mainittavia korjauksia – ne oli jo tehty iteratiivisen testausprosessin aikana.

tettäväksi annetuilla koeaineistoilla tehokkuusongelmien asettamien rajojen sisällä (arviolta noin 30 000 kolmikkoa / 2000 raportoitavaa asiaa) tavallisella kannettavalla tietokoneella[11] käytettäessä Microsoft Windows 10 Enterprise -käyttöjärjestelmää ja Mozilla Firefox -selainta. Suurten aineistojen validointi ei siten tällä hetkellä ole käyttöliittymässä käytännöllistä ennen ohjelmakoodin optimointia, mutta asia olisi mahdollisesti kierrettävissä syyskuussa 2019 julkaistuun Apache Jena 3.13.0 -ohjelmistoversioon [93] sisällytetyn SHACL-tuen avulla, joka yhdessä Apache Jena Fuseki -SPARQL-palvelinohjelman[12] kanssa mahdollistaa palvelupisteessä olevan datan validoinnin [8], jonka kirjoittaja olisi halunnut saada työhön toteutettua, mutta jonka tehokkaaseen toteuttamiseen tarvittavaa ominaisuutta ei vielä ROTEVAn ohjelmakoodin jäädyttämisen aikaan kesäkuussa 2019 ollut saatavilla[13].

[11] HP EliteBook 840 G4 varustettuna Intel i5-7200U @ 2,50 GHz -prosessorilla ja 16 gigatavulla muistia @ 2133 GHz.

[12] `https://jena.apache.org/documentation/fuseki2/`

[13] Toki on huomioitava, että tällainen palvelupistevalidointimenetelmä ei mahdollista sovelluksessa sisäisesti suoritettavaa iteroivaa validointi- ja korjausprosessia, vaikka SPARQL-palvelupisteen dataa olisikin mahdollista päivittää SPARQL UPDATE -kyselyillä.

Luku 8

Yhteenveto ja jatkokehitys

Avoimesti julkaistun linkitetyn datan määrä Internet-verkossa kasvaa ennennäkemätöntä tahtia. Datan onnistunut hyödyntäminen erinäisissä sovelluksissa vaatii kuitenkin julkaistuilta data-aineistoilta odotustenmukaisten laatumääritysten täyttämistä. Tähän tehtävään käytetään validaattoreita, jotka tarkastavat datan käypäisyyden jonkin määritelmän suhteen. Havaitut virheet tulee lisäksi voida korjata. Tarpeisiin vastaavaa validointikieltä ei kuitenkaan ole ollut olemassa ennen W3C:n vuonna 2017 julkaisemaa SHACL-suositusta, joka kilpailevan ShEx-rajoitekielen kanssa käyttää erityisiä muotoja RDF-datan rakenteiden kuvailuun ja validointiin. Näiden muotorajoitteiden rikkeet tuottavat rakenteista tietoa virheistä, mikä puolestaan mahdollistaa niiden ohjelmallisen ja avustetun korjaamisen. Korjaamisen mahdollistavia avoimen lähdekoodin ja menetelmän sovelluksia ei kuitenkaan aikaisemmin ole ollut saatavilla huolimatta linkitetyn datan työkalujen ja teknologioiden kehittymisestä.

Tässä työssä kehitettiin menetelmä linkitetyn datan yhdistettyyn validointiin ja korjaamiseen, vastaten työn ainoaan tutkimuskysymykseen sekä ensimmäiseen tavoitteeseen. Menetelmästä onnistuttiin kehittämään iteroiva prosessi, ja sen soveltuvuus käytäntöön testattiin työn toisen tavoitteen mukaisesti kehittämällä prosessia mukaileva suhteellisen helppokäyttöinen ohjelmallinen, avoimen lähdekoodin linkitetyn datan rajoitevalidaattori, joka työn aiheeseen liittyen sai nimen ROTEVA. Menetelmää ja sovellusta testattiin kahdella erityyppisellä koeaineistolla: SKOS-muotoisella Metatietosanastolla sekä CIDOC CRM -tietomallin mukaisella Sotasampo-projektin talvi- ja jatkosotien tapahtumagraafilla. Ohjelmalla onnistuttiin tavoitteiden mukaisesti validoimaan molemmat koeaineistot työn aikana sovelluksella kehitetyillä, aineistojen skeemoihin perustuvilla rajoitteilla. Aineistoista

löydettyjen virheiden korjaaminen päätettiin kuitenkin jättää sovelluksesta riippumattomista syistä johtuen tuonnemmaksi, sillä datalähtöisten virheiden korjaamista lähtöaineistoihin ei nähty mielekkäänä. Sen sijaan työn aikana havaituista virheistä raportoitiin Sotasampo-aineiston tapauksessa graafin tarjoavalle taholle (Metatietosanaston oletettiin olevan jatkuvan kehityksen alainen sanasto ja täten siinä havaittujen virheiden olevan jo korjattu tätä kirjoitettaessa 29.10.2019). Tuloksista päätellen työssä kehitetty menetelmä soveltuu linkitetyn datan aineistojen validointiin ja korjaukseen.

ROTEVAn graafinen käyttöliittymä on saavutettavissa osoitteessa `https://roteva.demo.seco.cs.aalto.fi` ja ohjelmallinen, REST-perustainen, tilaton rajapinta osoitteessa `https://roteva.demo.seco.cs.aalto.fi/api/rest/`. Rajapinnan kuvaus on esitetty liitteessä A. Ohjelmakoodista luotiin nykyohjelmointikäytäntöjen mukaisesti Git-projekti, joka on julkaistu avoimesti saataville Aalto-yliopiston GitLab-verkkoversionhallintajärjestelmässä osoitteessa `https://version.aalto.fi/seco/Roteva`. Projektia tehdessä pyrittiin seuraamaan hyviä ohjelmointitapoja; säilö esimerkiksi sisältää ohjeet järjestelmän käyttöönottoon sekä listauksen ohjelmassa havaituista, vielä virheellisesti toimivista tai puutteellisista ominaisuuksista[1]. Edellä mainittu listaus sisältää lisäksi hyvän projektinhallintatavan mukaisesti ohjelman kehittäjän kirjaamia tai ohjelman kehittäjälle esitettyjä ominaisuuspyyntöjä ja jatkokehityskohteita – sovelluksen kehitys on täten keskitettyä ja hallittua. Seuraavassa on listattu muutamia ROTEVA-rajoitevalidaattorin helppokäyttöisyyttä, käytännöllisyyttä ja tehokkuutta lisääviä toimia:[2]

- Siirtyminen datan kaksinkertaisesta kirjanpidosta yksinkertaiseen kirjanpitoon (#1)
- Suurten, yli 10 000 riviä sisältävien tiedostojen tehokas käsittely (#3)
- Asynkronisen ROTEVA-validointiputken rakentaminen eri ajovalinnoilla (syntaksivalidaattorit, päättelykoneet), palvelupisteiden validointipalvelu ja SHACL $\leftrightarrow$ SPARQL-ominaisuuspolkujen muuntopalvelu (#6)
- SHACL-JS-tuki: paikallinen, selaimessa tapahtuva data-aineiston validointi (#12)

[1] Tämä listaus on esitetty hyvän ohjelmointitavan mukaisesti säilön *issues*-välilehdellä (suom. ongelmat).

[2] Kukin listan alkio päättyy (#$LUKU)-merkintätapaa noudattavaan merkkijonoon, jossa $LUKU on osoitemallia `https://version.aalto.fi/gitlab/seco/Roteva/issues/` täydentävä, kyseistä listan alkiota vastaava, yksilöivä issue-tunnus säilössä. Osoitemallin osoitteesta on lisäksi saavutettavissa kaikki projektin jatkokehityskohteet.

- Ohjelmallisen rajapinnan tasolla tuki erillisille aineistodata- ja rajoiteparametreille, skolemointituki ja JENA-toimintojen tarjoaminen (#13)
- Mahdollisuus siirtyä virheen sisältävästä Datajoukko-taulukon alkiosta Raportti-taulukon virheeseen (#20)
- Validointiraportin ja Datajoukko-taulukon näyttäminen samanaikaisesti vierekkäin selainnäkymässä (#21)
- Valmiit profiilit erityyppisille SHACL-muodoille (#23)
- Automaattisesti täydentyvä tekstinsyöttö avustettuun muotojen muokkaamiseen (#25)
- Toimintojen modularisointi ja erillinen komentorivityökalu (#27)
- Työn aikana SHACL-näkymässä havaittujen tehokkuusongelmien ratkaiseminen (#28)

Edellä esitetyistä ominaisuuksista osa (#6, #13, #27) liittyy työn kolmanteen, toissijaiseen tavoitteeseen eli ROTEVAn toimimiseen täysimittaisena linkitetyn datan alustana. Tämä osoittautui työn edetessä liian suuritöiseksi, minkä takia tähän tarvittavat ominaisuudet päätettiin jättää työn jatkokehitysaiheiksi. Toisaalta esimerkiksi käyttöliittymässä mahdollisuus siirtää validointiraportti data-aineistotaulukon yhteyteen (#20) poistaisi tarpeen siirtää ohjelmallisesti selaimessa näytettävää osaa. Tämä tehostaisi eritoten data-aineiston korjaamista (ja täten käytettävyyttä), sillä virhe sekä sen kohde voisivat tällöin olla samaan aikaan esillä sovelluksessa. ROTEVAan suunniteltiin alun perin mahdollisuutta soveltaa rajoitteita käyttäjän määriteltävissä olevan SPARQL-palvelupisteen datajoukkoon (#6). Tämä ei kuitenkaan vielä työn ohjelmallisen osion toteuttamisvaiheen alkaessa vuonna 2017 ollut mahdollista työn rajoitekieleksi valitulla SHACL-kielen moottorilla [67]. Syyskuussa 2019 julkaistu Apache Jena 3.13.0 -ohjelmistoversio yhdessä Apache Jena Fuseki -SPARQL-palvelimen kanssa kuitenkin mahdollistaisi tämän, alleviivaten työssä todettua viime vuosina tapahtunutta linkitetyn datan työkalujen kypsymistä[3,4].

[3]Aiemmin tämä olisi toki voitu toteuttaa naiivisti esimerkiksi lataamalla koko palvelupisteen sisältö palvelimelle.

[4]ROTEVAn ohjelmakoodi oli jo Apache Jena 3.13.0 -ohjelmistoversion julkaisun aikaan jäädytetty, joten tätä toiminnallisuutta ei ole sovelluksessa.

Tulevaisuudessa SHACL- ja ShEx-validointikieliä voidaan käyttää esimerkiksi datataulukoiden ja -sanastojen kuvauksissa ja visualisoinneissa, kunhan vain asian suhteen vielä keskeneräinen ja tarpeellinen kehitystyö saadaan tehtyä. Lisäksi näiden kielten rajoitemuodot mahdollistavat tehokkaan aligraafien määrittelyn ja ekstrahoinnin, sillä vaikka yksinkertaisten, itsenäisten muotojen tapauksessa tämä olisi helposti toteutettavissa muullakin tavalla, monimutkaisten skeemojen tapauksessa se voi olla vaikeaa – toisin kuin toisiinsa viittaavien muotojen avulla. [68]

Muotojen mahdollisuudet eivät kuitenkaan rajoitu edellä esitettyihin käyttötarkoituksiin, vaan niiden koko hyödyntämispotentiaalin realisoituminen erilaisissa käyttökonteksteissa – joista osaa ei vielä osata kuvitellakaan – odottaa vielä toteutumistaan linkitetyn datan data-aineistojen avautuessa, kasvaessa ja yhdistyessä sekä käytettävien työkalujen kehittyessä. ROTEVA-rajoitevalidaattori tarjoaa vain pienen katsauksen data-aineistojen validointiin, mutta se on yhtä kaikki yksi vaihe eteenpäin pitkällä matkalla kohti Tim Berners-Leen ja kumppanien jo vuonna 2001 visioimaa agenttiperustaista, tietokoneiden yhteismitallista ja yhteisesti ymmärtämää semanttista webiä [17].

Lähdeluettelo

[1] ECMA-262, 8th edition. ECMAScript® 2017 Language Specification. Standardi, Ecma International, 2017.

[2] ECMA-262, 10th edition. ECMAScript® 2019 Language Specification. Standardi, Ecma International, 2019.

[3] ECMA-404. The JSON Data Interchange Syntax. Standardi, Ecma International, 2017.

[4] ISO 21127:2014. Information and documentation – A reference ontology for the interchange of cultural heritage information. Standardi, International Organization for Standardization, 2014.

[5] ISO/IEC 10646:2017. Information technology – Universal Coded Character Set (UCS). Standardi, International Organization for Standardization, 2017.

[6] B. Adida, M. Birbeck, S. McCarron ja I. Herman. RDFa Core 1.1 – Third Edition – Syntax and processing rules for embedding RDF through attributes. W3C Recommendation, W3C, 2015. `http://www.w3.org/TR/2015/REC-rdfa-core-20150317/`.

[7] M. Ali ja M. Alchaita. Enhancing DBpedia Quality Using Markov Logic Networks. *Journal of Theoretical and Applied Information Technology*, 96(12):3924–3936, 2018.

[8] Apache Software Foundation. Apache Jena Shacl – Integration with Apache Jena Fuseki, 2019. `https://jena.apache.org/documentation/shacl/#integration-with-apache-jena-fuseki`. Viitattu 13.10.2019.

[9] K. Arnold, J. Gosling ja D. Holmes. *The Java Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, Massachusetts, Yhdysvallat, kolmas painos, 2000.

[10] T. Atkins, Jr., E. J. Etemad ja F. Rivoal. CSS Snapshot 2018. W3C Working Group Note, W3C, 2019. https://www.w3.org/TR/2019/NOTE-css-2018-20190122/.

[11] T. Baker ja E. Prud'hommeaux. Shape Expressions (ShEx) Primer. W3C Community Group Draft Report – Candidate Release, Shape Expressions Community Group, 2017. http://shex.io/shex-primer-20170713/.

[12] D. Beckett. RDF 1.1 N-Triples – A line-based syntax for an RDF graph. G. Carothers ja A. Seaborne, toimittajat. W3C Recommendation, W3C, 2014. http://www.w3.org/TR/2014/REC-n-triples-20140225/.

[13] D. Beckett, T. Berners-Lee, E. Prud'hommeaux ja G. Carothers. RDF 1.1 Turtle – Terse RDF Triple Language. E. Prud'hommeaux ja G. Carothers, toimittajat. W3C Recommendation, W3C, 2014. http://www.w3.org/TR/2014/REC-turtle-20140225/.

[14] T. Berners-Lee. Linked Data – Design Issues, 2009. https://www.w3.org/DesignIssues/LinkedData.html. Viitattu 29.6.2018.

[15] T. Berners-Lee ja D. Connolly. Notation3 (N3): A readable RDF syntax. W3C Member Submission, W3C, 2011. https://www.w3.org/TeamSubmission/2011/SUBM-n3-20110328/.

[16] T. Berners-Lee, L. Masinter ja M. McCahill. Uniform Resource Locators (URL). RFC 1738, RFC Editor, 1994. http://www.rfc-editor.org/rfc/rfc1738.txt.

[17] T. Berners-Lee, J. Hendler ja O. Lassila. The Semantic Web. *Scientific American*, 284(5):28–37, 2001.

[18] T. Berners-Lee, R. Fielding ja L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, RFC Editor, 2005. http://www.rfc-editor.org/rfc/rfc3986.txt.

[19] C. Bizer ja R. Cyganiak. RDF 1.1 TriG – RDF Dataset Language. G. Carothers ja A. Seaborne, toimittajat. W3C Recommendation, W3C, 2014. http://www.w3.org/TR/2014/REC-trig-20140225/.

[20] T. Bosch ja K. Eckert. Requirements on RDF Constraint Formulation and Validation. *DCMI'14: Proceedings of the 2014 International Conference on Dublin Core and Metadata Applications*, W. Moen ja A. Rushing, toimittajat, sivut 95–108. Dublin Core Metadata Initiative, 2014.

[21] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler ja F. Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation, W3C, 2008. http://www.w3.org/TR/2008/REC-xml-20081126/.

[22] D. Brickley ja R. V. Guha. RDF Schema 1.1. W3C Recommendation, W3C, 2014. http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[23] G. Carothers. RDF 1.1 N-Quads – A line-based syntax for RDF datasets. W3C Recommendation, W3C, 2014. http://www.w3.org/TR/2014/REC-n-quads-20140225/.

[24] V. Cerf. ASCII format for Network Interchange. RFC 20, RFC Editor, 1969. https://www.rfc-editor.org/rfc/rfc20.txt.

[25] R. Chinnici, H. Haas, A. A. Lewis, J.-J. Moreau, D. Orchard ja S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts. W3C Recommendation, W3C, 2007. http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626/.

[26] R. Chinnici, J.-J. Moreau, A. Ryman ja S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Recommendation, W3C, 2007. http://www.w3.org/TR/2007/REC-wsdl20-20070626/.

[27] K. Clark ja E. Sirin. On RDF Validation, Stardog ICV, and Assorted Remarks. *RDF Validation Workshop. Practical Assurances for Quality RDF Data*, Cambridge, Massachusetts, Yhdysvallat, 2013.

[28] S. Cook, C. Bock, P. Rivett, T. Rutt, E. Seidewitz, B. Selic ja D. Tolbert. Unified Modeling Language (UML) Version 2.5.1. Standardi, Object Management Group (OMG), 2017. https://www.omg.org/spec/UML/2.5.1.

[29] J. Corman, J. L. Reutter ja O. Savković. Semantics and Validation of Recursive SHACL. *The Semantic Web – ISWC 2018. 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part I*, D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L.-A. Kaffee ja E. Simperl, toimittajat, sivut 318–336. Springer International Publishing, 2018.

[30] J. Corman, J. L. Reutter ja O. Savković. A Tractable Notion of Stratification for SHACL. *Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th*

International Semantic Web Conference (ISWC 2018), M. van Erp, M. Atre, V. Lopez, K. Srinivas ja C. Fortuna, toimittajat. CEUR Workshop Proceedings, 2018.

[31] R. Cyganiak, D. Wood ja M. Lanthaler. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, W3C, 2014. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[32] E. Dahlström, P. Dengler, A. Grasso, C. Lilley, C. McCormack, D. Schepers ja J. Watt. Scalable Vector Graphics (SVG) 1.1 (Second Edition). W3C Recommendation, W3C, 2011. http://www.w3.org/TR/2011/REC-SVG11-20110816/.

[33] J. Debattista, S. Auer ja C. Lange. Luzzu – A Framework for Linked Data Quality Assessment. *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*, sivut 124–131. IEEE Computer Society, 2016.

[34] S. Deering ja R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 8200, RFC Editor, 2017. http://www.rfc-editor.org/rfc/rfc8200.txt.

[35] M. Duerst ja M. Suignard. Internationalized Resource Identifiers (IRIs). RFC 3987, RFC Editor, 2005. http://www.rfc-editor.org/rfc/rfc3987.txt.

[36] I. Ermilov, J. Lehmann, M. Martin ja S. Auer. LODStats: The Data Web Census Dataset. *The Semantic Web – ISWC 2016. 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part II*, P. Groth, E. Simperl, A. Gray, M. Sabou, M. Krötzsch, F. Lecue, F. Flöck ja Y. Gil, toimittajat, sivut 38–46. Springer International Publishing, 2016.

[37] Euroopan unionin julkaisutoimisto. ELI Validator documentation, 2019. https://publications.europa.eu/eli-validator/documentation. Viitattu 10.7.2019.

[38] S. Faulkner, A. Eicholz, T. Leithead, A. Danilo ja S. Moon. HTML 5.2. W3C Recommendation, W3C, 2017. https://www.w3.org/TR/2017/REC-html52-20171214/.

[39] K. C. Feeney, G. Mendel Gleason ja R. Brennan. Linked data schemata: Fixing unsound foundations. *Semantic Web*, 9(1):53–75, 2018.

[40] J. D. Fernández, W. Beek, M. A. Martínez-Prieto ja M. Arias. LOD-a-lot: A Queryable Dump of the LOD Cloud. *The Semantic Web — ISWC 2017. 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II*, C. d'Amato, M. Fernandez, V. Tamma, F. Lecue, P. Cudré-Mauroux, J. Sequeda, C. Lange ja J. Heflin, toimittajat, sivut 75–83. Springer International Publishing, 2017.

[41] J. D. Fernández, M. A. Martínez-Prieto, C. Gutiérrez, A. Polleres ja M. Arias. Binary RDF representation for publication and exchange (HDT). *Journal of Web Semantics*, 19:22–41, 2013.

[42] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach ja T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, 1999. `http://www.rfc-editor.org/rfc/rfc2616.txt`.

[43] R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Väitöskirja, University of California, Irvine, 2000.

[44] B. Forouzan. *TCP/IP Protocol Suite*. McGraw-Hill, Inc., New York, New York, Yhdysvallat, neljäs painos, 2010.

[45] L. Galárraga, S. Razniewski, A. Amarilli ja F. M. Suchanek. Predicting Completeness in Knowledge Bases. *WSDM '17 Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, sivut 375–383. ACM, 2017.

[46] F. Gandon ja G. Schreiber. RDF 1.1 XML Syntax. W3C Recommendation, W3C, 2014. `http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/`.

[47] S. Gao, C. M. Sperberg-McQueen ja H. S. Thompson. W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. W3C Recommendation, W3C, 2012. `http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/`.

[48] J. J. Garrett. Ajax: A New Approach to Web Applications. *Adaptive Path*, 2005.

[49] P. Gearon, A. Passant ja A. Polleres. SPARQL 1.1 Update. W3C Recommendation, W3C, 2013. `http://www.w3.org/TR/2013/REC-sparql11-update-20130321/`.

[50] S. Harris ja A. Seaborne. SPARQL 1.1 Query Language. W3C Recommendation, W3C, 2013. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[51] P. J. Hayes ja P. F. Patel-Schneider. RDF 1.1 Semantics. W3C Recommendation, W3C, 2014. http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/.

[52] E. Heino. Sotahistorian kuvaaminen ja rikastaminen linkitettynä datana. Pro gradu -tutkielma, Helsingin yliopisto, 2017.

[53] I. Horrocks ja P. F. Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. *The Semantic Web – ISWC 2003. Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003. Proceedings*, D. Fensel, K. Sycara ja J. Mylopoulos, toimittajat, sivut 17–29. Springer Berlin Heidelberg, 2003.

[54] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof ja M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, W3C, 2004. https://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

[55] I. Horrocks, O. Kutz ja U. Sattler. The Even More Irresistible SROIQ. *KR2006: Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning*, P. Doherty, J. Mylopoulos ja C. A. Welty, toimittajat, sivut 57–67. AAAI Press, 2006.

[56] E. Hyvönen, E. Heino, P. Leskinen, E. Ikkala, M. Koho, M. Tamper, J. Tuominen ja E. Mäkelä. WarSampo Data Service and Semantic Portal for Publishing Linked Open Data about the Second World War History. *The Semantic Web. Latest Advances and New Domains – 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 – June 2, 2016, Proceedings*, H. Sack, E. Blomqvist, M. d'Aquin, C. Ghidini, S. P. Ponzetto ja C. Lange, toimittajat, sivut 758–773. Springer International Publishing, 2016.

[57] Kansalliskirjasto. Metatietosanasto, 2019. http://finto.fi/mts/fi/. Viitattu 9.9.2019.

[58] Kansalliskirjasto. YSA – Yleinen suomalainen asiasanasto, 2019. http://finto.fi/ysa/fi/. Viitattu 27.9.2019.

[59] Kansalliskirjasto. YSO – Yleinen suomalainen ontologia, 2018. http://finto.fi/yso/fi/. Viitattu 7.6.2018.

[60] A. van Kesteren, A. Gregor, Ms2ger, A. Russell ja R. Berjon. W3C DOM4. W3C Recommendation, W3C, 2015. http://www.w3.org/TR/2015/REC-dom-20151119/.

[61] H. Knublauch. SPIN – SPARQL Inferencing Notation, 2017. http://spinrdf.org/spin-shacl.html. Viitattu 17.7.2018.

[62] H. Knublauch. From SPIN to SHACL, 2017. http://spinrdf.org/spin-shacl.html. Viitattu 17.7.2018.

[63] H. Knublauch ja D. Kontokostas. Shapes Constraint Language (SHACL). W3C Recommendation, W3C, 2017. https://www.w3.org/TR/2017/REC-shacl-20170720/.

[64] H. Knublauch ja P. Maria. SHACL JavaScript Extensions. W3C Working Group Note, W3C, 2017. https://www.w3.org/TR/2017/NOTE-shacl-js-20170608/.

[65] H. Knublauch, J. A. Hendler ja K. Idehen. SPIN - Overview and Motivation. W3C Member Submission, W3C, 2011. https://www.w3.org/Submission/2011/SUBM-spin-overview-20110222/.

[66] D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann ja R. Cornelissen. Databugger: A test-driven framework for debugging the web of data. *WWW'14 Companion: Proceedings of the 23rd International Conference on World Wide Web*, sivut 115–118. ACM, 2014.

[67] kouralex ja HolgerKnublauch. Querying an endpoint via SPARQL generated by the SHACL engine – Issue #17 – TopQuadrant/shacl, 2017. https://github.com/TopQuadrant/shacl/issues/17. Viitattu 16.10.2019.

[68] J. E. Labra Gayo, E. Prud'hommeaux, I. Boneva ja D. Kontokostas. *Validating RDF Data*. Morgan & Claypool, 2017.

[69] J. E. Labra-Gayo, D. Fernández-Álvarez ja H. García-González. RDFShape: An RDF Playground Based on Shapes. *Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th International Semantic Web Conference (ISWC 2018)*, M. van Erp, M. Atre, V. Lopez, K. Srinivas ja C. Fortuna, toimittajat. CEUR Workshop Proceedings, 2018.

[70] J. E. Labra Gayo, H. Knublauch ja D. Kontokostas. SHACL Test Suite and Implementation Report. W3C Document, W3C, 2018.

https://w3c.github.io/data-shapes/data-shapes-test-suite/. Viitattu 18.6.2018.

[71] L. Lacroix. Shape Expressions arrive on Wikidata on May 28th, 2019. https://www.wikidata.org/wiki/Wikidata:Project_chat/Archive/2019/05#Shape_Expressions_arrive_on_Wikidata_on_May_28th. Viitattu 25.6.2019.

[72] J. Laitio. Semantic Web Data Quality Control. Diplomityö, Aalto-yliopiston sähkötekniikan korkeakoulu, 2011.

[73] O. Lassila ja R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, W3C, 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[74] Lightbend Inc. ApplicationSecret – 2.5.x, 2018. https://www.playframework.com/documentation/2.5.x/ApplicationSecret. Viitattu 9.6.2018.

[75] Lightbend Inc. Assets – 2.5.x, 2018. https://www.playframework.com/documentation/2.5.x/Assets. Viitattu 23.6.2018.

[76] F. Manola, E. Miller ja B. McBride. RDF Primer. W3C Recommendation, W3C, 2004. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

[77] D. L. McGuinness ja F. van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, W3C, 2004. http://www.w3.org/TR/2004/REC-owl-features-20040210/.

[78] A. Melo. *Automatic refinement of large-scale cross-domain knowledge graphs*. Väitöskirja, Univärsität Mannheim, School of Business Informatics and Mathematics, 2018.

[79] A. Miles ja S. Bechhofer. SKOS Simple Knowledge Organization System Reference. W3C Recommendation, W3C, 2009. http://www.w3.org/TR/2009/REC-skos-reference-20090818/.

[80] P. Mockapetris ja K. J. Dunlap. Development of the Domain Name System. *SIGCOMM '88: Symposium Proceedings on Communications Architectures and Protocols*, V. Cerf, toimittaja, sivut 123–133. ACM, 1988.

[81] B. Motik, P. F. Patel-Schneider ja B. Parsia. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Ruttenberg, U. Sattler ja M. Smith, avustajat, W3C Recommendation, W3C, 2012. http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/.

[82] M. Nilsson. Description Set Profiles: A constraint language for Dublin Core Application Profiles. DCMI Working Draft, DCMI, 2008. http://www.dublincore.org/specifications/dublin-core/dc-dsp/2008-03-31/.

[83] M. Odersky. The Scala Language Specification Version 2.9. Tekninen raportti, Programming Methods Laboratory, EPFL, 2014.

[84] H. Paulheim. Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods. *Semantic web*, 8(3):489–508, 2017.

[85] D. Peterson, S. Gao, A. Malhotra, C. M. Sperberg-McQueen ja H. S. Thompson. W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. W3C Working Group Note, W3C, 2001. http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/.

[86] A. Phillips ja M. Davis. Tags for Identifying Languages. RFC 5646/BCP 47, RFC Editor, 2009. https://www.rfc-editor.org/bcp/bcp47.txt.

[87] E. Prud'hommeaux ja T. Baker. ShapeMap Structure and Language. W3C Community Group Draft Report, Shape Expressions Community Group, 2017. http://shex.io/shape-map-20170713/.

[88] E. Prud'hommeaux, I. Boneva, J. E. Labra Gayo ja G. Kellogg. Shape Expressions Language 2.0. W3C Community Group Draft Report – Candidate Release, Shape Expressions Community Group, 2017. http://shex.io/shex-semantics-20170713/.

[89] F. Radulovic, N. Mihindukulasooriya, R. García-Castro ja A. Gómez-Pérez. A Comprehensive Quality Model for Linked Data. *Semantic Web*, 9(1):3–24, 2018.

[90] E. Rescorla. HTTP Over TLS. RFC 2818, RFC Editor, 2000. http://www.rfc-editor.org/rfc/rfc2818.txt.

[91] A. Ryman. Resource Shape 2.0. W3C Member Submission, W3C, 2014. https://www.w3.org/Submission/2014/SUBM-shapes-20140211/.

[92] S. Sam, P. Hitzler ja K. Janowicz. On the Quality of Vocabularies for Linked Dataset Papers Published in the Semantic Web Journal. *Semantic Web*, 9:207–220, 2018.

[93] A. Seaborne. [ANN] Apache Jena 3.13.0, 2019. https://www.mail-archive.com/users@jena.apache.org/msg16479.html. Viitattu 13.10.2019.

[94] Shape Expressions Community Group. ShEx2 – Simple Online Validator, 2019. https://rawgit.com/shexSpec/shex.js/master/packages/shex-webapp/doc/shex-simple.html. Viitattu 26.6.2019.

[95] H. Solbrig ja E. Prud'hommeaux. Shape Expressions 1.0 Definition. W3C Member Submission, W3C, 2014. https://www.w3.org/Submission/2014/SUBM-shex-defn-20140602/.

[96] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler ja N. Lindström. JSON-LD 1.0 – A JSON-based Serialization for Linked Data. M. Sporny, G. Kellogg ja M. Lanthaler, toimittajat. W3C Recommendation, W3C, 2014. http://www.w3.org/TR/2014/REC-json-ld-20140116/.

[97] Standing Committee of the IFLA Cataloguing Section, toimittaja. *ISBD: International Standard Bibliographic Description – Consolidated Edition*. De Gruyter Saur, 2011.

[98] Stardog Union. Integrity Constraint Validation, 2018. https://github.com/stardog-union/stardog-examples/tree/develop/examples/cli/icv. Viitattu 10.7.2019.

[99] Stardog Union. Stardog 6: The Manual, 2019. https://www.stardog.com/docs/6.2.1/. Viitattu 9.7.2019.

[100] S. Steyskal ja K. Coyle. SHACL Use Cases and Requirements. W3C Working Group Note, W3C, 2017. https://www.w3.org/TR/2017/NOTE-shacl-ucr-20170720/.

[101] O. Suominen ja E. Hyvönen. Improving the Quality of SKOS Vocabularies with Skosify. *Knowledge Engineering and Knowledge Management – 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012. Proceedings*, A. ten Teije, J. Völker, S. Handschuh, H. Stuckenschmidt, M. d'Acquin, A. Nikolov, N. Aussenac-Gilles ja

N. Hernandez, toimittajat, sivut 383–397. Springer Berlin Heidelberg, 2012.

[102] O. Suominen ja C. Mader. Assessing and Improving the Quality of SKOS Vocabularies. *Journal on Data Semantics*, 3(1):47–73, 2014.

[103] D. Tomaszuk. RDF Validation: A Brief Survey. *Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation – 13th International Conference, BDAS 2017, Ustroń, Poland, May 30 - June 2, 2017, Proceedings*, S. Kozielski, D. Mrozek, P. Kasprowski, B. Małysiak-Mrozek ja D. Kostrzewa, toimittajat, sivut 344–355. Springer International Publishing, 2017.

[104] TopQuadrant. TopBraid Composer – Maestro Edition, 2019. `https://www.topquadrant.com/products/topbraid-composer/`. Viitattu 9.7.2019.

[105] R. Trygve. Models-Views-Controllers. Xerox PARC technical note, Xerox PARC, 1979. `https://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf`.

[106] P.-Y. Vandenbussche, J. Umbrich, L. Matteis, A. Hogan ja C. Buil-Aranda. SPARQLES: Monitoring Public SPARQL Endpoints. *Semantic Web*, 8(6):1049–1065, 2017.

[107] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (Second Edition). W3C Recommendation, W3C, 2012. `http://www.w3.org/TR/2012/REC-owl2-overview-20121211/`.

[108] R. Y. Wang ja D. M. Strong. Beyond Accuracy: What Data Quality Means to Data Consumers. *Journal of management information systems*, 12(4):5–33, 1996.

[109] Wikipedia. Luettelo Suomen ilmavoimien ilma-aluksista, 2019. `https://fi.wikipedia.org/wiki/Luettelo_Suomen_ilmavoimien_ilma-aluksista#Viides_merkint%C3%A4tapa`. Viitattu 27.9.2019.

[110] D. Wood. What's New in RDF 1.1. W3C Working Group Note, W3C, 2014. `http://www.w3.org/TR/2014/NOTE-rdf11-new-20140225/`.

[111] F. Yergeau. UTF-8, a transformation format of ISO 10646. RFC 3629, RFC Editor, 2003. `http://www.rfc-editor.org/rfc/rfc3629.txt`.

[112] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann ja S. Auer. Quality Assessment for Linked Data: A Survey. *Semantic Web*, 7(1): 63–93, 2016.

[113] A. Zimmermann. RDF 1.1: On Semantics of RDF Datasets. W3C Working Group Note, W3C, 2014. `http://www.w3.org/TR/2014/NOTE-rdf11-datasets-20140225/`.

Liite A

REST-rajapinta

POST /api/rest/validate

Työn päämetodi, joka validoi annetun datan ja tuottaa validointiraportin käyttäjän haluamassa formaatissa.

Otsakkeet

Content-Type

Sisällön MIME-tyyppi. Mahdollisia arvoja ovat *application/json* (oletusarvo), *application/x-www-form-urlencoded* ja *multipart/form-data*). Valinnainen.

Accept

Vastauksena halutun formaatin MIME-tyyppi. Ohjelma tukee kehyksellisiä JSON-LD-tallennusmuotoja[1] lukuun ottamatta kaikkia Apache Jena RIOT -kirjoittimen tarjoamia syntakseja[2]. Yliajettavissa ja tarkennettavissa[3] *to*-parametrilla. Valinnainen.

[1] JSONLD_FRAME_PRETTY ja JSONLD_FRAME_FLAT.

[2] Listaus sivulla `https://jena.apache.org/documentation/io/rdf-output.html`.

[3] Ohjelma tulostaa raportin oletuksena käyttäen JSON-LD-syntaksia.

Parametrit

data

Validoitava data[4] *from*-parametrin formaatissa. Pakollinen.

from

Datan formaatti. Ohjelma tukee kehyksellisiä JSON-LD-tallennusmuotoja[1] lukuun ottamatta kaikkia Apache Jena RIOT -kirjoittimen tarjoamia syntakseja[2]. Valinnainen (oletuksena JSON-LD).

to

Validointiraportin haluttu formaatti. Ohjelma tukee kehyksellisiä JSON-LD-tallennusmuotoja[1] lukuun ottamatta kaikkia Apache Jena RIOT -kirjoittimen tarjoamia syntakseja[2]. Valinnainen (oletuksena JSON-LD).

Vastauskoodit

HTTP/1.1 200 OK

Otsakkeet: *Content-Type* asetettuna ensisijaisesti *to*-parametrin ja toissijaisesti *Accept*-otsakkeen MIME-tyypin arvoon.

Sisältö: Validointiraportti missä tahansa (kehyksettömiä JSON-LD-tallennusmuotoja[1] lukuun ottamatta) Apache Jena RIOT -kirjoittimen tarjoamassa syntaksissa[2].

HTTP/1.1 400 Bad Request

Otsakkeet: *Content-Type: text/plain; charset=utf-8*

Sisältö: Tapahtunut virhe.

[4]Ladattava tiedosto, kun *Content-Type* on asetettu arvoon *multipart/form-data*.

Liite B

Luettelo ei-standardeista pikanäppäimistä

Pikanäppäin	Funktio
Kun kohdistus on Dokumentissa	
D	Asettaa kohdistimen Datajoukko-taulukkoon.
R	Asettaa kohdistimen Raportti-taulukkoon.
V	Asettaa kohdistimen Validate-painikkeeseen.
T	Asettaa kohdistimen vapaamuotoiseen datan kenttään.
P	Asettaa kohdistimen etuliitteisiin.
Kun kohdistus on Datajoukko- tai Raportti-taulukossa	
M	Asettaa kohdistimen taulukon valikkopalkkiin.
F	Asettaa kohdistimen taulukon suodattimeen.
Välilyönti	Avaa/sulje aktiivinen ryhmä (vain SHACL-näkymässä).
Kun kohdistus on Datajoukko-taulukossa	
E	Muokkaa aktiivista riviä/ryhmää.
Delete	Poista aktiivinen rivi/ryhmä.
Kun kohdistus on taulukoissa tai vapaamuotoisen datan kentässä	
Esc+Sarkain	Asettaa kohdistimen seuraavaan elementtiin.
Vaihto+Esc	Asettaa kohdistimen edelliseen elementtiin.

Liite C

SPARQL-ominaisuuspolkulausekkeet SHACL-kielellä

Taulukossa C.1 on esitetty SHACL-rajoitekielelle mukautuvat SPARQL-ominaisuuspolkulausekkeet[1] sekä niiden SHACL-vastineet Turtle-sarjallistamismuodossa. Suurempi presedenssi tarkoittaa aiemmin suoritettavaa lauseketta. ELT*, ELT? ja ELT+ assosioituvat vasemmalle. IRI on lyhennetyssä tai lyhentymättömässä muodossa kirjoitettu IRI-tunniste ja ELT polun osa, joka voi koostua muista lausekkeista.

SPARQL-lauseke	**SHACL-vastine Turtlena**	**Presedenssi**
IRI	IRI	6
(ELT)	ELT	5
ELT*	[sh:zeroOrMorePath ELT]	4
ELT?	[sh:zeroOrOnePath ELT]	4
ELT+	[sh:oneOrMorePath ELT]	4
^ELT	[sh:inversePath ELT]	3
ELT_1 / ELT_2	(ELT_1 ELT_2)	2
ELT_1 | ELT_2	[sh:alternativePath (ELT_1 ELT_2)]	1

Taulukko C.1: SHACL-rajoitekielelle mukautuvat SPARQL-ominaisuuspolkulausekkeet sekä niiden SHACL-rajoitekielen mukaiset Turtle-sarjallistamismuodot presedenssittäin[2].

[1] Esimerkiksi SPARQL 1.1:n mukaiset *käänteiset ominaisuusjoukot* (engl. negated property sets) eivät muunnu SHACL-rajoitekielelle.

[2] Predikaattien kuvaamiseen on käytetty seuraavaa nimiavaruutta:
sh: <http://www.w3.org/ns/shacl#>.

Liite D

Metatietosanasto-testiaineistolle työn aikana kehitetyt rajoitemääritykset

```
@base <http://seco.cs.aalto.fi/roteva/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .

<uniikitArvotPrefLabel> sh:path skos:prefLabel ;
                        sh:uniqueLang true .

<terminologisetLähteet> sh:path dc:source ;
                        sh:targetClass skos:Concept ;
                        sh:pattern "^(ISBD)|(RDA)" .

<varoitaTyhjistäRyhmistä> sh:path skos:member ;
                          sh:targetClass skos:Collection ;
                          sh:minCount 1 ;
                          sh:severity sh:Warning .

<ryhmäRajoite> sh:path [ sh:inversePath skos:member ] ;
               sh:targetClass skos:Concept ;
               sh:minCount 1 .

<prefLabelTarkistin> sh:targetClass skos:Collection ,
                                    skos:Concept ;
                     sh:property <uniikitArvotPrefLabel> .
                     sh:sparql <suomiPrefLabel>,
                               <ruotsiPrefLabel> .
```

```
<ruotsiPrefLabel> sh:message
   "Ei suositeltavaa nimikettä ruotsiksi"@fi ;
   sh:select
   """SELECT $this $PATH ?value
      WHERE {
        OPTIONAL {
          $this $PATH ?value .
          FILTER (LANG(?value) = "sv")
        }
        FILTER (!BOUND(?value))
      }""" .

<suomiPrefLabel> sh:message
   "Ei suositeltavaa nimikettä suomeksi"@fi ;
   sh:select
   """SELECT $this $PATH ?value
      WHERE {
        OPTIONAL {
          $this $PATH ?value .
          FILTER (LANG(?value) = "fi")
        }
        FILTER (!BOUND(?value))
      }""" .
```

Liite E

Sotasampo-testiaineistolle työn aikana kehitetyt rajoitemääritykset

```
@base <http://seco.cs.aalto.fi/roteva/> .
@prefix we: <http://ldf.fi/warsa/events/>.
@prefix ws: <http://ldf.fi/schema/warsa/>.
@prefix wse: <http://ldf.fi/schema/warsa/events/>.
@prefix warsa: <http://ldf.fi/warsa/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix skos: <http://www.w3.org/2004/02/skos/core#>.
@prefix crm: <http://www.cidoc-crm.org/cidoc-crm/>.
@prefix sh: <http://www.w3.org/ns/shacl#>.

<prefLabelTarkistin> sh:path skos:prefLabel ;
                     sh:uniqueLang true ;
                     sh:targetClass ws:Event .

<muutKuinYsaRelated> sh:path skos:related ;
                     sh:pattern "^http://www.yso.fi/onto/ysa/" ;
                     sh:severity sh:Info ;
                     sh:targetClass ws:MilitaryActivity ,
                                    ws:PoliticalActivity .

<lentokoneMuoto> sh:path wse:aircraft ;
                 sh:pattern "^..-[0-9]+" .

<alkuEnnenLoppua> sh:path wse:time_start ;
                  sh:targetClass ws:Battle ;
                  sh:lessThan wse:time_end .
```

```
<aikaMuoto> sh:targetObjectsOf wse:time_start ,
                               wse:time_end ;
            sh:nodeKind sh:Literal ;
            sh:datatype xsd:time .

<taisteluMuoto> sh:targetClass ws:Battle ;
                sh:property <lentokoneMuoto> ;
                sh:sparql <toteuttajaEiOllutOsanottaja> ,
                          <tyhjäMerkkijonoRajoite> .

<tyhjäMerkkijonoRajoite> sh:message
   "Tyhjä merkkijono ominaisuuden arvona"@fi ;
   sh:select
   """SELECT ?this (?p AS ?path) ("" AS ?value)
      WHERE {
        ?this ?p "" .
      }""" .

<toteuttajaEiOllutOsanottaja> sh:message
   "Toteuttaja ei ollut osanottaja taistelussa"@fi ;
   sh:prefixes [
     sh:declare [
       sh:prefix "crm" ;
       sh:namespace "http://www.cidoc-crm.org/cidoc-crm/"
     ]
   ] ;
   sh:select
   """SELECT ?this ?value
      WHERE {
        ?this crm:P14_carried_out_by ?value .
        FILTER (NOT EXISTS {
          ?this crm:P11_had_participant ?value .
        })
      }""" .
```