UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Institute of Computer Science

Software Engineering

Mailis Toompuu

PaaS Cloud Service for Cost-Effective Harvesting, Processing and Linking of Unstructured Open Government Data

Master's thesis (30ECP)

Supervisor: Peep Küngas, PhD

Tartu 2015

PaaS Cloud Service for Cost-Effective Harvesting, Processing and Linking of Unstructured Open Government Data

Abstract

The aim of this project is to develop a cloud platform service for transforming Open Government Data to Linked Open Government Data.

This service receives log file, created by web crawler, with URLs (over 3000000) to some open document as an input. It then opens the document, reads its content and with using "Open source tools for Estonian natural language processing" (Estnltk), finds names of locations, organizations and people. Using Psython library "RDFlib", these names are added to the Resource Description Framework (RDF) graph, so that the names become linked to the URLs that refer to the documents. In order to archive current state of accessed document, this service downloads all processed documents. The service also enables monthly updates system of the already processed documents in order to generate new RDF relations if some of the documents have changed. Generated RDFs are publicly available and the service includes SPARQL endpoint for userss (graphical user interface) and machines (web services) for cost-effective querying of linked entities from the RDF files.

An important challenge of this service is to speed up its performance, because the documents behind these 3+ million URLs may be large. To achieve that, parallel processes are run where possible: using several virtual machines and all CPUs in a virtual machine. This is tested in Google Compute Engine.

Keywords

semantic web, RDF, open government data, cloud service, cost-effective processing, named entity extraction, named entity recognition, RDFlib, Estnltk, NERD, OWL, RDFs, FOAF, DC, Google Compute Engine

Pilveteenus Eesti struktrueerimata avalike andmete kuluefektiivseks töötlemiseks, linkimiseks ja päringute tegemiseks

Lühikokkuvõte

Selle projekti eesmärk on luua pilveteenus, mis võimaldaks struktueerimata avalike andmete töötlemist, selleks, et luua semantiline andmete (veebis olevatest dokumentidest leitud organisatsioonide, kohanimede ja isikunimede) ressursikirjeldusraamistiku - *Resource Description Framework* (RDF) - graaf, mis on ka masinloetav.

Pilveteenus saab sisendiks veebiroomaja toodetud logifaili üle 3 miljoni reaga. Igal real on veebiaadress avalikule dokumendile, mis avatakse, loetakse ning kasutades - tööriista eestikeelsest tekstist nimeolemite leidmiseks- Estnltk-d, eraldatakse organisatsiooonide ja kohtade nimetused ja inimeste nimed. Seejärel lisatakse leitud nimed/nimetused RDF graafi, kasutades olemasolevat Pythoni teeki RDFlib. RDF graafis nimed/nimetused lingitakse nende veebiaadressidega, kus asub seda nime/nimetust sisaldav avalik dokument. Dokumendid arhiveeritakse lugemise hetkel neis olnud sisuga. Lisaks sisaldab teenus igakuist andmete ülekontrollimist, et tuvastada dokumentide muutusi ja vajadusel värskendada RDF graafe. Genereeritud RDF graafe kasutatakse SPARQL päringute tegemiseks, mida saavad teha kasutajad graafilise kasutajaliidese kaudu või masinad veebiteenust kasutades.

Projekti oluline väljakutse on luua arhitektuur, mis töötleks andmeid võimalikult kiiresti, sest sisendfail on suur (test-logifailis on üle 3 miljoni rea, kus igal real olev URL võib viidata mahukale dokumendile). Selleks jooksutab teenus seal kus võimalik, protsesse paralleelselt, kasutades Google'i virtuaalmasinaid (*Google Compute Engine*) ja iga virtuaalmasina kõiki protsessoreid.

Võtmesõnad

semantiline veeb, RDF, avalikud valitsuse andmed, pilveteenus, kuluefektiivne töötlemine, nimeüksuse tuvastamine, nimeolemite tuvastamine, RDFlib, Estnltk, NERD, OWL, RDFs, FOAF, DC, Google Compute Engine

Contents

Abbreviations and Meanings of Words

PaaS	Platform as a Service
VM	Virtual Machine
VMs	Virtual Machines
master	Virtual Machine named “master”
worker	Virtual Machine named “worker”

million – UK English term for denoting number 1 000 000

RDFizing – process of generating RDF graphs from unstructured or structured data, databases etc.

unresolved entity – an entity extracted from natural language, but it is not confirmed that this entity actually exists

1 Introduction

Governmental institutions produce many documents that are freely accessible to everyone. These documents assumingly contain names of people, organizations and locations. While producing, reading or analysing these documents, it may be useful to be aware in what context and where these names appear elsewhere. Linking the names to all data sources where the names appear - in a machine-readable format - would substantially increase the quality, transparency, accessibility and reliability of Open Government Data (OGD) [1].

The open data documents are in various formats like web pages, XML-, PDF-, plain text-, Excel-files etc. These documents in most cases are written in natural language and are unstructured or semi-structured. In case of semi-structured XML formats, the exact meaning of XML nodes is unknown for consumer outside of an application. Semantic web movement has evolved increasingly popular solution for linking data sources: Resource Description Framework (RDF) [2] graphs that have machine-readable format. Additionally, RDF graphs can be integrated with other RDF graphs without the specific API constraints on data. Semantic web and RDF graphs are concisely introduced in chapter "Background".

This thesis is mainly focused on a strict solution of the RDFizing[1] problem for Estonian OGD and on achievement of its as high performance as possible. The problems of security and authentication of users are out of the thesis scope, therefore they are not addressed here

Before linking names and datasets, the names should be extracted. Regarding the amount of open data, linking names to their sources cannot be performed manually. Some well-known semantic web applications and initiatives that extract entities from unstructured or/and semi-structured data and face other challenges of semantic web (e.g. storage of large graphs and effective querying of RDF graphs) are introduced in chapters "Background" and "Related Work".

Considering there is no yet system for linking data from unstructured documents in Estonian, the aim of the thesis is to develop a cloud platform service for transforming Estonian OGD into Estonian Linked OGD.

The input data of this service is a log file produced by a web crawler [3] having a certain structure: every line of it contains URL to a file and some metadata. The log file used for testing this service, contains over 3.1 million lines. The service parses these lines, validates URLs using the metadata contained in a line and reads the document's contents at that URL. The range of the document types that this system processes has textual contents, because it later uses entity recognition tool (the open source tool for Estonian natural language processing (Estnltk) [4]), that extracts names of people, organizations and locations. Other formats, e.g. images are ignored. The process of this system is precisely described in the subchapter of chapter "Architecture".

After accessing the content of a document, the service uses Estnltk for extracting the named entities. Extracted entities are then added to RDF graph using Python's library "RDFlib" [5]. These and other technological decisions are reasoned in the subchapters of chapter “Background”. The metadata model used in these RDF graphs is described in chapter "RDF Data Model".

This service is supposed also to

- keep track on whether some document's content have changed after its last processing (RDFizing);

[1] RDFizing – process of creating RDF graphs from non-RDF resources (e.g. unstructured and semistructured texts, databases etc.)

- download the processed (RDFized) documents;
- enable browsing the datasets by document's metadata;
- call monthly updates of changed documents for updating RDF graphs,

therefore it is necessary to store metadata of documents and the time when a document was accessed/RDFized, in a structured way. Reading and writing this metadata model should be fast in order not to slow down the performance. This metadata is described in chapter "Metadata Model".

After the generation of RDF graphs is completed, the machines and also human users should be able to query names and web addresses of the documents described in RDF graph by using SPARQL endpoint. For human users the graphical user interface should be included into service: for

- uploading log files;
- starting RDFizing process;
- browsing datasets;
- making (simulating in the development phase) monthly updates;
- SPARQL endpoint for querying RDF graphs;
- reading thrown errors (for developers).

In addition to the development of described service, another challenge is to elaborate an architecture where this service generates RDF triples during a reasonable time, e.g. in 12-24 hours (regarding the input log file with over 3.1 million URLs to documents). For achieving high performance of this service, the program is adapted to cloud infrastructure. The cloud computing service provider used in this project is Google Compute Engine (GCE) [6], but the software (after changing the code parts related to GCE-specific API) can be migrated to other cloud service provider as well. 4+1 architecture is described in chapter "Architecture".

2 Background

This chapter briefly introduces the semantic web movement, the RDF [2] model technology and the reasons for the chosen technologies of the current project.

The freely accessible raw data documents in the Internet, like HTML pages, Excel sheets, XML and other textual documents are unstructured or semi-structured. It is hard to analyse them by machines. The ideas and technologies described in the following subchapters are supposed to solve this problem.

2.1 Open Data And Open Government Data

When browsing for definitions of open data (OD) and government open data, it is stated that open data is the data that is freely accessible and usable to everyone without any restrictions [7]; open government data (OGD) is the OD that is published by governmental institutions. Advocates of OGD stress that the publication of governmental data in open formats increases the government's transparency and liability. [8, point 1] reasons that „*Transparency isn't just about access, it is also about sharing and reuse*". One can imagine the huge amount of the OD/OGD and wonder how applications should utilize this in a constructive and valuable way: the OD/OGD must be machine-readable and the data sets that are related to each other must be linked/linkable.

2.2 Semantic Web Movement

The main value of linked data stands in having an advantage of being integrated to other open linked data and being machine-readable due to the semantic graph structure. W3C has defined the Semantic Web (Web of Data) that "/.../ provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries" [9]. Although the concept of Semantic Network is over 50 years old, the term Semantic Web was first coined in 2001 [10, page 34–43] and the movement of a semantic web of data is relatively young: in 2006, Tim Berners-Lee introduced 5-star rating system for data [11]. The lowest rating, as shown in Table 1, is assigned to the open data of non-specified format, while the highest rating to the data, which is additionally machine-readable and linked.

1	★	Available on the web (whatever format) but with an open licence, to be Open Data
2	★★	Available as machine-readable structured data (e.g. excel instead of image scan of a table)
3	★★★	as (2) plus non-proprietary format (e.g. CSV instead of excel)
4	★★★★	All the above plus, Use open standards from W3C (RDF and SPARQL) to identify things, so that people can point at your stuff
5	★★★★★	All the above, plus: Link your data to other people's data to provide context

Table 1: 5-star scheme for evaluating linked data usability. Based on the table at [11].

The idealistic vision of the semantic web movement is that all open data on the web becomes a huge open database, a giant global graph. „It's a linked information space in which data is being enriched and added. [12]" The resulting RDF files of this project meet these requirements: RDF graphs are not proprietary and available from web, RDF graphs are machine-readable, semantics of nodes and relations are described in widely known vocabularies and due to the nature of RDF graphs, they are linkable to other RDF graphs.

2.3 Resource Description Framework (RDF)

In 1997, the W3C defined the first Resource Description Framework specification [12, page 97]. It became a World Wide Web Consortium (W3C) recommendation by 1999 [12, page 97]. It is a directed graph-based data model. Its simple structure follows a semantic web standard, so-called triple. Each triple consists of subject, predicate and object, where predicate is a link between subject and object. The HTTP-based Uniform Resource Identifiers (URI) can uniquely identify/name each part in a triple. URI may be just an opaque identifier, a hyperlink to the resource of some entity, or a hyperlink to vocabulary of some metadata (e.g. a predicate's certain metadata is explained in a certain vocabulary). For objects, literals and blank nodes are also allowed. In [13] a list is given of how things are described with RDF:

1. Triples that describe a resource with literals;
2. Triples that describe a resource by linking to other resources (e.g., triples stating the resource's creator, or its type);
3. Triples that describe a resource by linking from other resources (i.e., incoming links);
4. Triples describing related resources (e.g., the name and maybe affiliation of the resource's creator);
5. Triples describing the description itself (i.e., data about the data, such as its provenance, date of collection, or licensing terms);
6. Triples about the broader data set of which this description is a part. [13]

Such a common data model enables machines to operate over all linked data sets.

The RDF metadata model in this project uses literals for extracted names and opaque URIs to them. It uses real URIs as identities to the web resources (real locations of documents) and URIs to vocabularies for describing nodes and relations.

For Python language, RDFlib [5] package exists for generating RDF graphs and this service's program also uses this package.

2.4 Vocabularies and Ontologies

The two terms often overlap in the articles of semantic web, but a W3C web page says that "The trend is to use the word "ontology" for more complex, and possibly quite formal collection of terms, whereas "vocabulary" is used when such a strict formalism is not necessarily used or only in a very loose sense." [14] Vocabularies are for defining and explaining concept of metadata terms used in RDF. In some RDF model, URI of metadata term dereferences to the vocabulary, where this particular term is explained. It is apparent that these vocabularies must be publicly accessible. Since the semantic web movement is young, only a few vocabularies are stable: some initiatives of creating of semantic web vocabularies have risen, some of them have fallen. As not a rear example, a book about linked data [13], issued 4 years ago in 2011, provides a list of widely used vocabularies, where the link to DOAP vocabulary, "http://trac.usefulinc.com/doap" has currently new address, "http://usefulinc.com/ns/doap#". When choosing some vocabulary, one must consider whether it will be supported and maintained in the future. Therefore a RDF publisher should choose a well-known vocabulary that is in widespread usage already. It increases the probability that applications can understand it and can easily integrate other linked data. Widely used vocabularies are for example 'Friend of a Friend' (FOAF) [15], „RDF Schema" (RDFs) [16], „Dublin Core" (DC) [17], „Semantically-Interlinked Online Communities" (SIOC) Core Ontology [18], „Web Ontology Language" (OWL) [19].

This project uses „Named Entity Recognition and Disambiguation“ (NERD) [20] ontology for unresolved extracted entities of names of people, organizations and locations. NERD has classes for Person, Organization and Location. Inside Person class, this system uses FOAF for describing full name, given name and family name.

When a RDF publisher does not find the vocabulary that exactly describes some metadata concept, then it should create a new one. The appropriate vocabulary may exist, but it is not always certain whether it will be supported in the future.

RDF publishers define their own vocabularies by combining DC and OWL.

Specifically, this system defines custom vocabulary NER and for describing its title, dataset properties and description, uses ontologies OWL, RDF and DC. In NER ontology two data type properties are described: “mentionedAtSite” and “lemma”. The first refers to a web source and the second to possible lemmas of recognized entity. The amount of data type properties inside a class may range from one to many. Lemmas are like alternatives of a name in this service.

NER defines and describes also two more types “locationName” and “orgName” for ensuring that these are names of unresolved[2] entities.

2.5 Sources for Linked Data

A lot of software is developed for converting data into RDF (see list at [21]). The source data may be unstructured (natural language), semi-structured (e.g. data in XML or JSON formats), or of special formats (iCalendar, Excel, Microformats), in relational databases. Applications (e.g. Lodifier [22], KnowledgeStore [23], Snowball [24], Open Calais [25], OntosLDIW [26], DBpedia Spotlight [27], this project), which extract entities and their relationships from the unstructured data make use of the named entity recognition (NER) techniques.

Semi-structured data, like in XML and Json formats, can be transformed into RDF, where several semantic transformation rules, that try to map e.g. XML schema to ontology structure, are used in order to convert semi-structured data into RDF structure. Some converters ([28], [29], [30]) use an approach that requires much effort from a developer: every different type of structure needs its own set of transformation rules programmed. Others ([31]) have come up with the idea of eliciting transformation rules automatically. DBpedia extracts the structured part of Wikipedia by converting Wiki markup language [32] into RDF. The mapping of relational database to RDF is described in detail in [33].

2.6 Extraction Tool Estnltk

Estnltk — Open Source Tools for Estonian Natural Language Processing.

Estnltk [http://tpetmanson.github.io/estnltk/index.html] is a core module of the system described in this thesis. Estnltk is developed by A. Tkachenko [34] and T.Petmanson [35]. With the help of Estnltk tool, the extraction of named entities of locations, organizations and people is made from Estonian natural language texts. This project uses Pyhton packages’ ner [36], tokenize [37] and morf [38] classes

- tokenize.Tokenizer();
- morf.PyVabamorfAnalyzer();
- ner.NerTagger().

The precision and recall of Estnltk work is presented in Table 2:

[2] Unresolved entity – extarcted entity from natural languuage, but it is not confirmed this entity (e.g. name of location) actually exists.

Best results for CRFs			Best results for MaxEnt		
	Precision	Recall		Precision	Recall
LOC	0.89	0.89	LOC	0.85	0.87
PER	0.89	0.89	PER	0.86	0.81
ORG	0.82	0.76	ORG	0.77	0.71

Table 2: Evaluation table of Estnltk, copied from [34, page 30]

2.7 *Storing Linked Data*

The linked data can be stored into RDF files (in several formats, like RDF/XML [39], Turtle, N-Triples [40] etc, and in special linked data databases, called triplestores.

The RDF graph files are queried with SPARQL [41] language; there are libraries for this in several programming languages, e.g. PHP [42], Python [43] etc. RDF files may grow very large and SPARQL query performance will become low. In that case the publisher should create rules for splitting large RDF into smaller fragments or store RDF graph triples into triple store.

Triple stores can be broadly classified into three types [44]: native triple stores, those that are implemented for labelled directed graph as RDF; triple stores that are built on relational database and NoSQL-based triple stores. Native and NoSQL databases are the best regarding the performance (becomes critical in case of huge amount of triple entries), because in relational databases SPARQL must be translated into SQL. The triple stores can process large (billions and trillions of triples [45]) RDF files with high performance. Many triple store implementations exist, from which widely known are e.g. Virtuoso [46], Oracle Spatial and Graph with Oracle Database [47], AllegroGraph [48], Stardog [49].

The RDF graphs in this service are split into three types: for organizations, locations and people, separately. As a result of processing over 3.1 million lines of log file, the RDF files reach approximately 6 Mb in size. Querying these large files are at the limit of patience: it takes approximately 15 seconds to get the SPARQL answer from such large files (although it also depends on the bandwidth). Therefore in future, some other storage method apart storage into files should be developed.

2.8 *Integrating data: interlinking with other linked data and mapping vocabularies*

Linked data publishers may integrate their own vocabularies to other, more widely known vocabularies in order to ensure machines can understand it. The more publishers' data are self-descriptive, the more easily consumers can integrate it with other Web data. OWL ontology and RDFs vocabulary have special predicates for mapping vocabularies, like owl:equivalentClass, rdfs:subClassOf, owl:equivalentProperty, rdfs:subPropertyOf.

For interlinking the same real world entity in other Web data, the publisher or consumer should use predicate's URI from OWL ontology: `http://www.w3.org/2002/07/owl#sameAs` (`sameAs`) for stating that the URI aliases refer to the same resource. Using `sameAs` is also known as a "data fusion" or "identity resolution" [50]. For example, if one linked data triple contains some URI that refers to literal `"Estonia"`, then in order to link it to DBpedia, a new triple should be added, in which the predicate is `http://www.w3.org/2002/07/owl#sameAs` and object is `http://dbpedia.org/page/Estonia`.

This thesis uses existing ontologies/vocabularies and does not use mapping predicates to other vocabularies, although in the NERD ontology, there is a list of ontologies which are mapped to NERD classes. This thesis also defines a few datatype properties for being more precise and letting consumer be aware that these RDF graphs are for linking extracted names with their source URLs and that these names may have alternatives. These names are not resolved. It means that it is not confirmed whether such a name really exists or is just a false positive result of Estnltk work. These RDF graphs are not linked to DBpedia or Wikipedia, but anyone can link these graphs to their own or any other necessary data.

2.9 Linked Data Use Cases

Linked data can be used in applications, web pages and search engines. It provides richer, intelligent information space for data consumer: for example, when searching for some organization, the linked data approach is able to provide links to opinions of people of this organization or links about the location of this organization. Another simple example: some tourist uses a smartphone an application of which delivers links to information about the current geographical location of the tourist. Some cases of usage and user stories about several life events are described in [51]. A long list of publicly accessible Linked Data infrastructures are given and described in [52, page 25].

These RDF graphs that are the products of this service, described in this thesis, can be used e.g. in applications to make them richer by providing links to the sources of names of people, organizations and locations that are included in the applications.

2.10 Google Cloud Platform

For thesis cloud service provider the author selected Google Compute Engine [6].

For storing metadata files and generated errors there is a Google Cloud Storage [53] for storing file's objects, designed to store extremely large amounts of relatively static data [54].

There is another option for storing Big Data, BigQuery [55]. The data is saved there into table and it uses a query language similar to SQL. Author considered selecting it in order to save more free space at master virtual machine (see chapter "Architecture"); and in BigQuery documentation [55] it is advertised as a low-cost and quick-access repository. However, it was simpler to let user to upload log file directly into master server, then read and parse it locally for posting URLs to worker servers, rather than communicating to BigQuery, primary intention of which is storage of large datasets for easier data mining.

There are several options for creating disks in Google Compute Engine (GCE) that vary in size of the volume, I/O speed and number of CPUs. For this project, the high-CPU disks and I/O are important for processing big files. Larger volumes can achieve higher I/O levels [56]. It is reasonable to use disks with maximum number of CPU (which is 8 in GCE during free trial period). With the master VM (see chapters 6.4, 6.5) created with 1 CPU, 7 CPUs are left over for worker VMs (see chapters 6.4, 6.5).

Author also tried Google's HTTP load balancing [57] with health check service [58] of virtual machine (VM) group. Load balancing is used for detecting the target (worker machine) that has the lowest current CPU usage %. However, it did not work out, maybe because of its current beta release.

Another option for parallel processing would be using Hadoop MapReduce, in which in the map function would be the workers' process (getting metadata, extracting entities, generating RDF graph files) and in the reduce function would be aggregation of generated RDF graphs. This option is yet unproven and is out of the scope of this project.

3 Related Work

In this chapter some linked data initiatives and applications, that make RDF graphs (software module that extracts the knowledge from unstructured and semi-structured files, files of special types or relational databases and maps to ontologies/vocabularies) are briefly described according to their similarities/differences with the aims of the current project.

3.1 DBpedia

DBpedia [http://wiki.dbpedia.org/] is a semantic web initiative that is widely known and interlinked (around 50 million RDF links to other datasets) It describes 38.3 million things [59] (counted things in all languages) and altogether stores 3 billion RDF triples. It does not cover only one topic, but spans multiple domains. It takes Wikipedia pages, finds semantically structured (Wikipedia uses its own Wiki markup language [32] for metadata) parts of it and translates into RDF. It declares mappings in 125 languages, no mapping is done in Estonian yet [60] [61]. The DBpedia's NER "DBpedia Spotlight" does not recognize Estonian locations, organizations and names. While writing this, the link to DBpedia Extraction Framework [wiki.dbpedia.org/Documentation] documentation does not work (it means author cannot compare it with the current project). DBpedia has currently over 15 projects for maintaining this crowd-sourced community effort, including descriptions of ontology, datasets statistics, how to interlink its data, community support etc.

3.2 Wordnet RDF

WordNet [http://wordnet.princeton.edu/wordnet/] is a large lexical database in English [62]. WordNet RDF [http://wordnet-rdf.princeton.edu/] is the RDF version of WordNet, created by mapping the existing WordNet data into RDF. The data is structured according to the lemon (The Lexicon Model for Ontologies) model [63]. In addition, links have been added from several lexicon sources [63]. Estonian WordNet [64] also exists and uses semantic descriptions for relations, but these are not in RDF graph format. The Estonian WordNet would be useful when resolving entities to other words than names, so currently there is no use for this system/service.

3.3 Yago

YAGO (Yet Another Great Ontology) [http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago//], another cross-domain data RDFizer, derives data from Wikipedia, WordNet and GeoNames [65]. Currently, YAGO has knowledge of more than 10 million entities (like persons, organizations, cities, etc.) and contains more than 120 million facts about these entities [65]. It does not extract entities from Estonian neither offers customization of extractor.

3.4 OpenCyc

OpenCyc [http://www.cyc.com/platform/opencyc/] offers its carefully designed ontology at no cost for research and can be used for rich domain modelling, semantic data integration, text understanding, domain-specific expert systems and game AIs [66]. It is developed by CyCorp, that states that OpenCyc is "*the world's largest and most complete general knowledge base and commonsense reasoning engine*" [66]. OpenCyc contains more than 500,000 Cyc terms [66]. The publisher also states that the platform includes in other modules the natural language parsers and CycL-to-English generation functions [66]. If developer wants to use it, he/she must request a free licence beforehand. Platform is developed in Java. It does nothing about extracting from documents written in Estonian.

3.5 Virtuoso Sponger

The Sponger provides a key platform for developers to generate quality data meshes from unstructured or semi-structured data [67]. It calls its RDFizers „cartridges“. Virtuoso has a set of prewritten domain-specific cartridges for e.g. Amazon, Digg, Flickr, Ebay, Technorati, RSS, Atom and many more [68], that reads metadata from the formats under question. Virtuoso Sponger RDFizer, given the URL to some document, starts from requesting RDF form (explicitly via HTTP Accept Headers), and if RDF is returned nothing further happens [69]. It then tries to find links to RDF documents, RDFa data, microformats, RSS and Atom feeds, web services. When something is found, it adds results to RDF graph by mapping metadata to appropriate vocabularies. Sponger also enables customizing of cartridges [70], but does not use any plain text processing tools, far from extracting entities from Estonian.

3.6 Freebase

Freebase [http://www.freebase.com/] is also community-curated database and spans cross-domain data, transforms Wikipedia into RDF-structured data and maps these to DBpedia. Most of its data is added by community members. In Spring 2015 it went into read-only state in order to develop new version of API [71]. It declares having over 48 million facts and it currently covers the topics of music, books, media, people, film, TV, business, location, government, science, arts and sports realms etc. There are links to Wikipedia pages in all given wikipage's possible languages in RDF data files, but its types are currently implemented only in English. It runs publicly since 2007 and was initiated by the American software company Metaweb. In 2010 it was acquired by Google [72], who currently maintains it and Google's Knowledgebase is partly powered with Freebase [73]. It does not extract entities from natural language.

3.7 The GeoNames Ontology

The GeoNames Ontology [http://www.geonames.org/ontology/documentation.html] makes it possible to add geospatial semantic information to the Word Wide Web [74]. All over 8.3 million geonames toponyms now have a unique URL with a corresponding RDF web service [74]. Other services describe the relation between toponyms [74]. The Ontology for GeoNames is available in OWL: http://www.geonames.org/ontology/ontology_v3.1.rdf, mappings [74]. GeoNames data is useful if hierarchical geospatial information is needed to integrate e.g. in applications. It can be linked to this project's location names, after the names are resolved or while resolving.

3.8 TWC LOGD

TWC LOGD [https://logd.tw.rpi.edu/home] projects translate government-related datasets into RDF, linking them to the Web of Data and providing demos and tutorials on mashing up and consuming linked government data [75]. It hosts 9,951,771,397 RDF triples. It has created 1,888 RDFized datasets originating from 119 sources and enhanced 1,686 RDFized datasets using 9,499 links to other LOD datasets [75]. TWC won 2nd prize at semantic web challenge [75]. Project "csv2rdf4lod" RDFizes data from comma-separated-values (CSV). It does not extract entities from natural language.

3.9 PoolParty

Poolparty [http://www.poolparty.biz/] is a proprietary application and sells services for enterprises. They advertise in their home page that their service provides "*precise text mining algorithms based on semantic knowledge models. With PPX, large amounts of documents and data records can be analysed in an automated fashion, extracting meaningful phrases, named entities, categories, sentiment values or other metadata nearly instantaneously* [76]". It uses Virtuoso software for storing. In its homepage is said that it imports Excel and RDF files, nothing is said about extracting names from natural language.

3.10 Metafacture

Metafacture [https://github.com/culturegraph/metafacture-core/blob/master/README.md] focuses mainly on semi-structured library data. It provides a versatile set of tools for reading, writing and transforming data [77]. It can be used as stand-alone application and also its java library is usable tool inside Eclipse environment as a Maven dependency. The library ontologies are not useful in this project's context. It differs technologically from this project by the development language and by having no functionality for extracting from texts written in Estonian.

3.11 Some Widely Known Applications

BBC [78] uses LOD to describe sports by relating events, discipline, athletes, etc. Similar to this project, it is hosted in cloud. The BBC has an unusually large amount of high value audio, video, images and text content spanning nearly 90 years [78]. Managing that content, managing the information within it and making sure that it is available to people throughout the organisation is a huge task [78]. The development of the linked data platform is a response to this demand [78]. This also ensures the usefulness of linked data: news files/information should be quickly accessible and modifiable. It does not have functionality for extracting from texts written in Estonian.

Europeana Linked Open Data [79] is an initiative for promoting more open data, also for machine-readings, similar to current project. It also has a datasets repository, SPARQL endpoint, REST API. It links together different data providers' data and thereby enhances the presentation of data to users. It does not have functionality for extracting from texts written in Estonian.

European Commission. Health and food safety [80] manages non-relational data using semantic graphs data management technologies. It also have understood significance of semantic web: „*The main benefits of exposing data semantically are: (a) the capacity to link information; (b) the automatic guaranteeing of the consistency of linked data.*" [81]

To sum up, there is no system for RDFizing entities from Estonian OGD. This thesis describes this kind of system.

3.12 Data Harvesting. File/document repositories.

Many LOGD applications use CKAN [82] for storing and harvesting open datasets, for cataloguing, searching and displaying data: Government of United Kingdom [83], Europe's Public data [84], Helsinki Region Infoshare online service [85], International Aid Transparency Initiative [86]. While planning this project there was a intention to store datasets also with CKAN application. But during reading CKAN documentation in December 2014, it appeared, that documents cannot be stored remotely over URL, but only locally ("*we have simplified this to only allow local file uploads*" [87]). To avoid double storage, author decided to download documents' contents locally into a file system directory.

There is a buckets' system in Google Cloud Storage for storing files and perform authorized access to these from GCE virtual machines (VMs). The cloud of virtual machines in this service save metadata of processed documents into certain structure using json format. These json-formatted files are stored in GCE buckets. After RDFizing process is finished, these json files are used as an information base for documents archivation. Archivation means downloading these documents into master VM (master VM is like a 'manager' of several 'worker' VMs). Then these metadata-json-files are also downloaded into master's local file system in order to speed up browsing of archived datasets: these metadata-json-files enable faceted search/browsing by filtering data based on its rich metadata (e.g. content type) and the sequence of characters in the file name.

4 Metadata Model

In this chapter the metadata model of accessed documents is explained.

The metadata creation is driven by the functional requirements with identifiers "genMD", "updateMD", "readMD", "updateRDF", "downD" and "readD" (see subchapter "Functional Requirements" of chapter "Architecture").

Document's metadata and its accessing time is saved into json-formatted file. It is used in all parts of the system as an auxiliary file. The structure of metadata is saved in a way that enables downloading documents, browsing downloaded documents, performing monthly updates. During every RDFizing process and during updating process, these metadata structures are used for avoiding duplicate analysing of the same unchanged content.

It is first used when a document is opened at its URL and the document content together with its metadata are read into memory stream. Its name is host name, extracted from the URL: metadata of all documents from the same host are saved into the same json-formatted file. For example, two files with URLs '`http://www.example.ee/doc1.pdf`' and '`http://www.example.ee/doc2.xsl`' are saved into same file with name '`www.example.ee.json`'. Firstly, the URL is used to extract a host name and the host name is then used to check whether the json-file with this host name already exists. **(A) IF THE JSON-FORMATTED FILE <u>DOES NOT EXIST</u> YET**, the information in memory stream is used to create one. This file naming strategy enables fast finding of the right information. The created json-formatted file contains two types of objects at first level: the basic URL (e.g. '`http://www.example.ee/`') and the hash value of URL of document (e.g. sha224(`http://www.example.ee/doc1.pdf`')). The hash value strategy is used for two reasons:

1. When a program opens json-formatted file, it wants to know whether this document at this URL has already been used for RDFizing earlier. Comparing hash values is a fast way to find it out.
2. The hash value is used as file name during the archival process: the documents are downloaded into master VM file system after RDFizing process and naming file with its actual URL may issue 'too-long file-name' error.

One json-formatted file can contain one or many such hashes, according to the number of different documents used in RDFizing process from the same host.

These hashes are also keys to the metadata information of the document. This information additionally contains the date that marks the date when this document was used in RDFizing process.

An example of this json-formatted file is shown in Figure 1. Its name is "`metsakontor.ee.json`", extracted from URL "`http://metsakontor.ee/`". It has two types of first-level keys, marked with yellow background: "base_url" and hash of document URL. Hash of URL is a key to the second level key (marked with green background) which is hash of document's content as it was at the time of RDfizing process.

Figure 1: Example of json-formatted metadata file named “metsakontor.ee.json”.

```
{
  "base_url": "metsakontor.ee",
  "3f697899213213baa36f53d0096eaaee63a9d51ad119809e5aea1a00": {
    "4a3ddc515218f2dbfdc0bb1039c4caabd7e6b168e905057b70f936b6": {
      "status": 200,
      "Content-Length": "2300",
      "Content-Encoding": "gzip",
      "sha224":
"4a3ddc515218f2dbfdc0bb1039c4caabd7e6b168e905057b70f936b6",
      "Vary": "Accept-Encoding,User-Agent",
      "X-Powered-By": "PHP/5.3.28",
      "file_url": "http://metsakontor.ee/?Kus_me_asume...&login",
      "Keep-Alive": "timeout=1, max=100",
      "Server": "Apache/2",
      "localFilename":
"3f697899213213baa36f53d0096eaaee63a9d51ad119809e5aea1a00",
      "Connection": "Keep-Alive",
      "Date": "Fri, 07 Aug 2015 12:35:08 GMT",
      "timeDir": "07_08_2015",
      "Content-Type": "text/html"
    }
  },
  "256855198c9694ef0553ba1f3821098ee90f6a30e2923287ee857d66": {
    "73cb7db500ad650624e9d1532fe7a8cf594163cbc3f1b71b94fbcbfa": {
      "status": 200,
      "Content-Length": "5245",
      "X-Powered-By": "PHP/5.3.28",
      "sha224":
"73cb7db500ad650624e9d1532fe7a8cf594163cbc3f1b71b94fbcbfa",
      "Vary": "Accept-Encoding,User-Agent",
      "Content-Encoding": "gzip",
      "file_url": "http://metsakontor.ee/?Metsakontor&print",
      "Keep-Alive": "timeout=1, max=100",
      "timeDir": "07_08_2015",
      "localFilename":
"256855198c9694ef0553ba1f3821098ee90f6a30e2923287ee857d66",
      "Connection": "Keep-Alive",
      "Date": "Fri, 07 Aug 2015 11:39:50 GMT",
      "Server": "Apache/2",
      "Content-Type": "text/html"
    }
  },
  <another hash of URL>{
       <another hash of content>
    ...
  }
}
```

When the document together with its metadata have been read into memory stream and checked whether the json-file with this host name exists already and **(B) IF THE JSON-FORMATTED FILE FOR THIS HOST NAME EXISTS**, the next step is finding out whether the hash of document's URL exists**. (B1) If the hash of document's URL does not exist**, 'new part', representing given document, is added into existing json-formatted file and the json-file is updated in file storage. This 'new part' means that it has a

- first level key: hash of document URL;
    - second level key: hash of document's contents at the time it was opened and read during RDFizing process;

      and under second level's key there is an information that contains key-value pairs of
        - document's metadata (like content type etc),
        - the date when it was accessed during RDFizing ,
        - hash of file URL,
        - hash of document's content.

(B2) If the hash of document's URL exists, the next step is to check whether the content of the existing document have changed**. (B2a)** ***If the content of a existing document have not changed***, then the information and contents of this document is not used in RDFizing process. **(B2b)** ***If the content of a existing document have changed***, then the new hash of document's content is calculated and the RDFizing process continues with this changed content.

4.1 Using Metadata Model for Downloading Datasets

This json-file is used as information base during downloading process of documents. The downloading process of documents starts, when the RDFizing process have finished. The json-formatted files are stored in GCE Cloud Storage. The parts necessary for downloading are the actual URL and the date which are saved under the mentined second-level key (hash of file content). The document is downloaded from that URL and stored into master VM folder for downloaded files, into subfolder with the name that is derived from that mentined date. For example, if the date of RDFizing of this document was 15/05/2015 (in a format dd/mm/yyyy), the name of a folder becomes 15_05_2015. The date information is used also before the downloading of this document starts to avoid overwritting older documents that were downloaded earlier. The program knows the date when RDFizing process started and downloads only these documents which date in json-formatted file is 'smaller' or equal to the current date.

The structure of file system of downloaded documents' is as follows:

- <date>/<host name>/< local file name (hash of document's URL)>
- <date>/<host name>/< another local file name>
- <date>/<another host name>/< yet another local file name>
- <another date>/<yet another host name>/< yet yet another local filename>

4.2 Using Metadata Model in Updating Process

These metadata json-formatted files are downloaded into master VM, after RDFizing process and downloading process have finished. Updating process uses these json-formatted files as knowledge base for

- getting URLs of documents

- comparing second-level keys (hash of document contents) to the current hash of document's content. If the conetnt hashes are different, it means that document's content have changed and it is necessary to start new RDFizing process in order to find new named entitties and update RDF files. Next step in this case is **(B2b)** (explained at the start of this chapter).

4.3 Using Metadata Model for Browsing Datasets

The information that is saved into these json-formatted metadata files are a knowledge base for displaying downloaded datasets to users:

- it gets and displays the actual URLs of the documents (key-value pairs under the second level key (hash of file content)),
- it displays also path to locally archived document (using date and local file name (hash of document's URL))
- it can filter and sort datasets by the document's metadata: e.g. by content type, accessing date and so on. Currently, in this service, the filtering of documents by their content type is implemented.

Few examples of the downloaded documents' local paths would be

- 12_05_2015/arileht.delfi.ee/d5fbf9cf5907ebc8fa88ce8950fa6108e582da92c5ed462348b34d3b
- 12_05_2015/arileht.delfi.ee/d78992ca7da86f09cd0a264b464b26d39be0891441c2b1bd2aea5838

That follows the structure
<date>/<host name>/<local file name (hash of document's URL)>

5 RDF Data Model

In this chapter the structure of RDF graphs' data model is explained.

5.1 Used Vocabularies/Ontologies

The vocabularies used in RDF graphs are RDF, NERD, FOAF, OWL, DC and NER. This service uses RDF/XML format for RDF graphs.
Prefixes inside RDF files look as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:ner="http://www.estner.ee/estnerd#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:nerd="http://nerd.eurecom.fr/ontology#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
>
</rdf:RDF>
```

- NERD [20]
This ontology represents recognised named entities. In this service's RDF graphs are used its classes
    - **nerd:Location**
    - **nerd:Organization**
    - **nerd:Person**

- RDF
This ontology is always used in RDF graphs

- FOAF
FOAF is used in order to be more precise inside nerd:Person class. Its terms
foaf:gname is used for saving person's given names
foaf:fname is used for saving person's family name
foaf: name is used to save persons full name

Example of using FOAF inside NERD class Person:

```
 <nerd:Person rdf:about="http://nerd.eurecom.fr/ontology#katrin_paio">
    <ner:mentionedAtSite rdf:resource="http://online.le.ee/2014/02/"/>
    <foaf:givenName>Katrin</foaf:givenName>
    <foaf:familyName>Paio</foaf:familyName>
    <ner:lemma>katrin paio</ner:lemma>
    <foaf:name>Katrin Paio</foaf:name>
 </nerd:Person>
```

- OWL and NER

owl:Ontology is used for defining own custom ontology in order for being more precise. Its namespace is ‘`http://www.estner.ee/estnerd#`’ (with prefix ner). It is used inside NERD classes of Location and Organization for precisely defining that we deal with the term **name** inside classes nerd:Organization or nerd:Location. These are

- **ner:locationName** and

with namespace `http://www.estner.ee/estnerd#locationName`

- **ner:orgName**.

With namespace `http://www.estner.ee/estnerd#orgName`

In this ner ontology are also properties that describe that some name has found from some web site

- **ner:mentionedAtSite**

with namespace `http://www.estner.ee/estnerd#mentionedAtSite`

and another property

- **ner:lemma** for describing that certain recognized name entity has alternatives .

with namespace `http://www.estner.ee/estnerd#lemma`

The latter is necessary, because Estonian words may be written in over 14 cases. Two or more formed words may have same shape, but different meaning and lemma. The property for alternatives is also used for saving the same word in upper-case and lower-case for simplifying work of SPARQL query.

This ner ontology inside RDF graph looks as follows (in RDF/XML format):

```
<owl:Ontology rdf:about="http://www.estner.ee/estnerd#">
    <dc:title>estNERD Ontology</dc:title>
      <dc:description>Locations, organizations and persons extracted
      from Estonian open data, using Estnltk – Open source tools for
      Estonian natural language processing
      [http://tpetmanson.github.io/estnltk/].</dc:description>
  </owl:Ontology>
<owl:DatatypeProperty rdf:about="http://www.estner.ee/estnerd#orgName">
      <dc:description>Unresolved name of organization. Exctarcted using
      Estnltk – Open source tools for Estonian natural language
      processing.
      [http://tpetmanson.github.io/estnltk/].</dc:description>
</ owl:DatatypeProperty >
<owl:DatatypeProperty rdf:about="http://www.estner.ee/estnerd#lemma">
      <dc:description>Alternative for unresolved name; exctarcted using
      Estnltk – Open source tools for Estonian natural language
      processing.</dc:description>
</ owl:DatatypeProperty >
<owl:DatatypeProperty
rdf:about="http://www.estner.ee/estnerd#mentionedAtSite">
      <dc:description>Web site, where unresolved name was found using
      Estnltk – Open source tools for Estonian natural language
      processing.</dc:description>
</ owl:DatatypeProperty >
```

- DC

dc:description term is used for describing the ner ontology and its terms ‘lemma’, ‘mentionedAtSite’, ‘locationName’ and ‘orgName’. [http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=elements#description]
dc:title term is used for writing title for ner ontology [http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=elements#title]

5.2 Example of RDF Graph for Organizations

RDF graph file for organizations has name ORG.rdf. Here is an example of one organization in this file:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:ner="http://www.estner.ee/estnerd#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:nerd="http://nerd.eurecom.fr/ontology#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
>
  <nerd:Organization
rdf:about="http://nerd.eurecom.fr/ontology#megastarile">
    <ner:orgName>Megastarile</ner:orgName>
    <ner:lemma>megastarile</ner:lemma>
    <ner:lemma>Megastar</ner:lemma>
    <ner:lemma>Megastarile</ner:lemma>
    <ner:lemma>Megastari</ner:lemma>
    <ner:mentionedAtSite
rdf:resource="http://www.megastar.ee/privaatsus"/>
  </nerd:Organization>
<owl:Ontology rdf:about="http://www.estner.ee/estnerd#">
    <dc:title>estNERD Ontology</dc:title>
    <dc:description>Locations, organizations and persons extracted from
     Estonian open data, using Estnltk – Open source tools for Estonian
     natural language processing
     [http://tpetmanson.github.io/estnltk/].</dc:description>
  </owl:Ontology>
<owl:DatatypeProperty rdf:about="http://www.estner.ee/estnerd#orgName">
    <dc:description>Unresolved name of organization. Exctarcted using
    Estnltk – Open source tools for Estonian natural language
    processing.
    [http://tpetmanson.github.io/estnltk/].</dc:description>
</ owl:DatatypeProperty >
<owl:DatatypeProperty rdf:about="http://www.estner.ee/estnerd#lemma">
    <dc:description>Alternative for unresolved name; exctarcted using
Estnltk – Open source tools for Estonian natural language
processing.</dc:description>
</ owl:DatatypeProperty >
<owl:DatatypeProperty
rdf:about="http://www.estner.ee/estnerd#mentionedAtSite">
```

```
    <dc:description>Web site, where unresolved name was found using
    Estnltk – Open source tools for Estonian natural language
    processing.</dc:description>
</ owl:DatatypeProperty >
</rdf:RDF>
```

5.3 Example of RDF Graph for Locations

RDF graph file for locations has name LOC.rdf. Here is an example of one location in this file:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:ner="http://www.estner.ee/estnerd#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:nerd="http://nerd.eurecom.fr/ontology#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
>
  <nerd:Location rdf:about="http://nerd.eurecom.fr/ontology#võru">
    <ner:mentionedAtSite
rdf:resource="http://www.rantell.ee/2012/11/"/>
    <ner:lemma>Võru</ner:lemma>
    <ner:locationName>Võru</ner:locationName>
    <ner:lemma>võru</ner:lemma>
    <ner:mentionedAtSite rdf:resource="http://www.rantell.ee/2012/"/>
    <ner:mentionedAtSite
rdf:resource="http://www.rantell.ee/author/admin/feed/"/>
  </nerd:Location>
<owl:Ontology rdf:about="http://www.estner.ee/estnerd#">
    <dc:title>estNERD Ontology</dc:title>
    <dc:description>Locations, organizations and persons extracted from
Estonian open data, using Estnltk – Open source tools for Estonian
natural language processing
[http://tpetmanson.github.io/estnltk/].</dc:description>
  </owl:Ontology>
<owl:DatatypeProperty
rdf:about="http://www.estner.ee/estnerd#locationName">
    <dc:description>Unresolved name of location. Exctarcted using
Estnltk – Open source tools for Estonian natural language processing
[http://tpetmanson.github.io/estnltk/].</dc:description>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.estner.ee/estnerd#lemma">
    <dc:description>Alternative for unresolved name; exctarcted using
Estnltk – Open source tools for Estonian natural language
processing.</dc:description>
</owl:DatatypeProperty >
<owl:DatatypeProperty
rdf:about="http://www.estner.ee/estnerd#mentionedAtSite">
```

```
    <dc:description>Web site, where unresolved name was found using
Estnltk – Open source tools for Estonian natural language
processing.</dc:description>
</owl:DatatypeProperty >
</rdf:RDF>
```

5.4 Example of RDF Graph for People

RDF graph file for people has name PER.rdf. Here is an example of one person in this file:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:ner="http://www.estner.ee/estnerd#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:nerd="http://nerd.eurecom.fr/ontology#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
>
  <nerd:Person rdf:about="http://nerd.eurecom.fr/ontology#katrin_paio">
    <ner:mentionedAtSite rdf:resource="http://online.le.ee/2014/02/"/>
    <foaf:givenName>Katrin</foaf:givenName>
    <foaf:familyName>Paio</foaf:familyName>
    <ner:lemma>katrin paio</ner:lemma>
    <foaf:name>Katrin Paio</foaf:name>
  </nerd:Person>
<owl:Ontology rdf:about="http://www.estner.ee/estnerd#">
    <dc:title>estNERD Ontology</dc:title>
    <dc:description>Locations, organizations and persons extracted from
Estonian open data, using 'Estnltk – Open source tools for Estonian
natural language processing'
[http://tpetmanson.github.io/estnltk/].</dc:description>
  </owl:Ontology>
<owl:DatatypeProperty rdf:about="http://www.estner.ee/estnerd#lemma">
    <dc:description>Alternative for unresolved name; exctarcted using
Estnltk – Open source tools for Estonian natural language
processing.</dc:description>
</ owl:DatatypeProperty >
<owl:DatatypeProperty
rdf:about="http://www.estner.ee/estnerd#mentionedAtSite">
    <dc:description>Web site, where unresolved name was found using
Estnltk – Open source tools for Estonian natural language
processing.</dc:description>
</ owl:DatatypeProperty >
</rdf:RDF>
```

5.5 Example of Triples in Locations Graph

Here for giving another viewpoint the triples of a location graph are shown in Table 3:

Nr	Subject	Predicate	Object
1	http://www.estner.ee/estnerd#	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Ontology
2	http://www.estner.ee/estnerd#	http://purl.org/dc/elements/1.1/description	"Locations, organizations and persons extracted from Estonian open data, using Estnltk Open source tools for Estonian natural language processing' [http://tpetmanson.github.io/estnltk/]."
3	http://www.estner.ee/estnerd#	http://purl.org/dc/elements/1.1/title	"estNERD Ontology"
4	http://nerd.eurecom.fr/ontology#sillamäe	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://nerd.eurecom.fr/ontology#Location
5	http://nerd.eurecom.fr/ontology#sillamäe	http://www.estner.ee/estnerd#mentionedAtSite	http://www.christiansen.ee/as-bct-labis-edukalt-integreeritud-juhtimissusteemi-valisauditid
6	http://nerd.eurecom.fr/ontology#sillamäe	http://www.estner.ee/estnerd#locationName	"Sillamäe"
7	http://nerd.eurecom.fr/ontology#sillamäe	http://www.estner.ee/estnerd#lemma	"sillamäe"
8	http://nerd.eurecom.fr/ontology#sillamäe	http://www.estner.ee/estnerd#lemma	"Sillamäe"
9	http://www.estner.ee/estnerd#locationName	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#DatatypeProperty
10	http://www.estner.ee/estnerd#locationName	http://purl.org/dc/elements/1.1/description	"Unresolved name of location. Exctarcted using Estnltk — Open source tools for Estonian natural language processing [http://tpetmanson.github.io/estnltk/]."
11	http://www.estner.ee/estnerd#lemma	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#DatatypeProperty
12	http://www.estner.ee/estnerd#lemma	http://purl.org/dc/elements/1.1/description	"Alternative for unresolved name; exctarcted using Estnltk — Open source tools for Estonian natural language processing."
13	http://www.estner.ee/estnerd#mentionedAtSite	http://purl.org/dc/elements/1.1/description	"Web site, where unresolved name was found using Estnltk — Open source tools for Estonian natural language processing."

Table 3: Triples of the Data Model of a Location RDF graph.

6 Architecture

In the first subchapter the selected programming languages are reasoned. In the second subchapter the functional and non-functional requirements are presented.

In the next subchapters the presented architecture follows 4+1 architecture guidelines of [88] in a broad way: it includes

- physical view of components in a cloud and how they communicate;
- development view shows what software part is in which physical component and what messages they send;
- in logical view no class relationships are shown, because in this service data flows sequentially like in pipeline and in parallel CPUs and worker VMs. This data flow is shown instead;
- process view shows what decision points are made in this data flow;
- potential use case scenarios of what kind of problems this system should solve.

In selecting these diagram types for each view, the transparency [89] of the system is born in mind. For complementary information, textual explanations are added to the diagrams.

As described in subchapters 6.4 and 6.5, the system mainly consists of two parts: master and group (cloud) of workers. The architecture is tested in the Google Compute Engine [6] environment and uses Google Compute Engine (GCE) API tools [90] for creating master server and virtual machines in the cloud of worker instances (see Figure 2).

This architecture does not include solutions for security and authentication of users, this topic is out of the thesis' scope, the focus is on developing the architecture and software for strictly solving RDFizing problem for Estonian OGD and achieving it with the service performance as high as possible.

6.1 Programming Languages

Python language versions 2.7 and 3.4 are both used. Python 2.7 is used in communicating with Google Cloud API [90], because it does not support newer versions. Python 3.4 is chosen for programming all other parts, because there is a simple python package „RDFlib“[5] for creating, parsing, serializing and querying RDF graphs. There is a multiprocessing package in Python, which is a simple tool for creating pool of processes. This pool is used in worker VMs (see subchapter 6.4) for forking URLs (from the incoming list of URLs) into several processes. Using multiple processes in parallel helps to speed up the performance. Next important reason is that Estnltk [4] is also developed in Python.

PHP is another used language, because it is simple to handle POST and GET requests; also to call Python modules.

6.2 Migrating Software to GCE

The installation of the required packages and dependences is described in Appendix A: Installation. Also, installation guidelines into master is at https://github.com/Mailis/EstNer/tree/master/cloud/master/readme.md. And installation into worker is described in https://github.com/Mailis/EstNer/tree/master/cloud/worker/readme.md.

Documentation is at https://github.com/Mailis/EstNer/tree/master/cloud/readme.md.

Setting project up in GCE, is described in https://cloud.google.com/compute/docs/projects.

Then create 2 instances of VMs: 1st for master and 2nd for worker. The instances should be high-CPU (GCE notation) with as many CPUs as possible. Use large persistent disk, because the larger the disk, the faster the I/O operations, in order to speed up the performance.

To enable networking, allow HTTP and HTTPS traffics for both instance types.

Enable permissions for both instance types as follows:

User info - Enabled

Compute - Read Write
Storage - Full
Task queue - Disabled
BigQuery -Disabled
Cloud SQL -Disabled
Cloud Datastore - Enabled
Cloud Logging API - Full
Cloud Monitoring API - Disabled
Cloud Platform - Enabled
Bigtable Data - Disabled
Bigtable Admin - Disabled

Example of creating master instance, using REST [91]

```
{
  "kind": "compute#instance",
  "id": "16567739923284107723",
  "creationTimestamp": "2015-07-31T15:38:36.696-07:00",
  "zone": "https://www.googleapis.com/compute/v1/projects/estn-
1006/zones/europe-west1-b",
  "status": "RUNNING",
  "name": "master",
  "description": "",
  "tags": {
    "items": [
      "http-server",
      "https-server"
    ],
    "fingerprint": "6smc4R4d39I="
  },
  "machineType": "https://www.googleapis.com/compute/v1/projects/estn-
1006/zones/europe-west1-b/machineTypes/n1-standard-1",
  "canIpForward": true,
  "networkInterfaces": [
    {
      "network": "https://www.googleapis.com/compute/v1/projects/estn-
1006/global/networks/default",
      "networkIP": "10.240.121.245",
      "name": "nic0",
      "accessConfigs": [
        {
          "kind": "compute#accessConfig",
          "type": "ONE_TO_ONE_NAT",
          "name": "External NAT",
          "natIP": "104.155.42.131"
        }
      ]
    }
  ],
  "disks": [
    {
      "kind": "compute#attachedDisk",
      "index": 0,
      "type": "PERSISTENT",
      "mode": "READ_WRITE",
      "source": "https://www.googleapis.com/compute/v1/projects/estn-
1006/zones/europe-west1-b/disks/master",
      "deviceName": "master",
      "boot": true,
      "autoDelete": true,
      "licenses": [
        "https://www.googleapis.com/compute/v1/projects/ubuntu-os-
cloud/global/licenses/ubuntu-1504-vivid"
      ],
      "interface": "SCSI"
    }
  ],
  "metadata": {
    "kind": "compute#metadata",
    "fingerprint": "4S8uYwkS5dM="
  },
  "serviceAccounts": [
    {
      "email": "662266012403-compute@developer.gserviceaccount.com",
      "scopes": [
        "https://www.googleapis.com/auth/cloud-platform",
        "https://www.googleapis.com/auth/compute",
        "https://www.googleapis.com/auth/datastore",
        "https://www.googleapis.com/auth/devstorage.full_control",
        "https://www.googleapis.com/auth/logging.admin",
        "https://www.googleapis.com/auth/userinfo.email"
      ]
    }
  ],
  "selfLink": "https://www.googleapis.com/compute/v1/projects/estn-
1006/zones/europe-west1-b/instances/master",
  "scheduling": {
    "onHostMaintenance": "MIGRATE",
    "automaticRestart": true,
    "preemptible": false
  },
  "cpuPlatform": "Intel Sandy Bridge"
}
```

Example of creating worker instance, using REST [91]

```
{
  "kind": "compute#instance",
  "id": "14911290095909708907",
  "creationTimestamp": "2015-08-06T06:59:19.766-07:00",
  "zone": "https://www.googleapis.com/compute/v1/projects/estn-
1006/zones/europe-west1-b",
  "status": "RUNNING",
  "name": "worker1",
  "description": "",
  "tags": {
    "items": [
      "http-server",
      "https-server"
    ],
    "fingerprint": "6smc4R4d39I="
  },
  "machineType": "https://www.googleapis.com/compute/v1/projects/estn-
1006/zones/europe-west1-b/machineTypes/n1-highcpu-2",
  "canIpForward": true,
  "networkInterfaces": [
    {
      "network": "https://www.googleapis.com/compute/v1/projects/estn-
1006/global/networks/default",
      "networkIP": "10.240.124.124",
      "name": "nic0",
      "accessConfigs": [
        {
          "kind": "compute#accessConfig",
          "type": "ONE_TO_ONE_NAT",
          "name": "External NAT",
          "natIP": "104.155.76.242"
        }
      ]
    }
  ],
  "disks": [
    {
      "kind": "compute#attachedDisk",
      "index": 0,
      "type": "PERSISTENT",
      "mode": "READ_WRITE",
      "source": "https://www.googleapis.com/compute/v1/projects/estn-
1006/zones/europe-west1-b/disks/worker1",
      "deviceName": "worker1",
      "boot": true,
      "autoDelete": true,
      "licenses": [
        "https://www.googleapis.com/compute/v1/projects/ubuntu-os-
cloud/global/licenses/ubuntu-1504-vivid"
      ],
      "interface": "SCSI"
    }
  ],
  "metadata": {
    "kind": "compute#metadata",
    "fingerprint": "4S8uYwkS5dM="
  },
  "serviceAccounts": [
    {
      "email": "662266012403-compute@developer.gserviceaccount.com",
      "scopes": [
        "https://www.googleapis.com/auth/cloud-platform",
        "https://www.googleapis.com/auth/compute",
        "https://www.googleapis.com/auth/datastore",
          "https://www.googleapis.com/auth/devstorage.full_control",
          "https://www.googleapis.com/auth/logging.admin",
          "https://www.googleapis.com/auth/userinfo.email"
        ]
      }
    ],
    "selfLink": "https://www.googleapis.com/compute/v1/projects/estn-
1006/zones/europe-west1-b/instances/worker1",
    "scheduling": {
      "onHostMaintenance": "MIGRATE",
      "automaticRestart": true,
      "preemptible": false
    },
    "cpuPlatform": "Intel Sandy Bridge"
  }
```

If instances are created, open the SSH's command prompts of these instances and follow installation guidelines. Before uploading software into instances, change the project name "`estn-1006`" of variable **`PROJECT_ID`** `= "estn-1006"` to your project's name. This must be changed in two files:

- in master server it is `upload_logfile/commonVariables.py` and
- in worker server it is `storage/commonvariables.py`.

When installation is ready, create snapshots of these instances. Then it is possible to delete VMs and create quickly from snapshots again. When creating master VM, put the name "`master`" to it. If you want to use other name for master, change the variable **`MASTERINSTANCE_NAME`** `= "master"` name in both,

- in the master VM `upload_logfile/commonVariables.py`
- and in the worker `VM storage/commonvariables.py.`

Also, if you use other GCE zone than "`europe-west1-b`", change the variable value **`DEFAULT_ZONE`** `= 'europe-west1-b'` in these same files.

Open GCE Cloud Storage [53] browser (using GCE console for developers) and create new bucket with name "`datadownload_jsons`" (for storing metadata json-formatted files) and another with name "`generated_files`" (for storing errors). If you want to use other names for buckets, change the respective names in these files

(in master server it is

- `upload_logfile/commonVariables.py`

and in worker server it is

- `storage/commonvariables.py`).

Demo version is at address during August, 2015: http://104.155.42.131/ .

6.3 *Requirements*

In this chapter the functional and nonfunctional requirements are described.

6.3.1Functional Requirements

In Table 4, functional requirements are shown. Basic requirements are **upLOG**, **genRDF**, **updateRDF**, **downD**, **readRDFm**, **readRDFh** and **readD**. Requirements related to metadata (see chapter 4, “Metadata Model”) are derived from basic requirements, because these (**updateRDF**, **downD** and **readD**, **readRDFm**) are not possible to fulfil without metadata ones. Requirements for developers (**readE**, **genE**) are important from developers point of view, mainly for testing purposes during development cycles. “States before act” are prerequisites. These prerequisites are fulfilled in other requirements’ “state after act”, except in **upLOG**. Related states are depicted with the same background colour in this Table 4.

Short overview of requirements (id of requirement is in bold, see Table 4):

upLOG
Human user can upload log file using graphical user interface (GUI).

genRDF
Generating RDF files: human user can push a button, using GUI, to start RDFizing process.

updateRDF
System should start monthly updates on a datetime that is programmed by developer. The starter is master VM.

downD
Download analyzed-RDFized documents. It starts automatically after RDFizing process is finished.

readD
Human can read and browse downloaded documents (aka datasets).

readRDFm
Machine can query RDF files and receive response using SPARQL endpoint webservices.

readRDFh
Humans can also query RDF files and receive response, using GUI for SPARQL endpoint.

genMD
Generating metadata derives from requirements **updateRDF** , **downD**, **readD**.

updateMD
Updating metadata derives from requirement **updateRDF**.

readMD
Reading metadata derives from requirement **readD**, **updateRDF** , **downD**, **readD**.

readE
Developer of this system should be aware of the thrown errors in order to make better software.

genE
For debugging and testing of software, developer must see thrown errors, errors must be stored. This derives from requirement **readE**.

id	Act	actor(s)	state before act	state after act
upLOG	upload log file	human user	new log file is not uploaded	new logfile is uploaded
genRDF	generate RDF graph nodes from log file, save them into rdf-files	**human user (act starter)**, master VM, worker VMs, GCE Storage	log file is uploaded; *graphs from this log file are not created or need to update; RDF graphs in master VM may or may not exist*	RDF graph nodes are created and saved into RDF-files `LOC.rdf. ORG.rdf, PER.rdf` in master VM, metadata in GCE Storage are created or updated; metadata are downloaded to master VM
updateRDF	update RDF graphs and datasets archive	**master VM (act starter)**, worker VMs, GCE Storage	metadata in GCE Storage must exist; *RDF graphs in master VM may or may not exist*	RDF graphs are updated in master VM; metadata in GCE Storage are updated; metadata are downloaded to master VM documents in archive are updated and downloaded into master VM
downD	download documents	**master server (act starter)**, GCE Storage	metadata in GCE Storage must exist; *documents may or may not exist in master VM*	documents in archive are downloaded into master VM
readD	read documents	**human user (act starter)**, master VM	documents are downloaded into master VM; metadata are downloaded to master VM	documents can be browsed, displayed and read
readRDFm	read RDF graphs	**machine (act starter)**, master VM	RDF graph nodes are created and saved into RDF-files `LOC.rdf. ORG.rdf, PER.rdf` in master VM; SPARQL endpoint webservices exist; valid query sentence is sent using POST method	query (using POST method) to web service received, response of web service is delivered
readRDFh	read RDF graphs	**human user (act starter)**, master VM	RDF graph nodes are created and saved into `RDF-files LOC.rdf. ORG.rdf, PER.rdf` in master VM; SPARQL endpoint GUI exists	graph is read
genMD	generate metadata	worker VMs	**genRDF** or **updateRDF** has started	metadata is read and stored in certain structure; metadata in GCE Storage exist;
updateMD	update metadata	worker VMs	metadata in GCE Storage must exist;	metadata in GCE Storage are updated; metadata in GCE Storage exist;
readMD	read metadata	worker VMs, master VM	metadata are downloaded to master VM	metadata is read

id	Act	actor(s)	state before act	state after act
genE	generate error and store it	master VM, worker VMs	error is thrown inside system	error is stored
readE	read generated errors	system developer (author)	error is stored	error is read

Table 4: Functional requirements.

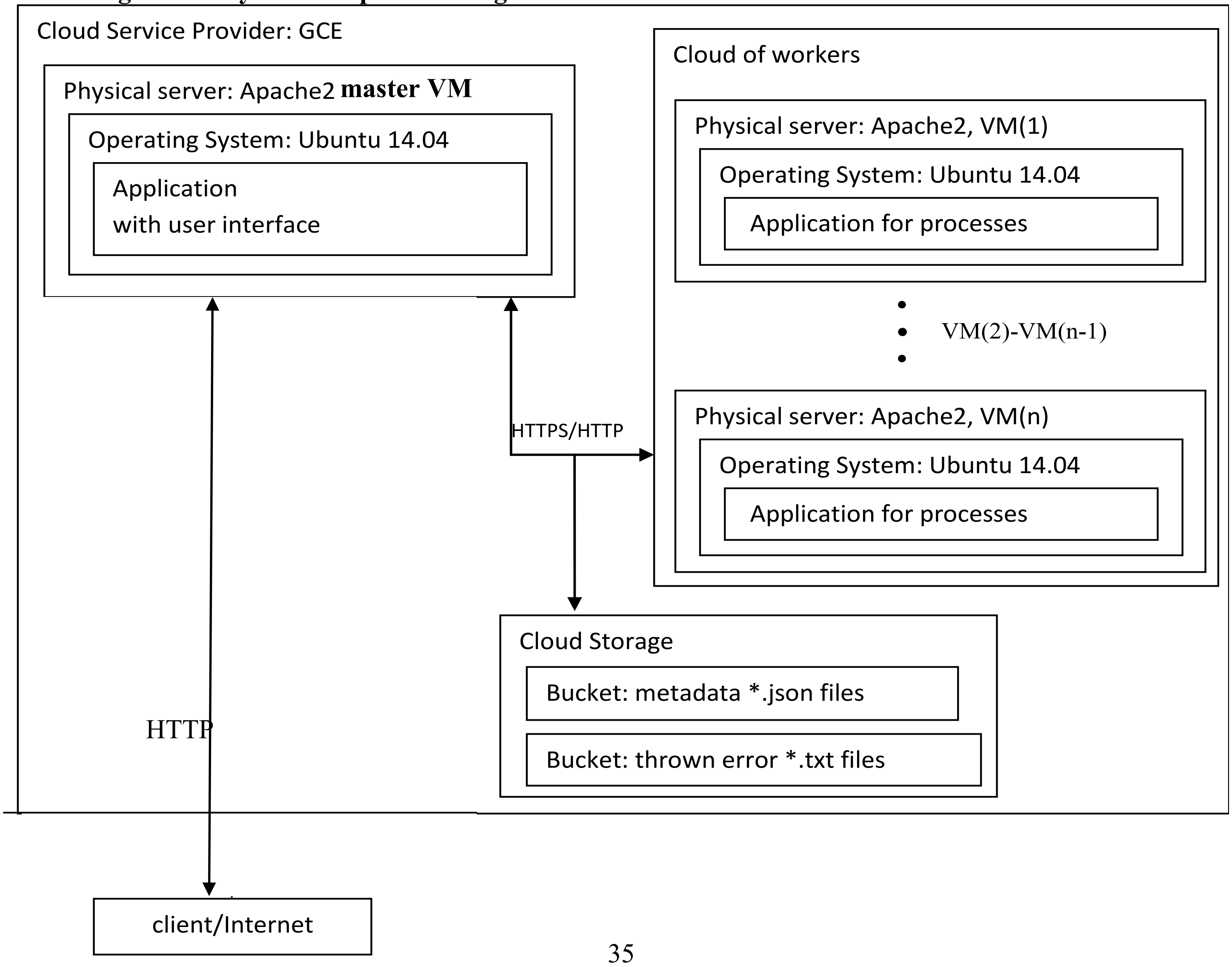
6.3.2Non-functional Requirements

System must finish working during reasonable time elapse (~during 24 hours), regarding log file consists of approximately 3.1 million lines and every line contains an URL to the document.

6.4 Physical View

The components of physical view are featured in Figure 2 and consist of master server, which sends URLs to the cloud of workers (virtual machine instances). Master server gets log file from client and stores it into its own file system. In Figure 2 it is shown that Ubuntu 14 is required, but this is also tested on Ubuntu 15.

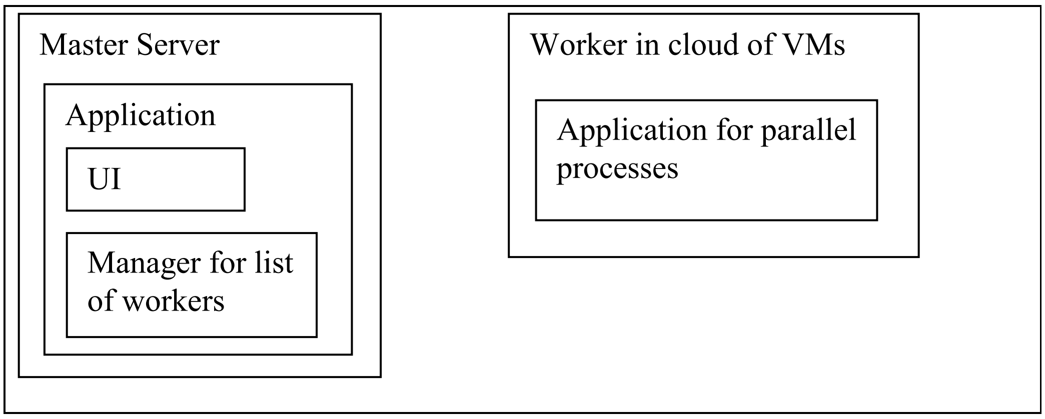
Figure 2: Physical components diagram


Cloud Service Provider: GCE
Physical server: Apache2 master VM
Operating System: Ubuntu 14.04
Application with user interface
Cloud of workers
Physical server: Apache2, VM(1)
Operating System: Ubuntu 14.04
Application for processes
VM(2)-VM(n-1)
Physical server: Apache2, VM(n)
Operating System: Ubuntu 14.04
Application for processes
HTTPS/HTTP
Cloud Storage
Bucket: metadata *.json files
Bucket: thrown error *.txt files
HTTP
client/Internet

6.5 *Development View*

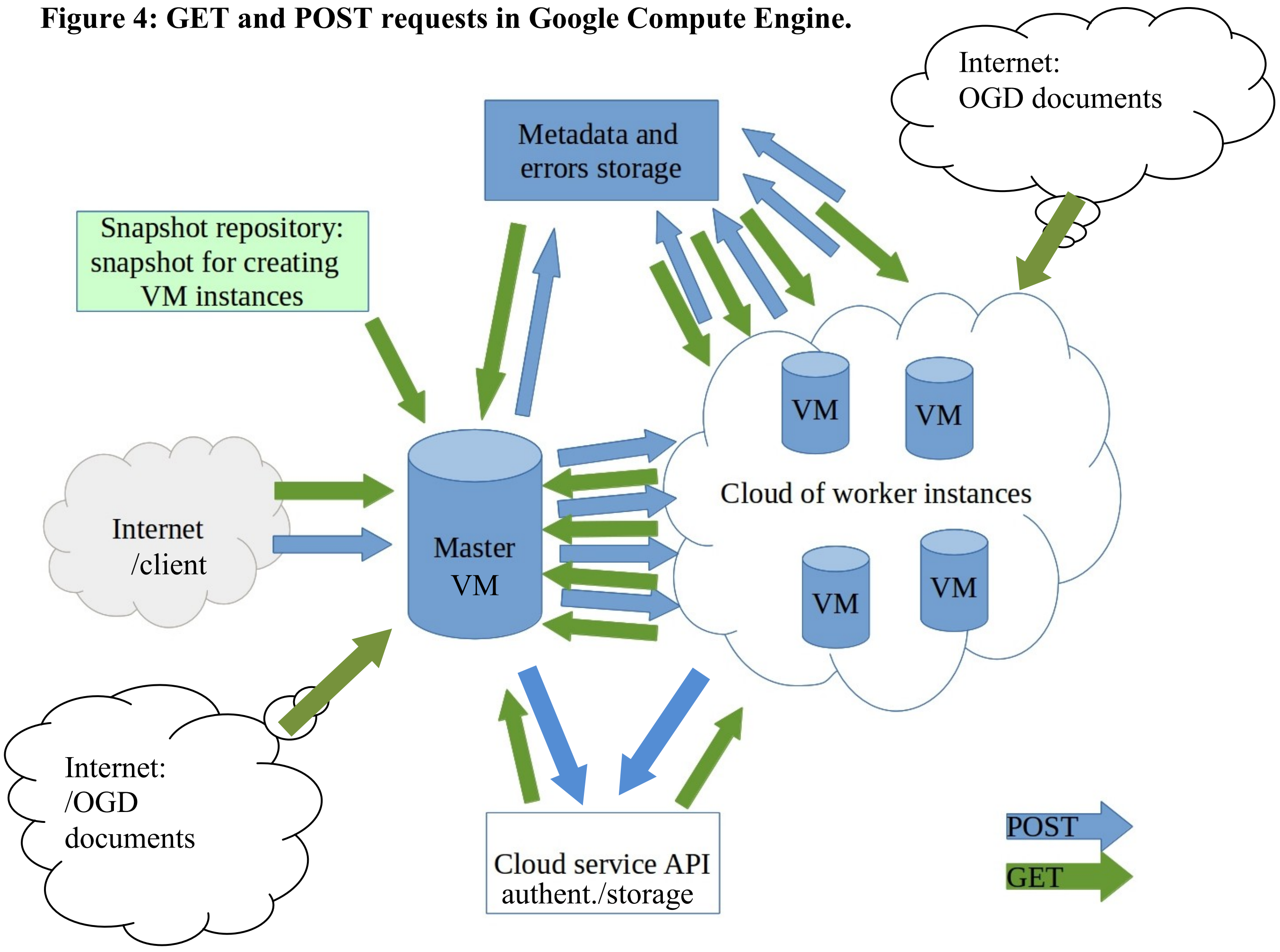
General development view is diagrammed in Figure 3. In Figure 4 a more precise diagram of components of GCE and their interlinks are shown. More close textual explanation for master and worker then follows.

Figure 3: development components diagram.



Master Server
Application
UI
Manager for list of workers
Worker in cloud of VMs
Application for parallel processes

Figure 4: GET and POST requests in Google Compute Engine.



Internet: OGD documents
Metadata and errors storage
Snapshot repository: snapshot for creating VM instances
VM
VM
Cloud of worker instances
VM
VM
Internet /client
Master VM
Internet: /OGD documents
Cloud service API authent./storage
POST
GET

In GCE it is possible to create VMs from snapshot programmatically. It is reasonable to create master and worker virtual machines (VMs) from GCE snapshot, especially when using many VMs. Because of the limited free trial period this is not currently developed for this service, it uses just 4 worker VMs. Author installed software onto 2 GCE disks, separately for master and workers, then made 2 snapshots, separately for master and workers, where all necessary software was present. Then created virtual machines from snapshots manually, 1 for master and from another snapshot 4 ones for workers. Eventually there were 4 worker VMs and one master VM.

In the master VM resides the software for

- communicating with client;
- POSTing tasks to worker VMs;
- GETting RDF graph files from worker VMs;
- communicating with GCE Cloud Storage;
- downloading OGD from Internet (other than Excel and PDF formats).

In worker VMs resides the software for

- reading POSTs from master VM;
- generating RDF graph files;
- communicating with GCE Cloud Storage;
- downloading OGD from Internet (Excel and PDF formats).

The master VM acts like a manager that sends tasks to worker VMs. It sends list of tasks to VMs, every worker VM gets different list, until all lines in log file are read. Worker VMs generate RDF files and download Excel and PDF files from Internet (these can be read only from local file storage). Worker VMs generate errors and metadata objects into GCE Cloud Storage (in Buckets). When worker VMs are finished, master collects generated data (RDF files and downloaded documents in Excel and PDF formats) from VMs. Finally VMs can be terminated either manually or from program in the manager (latter is not implemented in this service). After that the master uses structure of metadata files in json format (in GCE saved into Cloud Storage bucket) as a knowledge base and downloads the new documents' content files into its local file storage. It also downloads these json-formatted metadata files in order to enable smooth browsing of datasets. Errors that are thrown during program work (in worker VMs) are saved also into GCE Cloud Storage.

The cron job[3] task scheduler for monthly updates starts in master server and starts RDFizing process in worker VMs.

There is also a web application for human users, who can observe downloaded datasets and RDF graph files, in master VM.

In the worker VMs, the list of tasks is forked into as many parallel processes as available in this VM, equal to the number of CPUs in the current worker VM. The generation of RDF graph files happens separately in every VM. When tasks are executed and there is no more incoming tasks, the VM finishes its work. The next step is taken by the master server and it collects RDF graphs from all VMs and aggregates/merges them based on the graph type (org., loc. or per) into its local file system into certain folder. . It also downloads Excel and PDF files from worker VMs to another certain folder (`datadownload/downloaded_files/<dd_mm_yyyy>`) of its file system.

[3] Cron job – Linux/Unix command for creating/editing, listing and removing cron jobs is `crontab`.

6.6 *Logical View*

The master server and VMs are created in GCE. The logfile is uploaded into master server file system; the metadata of documents and errors are stored into Google Cloud Storage [53].

Packages' Diagrams

The package diagram is depicted in two figures. There are high level interrelations shown in the Figure 5 (blurred box inside dashed line means that it is another option for large log files storage) and master-worker interrelation in the Figures 6-7. There are listed Python modules, PHP files and directories which reside in each package. Dashed line denotes a term “uses”, where at the beginning of a line is a user and at the end of an arrow is module(s) or directory/file, that the user uses.

Figure 5: : Packages at high level and their interrelations.

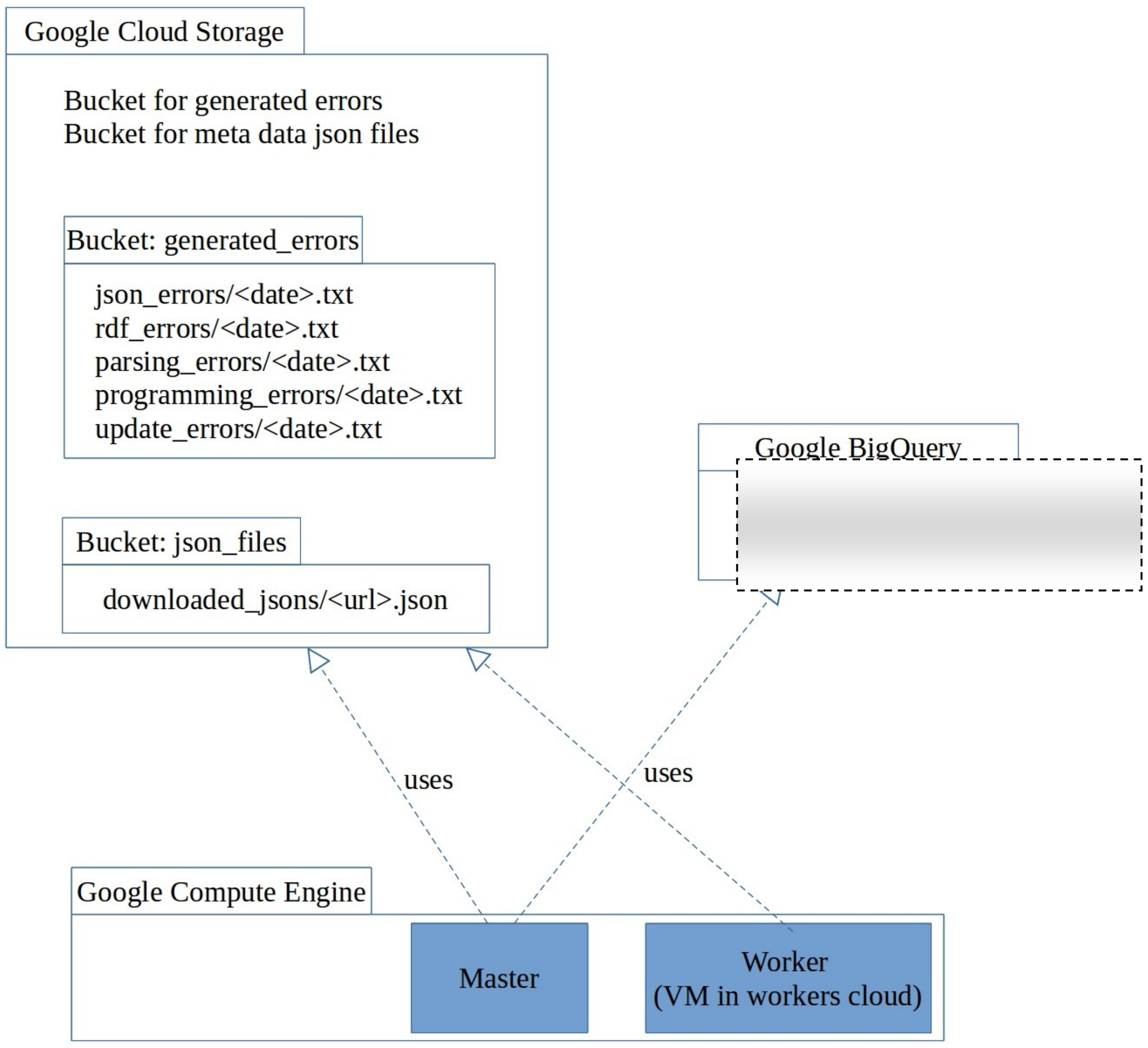


Figure 6: Packages diagram for master-worker level (master VM).

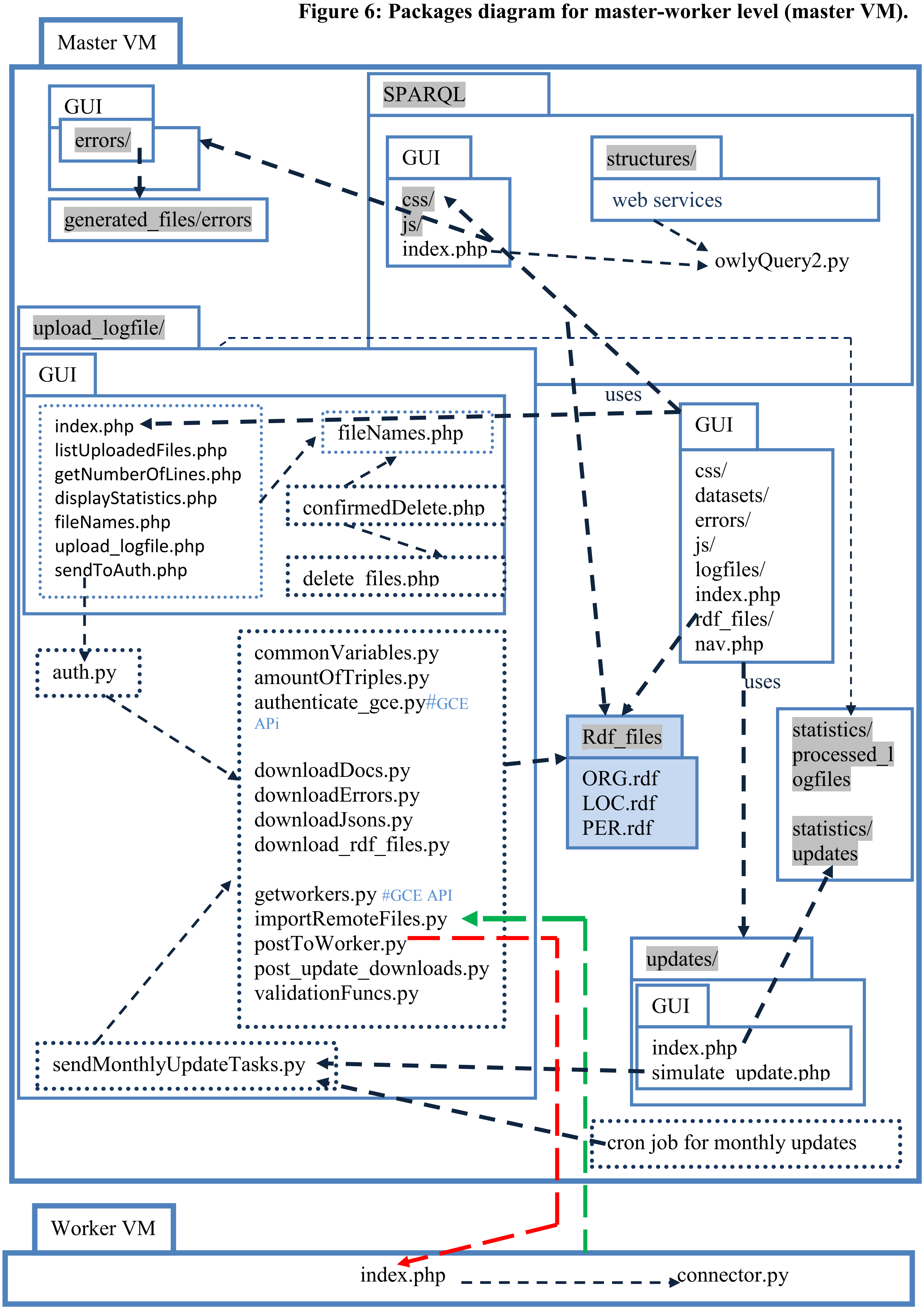
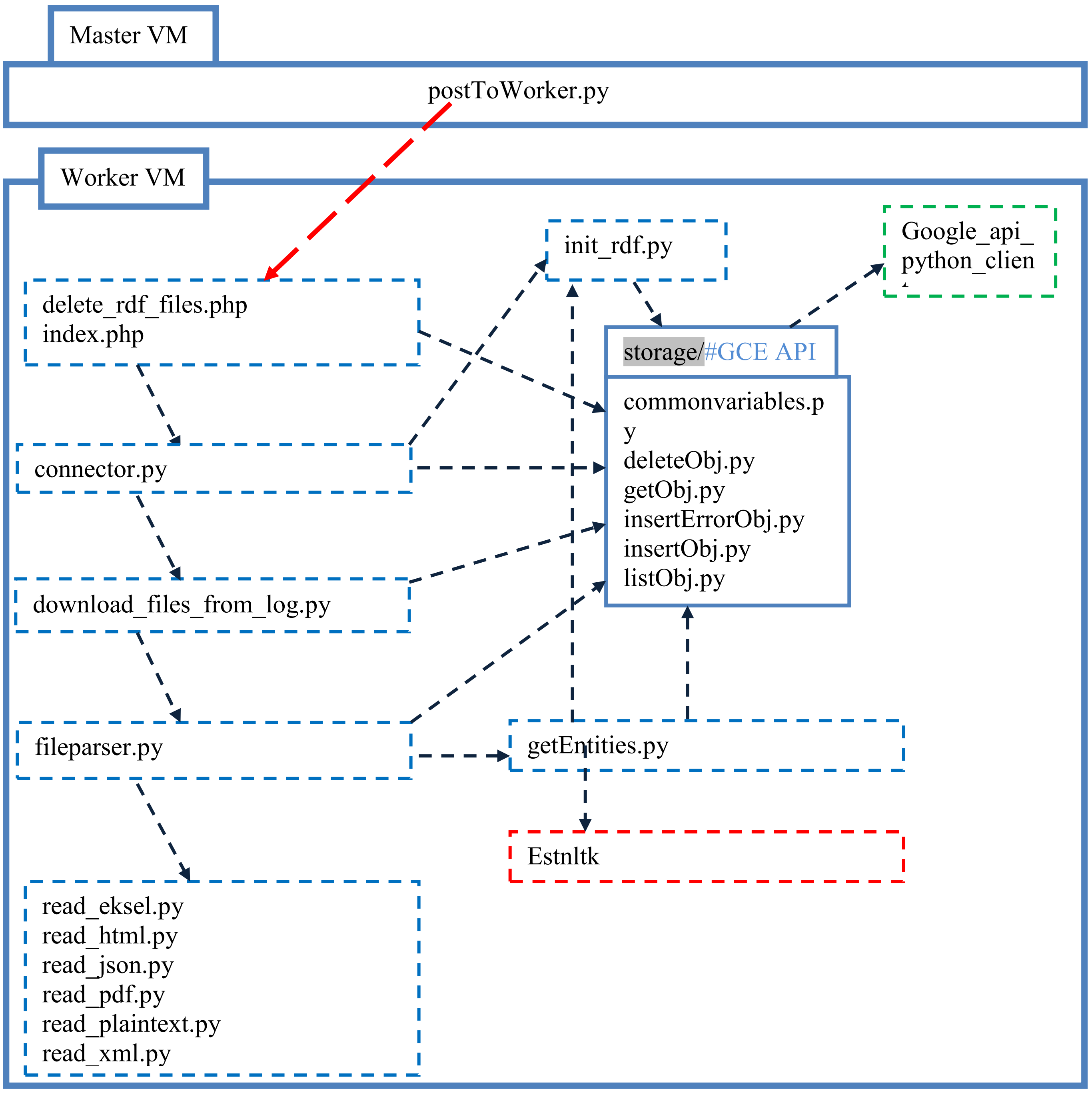


Figure 7: Packages diagram for master-worker level. (worker VM).



6.7 Process View

In the process view in Figures 8, 9, 11a and 11b the most crucial decision points of the process flow are shown. In Figures 8 and 9 the process in the master server and in Figures 11a-11b the process of a single worker VM in the workers' VMs cloud are shown.

6.7.1Master Server Process View

The process starts, when human user pushes button in web app, or when "cron job" (task scheduler for making monthly updates; also a functionality is implemented for simulating monthly manual updates (by pushing button)) - the master server sends task for updating RDF files.

6.7.1.1 Reading Log File, Validating and Sending Data

Process in Figure 8

Client in `http://<master-IP>/upload_logfile`

Client is a human user with computer (C). C uses GUI in web app for uploading log file and then C sees it in the table of log files. Next to the log file name there is a button for starting RDFizing this log file (see also Figure 16). C pushes button. Log file's path in the file system is sent to **`auth.py`**.

There is a program in the master server that reads lines of input data (a logfile). It filters out image and video files as these don't contain texts

The program asks (through API [92]) the list of virtual machines including their IP addresses. The program then uses POST method for sending accepted URLs to the cloud of workers. It collects some predefined number of URLs into a list and sends different lists to every virtual machine (worker.)

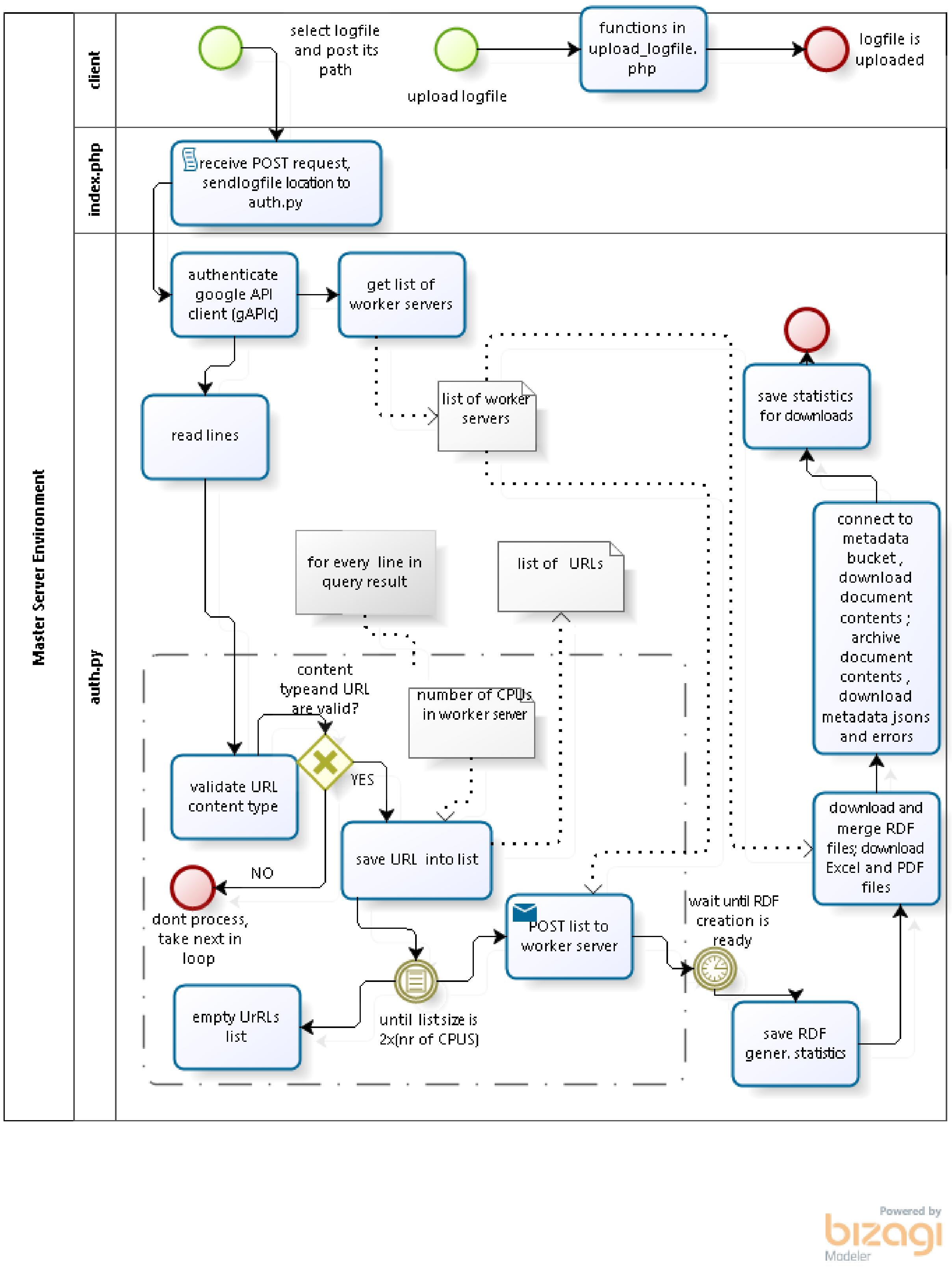
auth.py

auth.py receives log file's path.

Before reading it, it authenticates itself against GCE and then gets list of worker VMs. If workers' list is not empty, it reads log file line by line. It reads the info in a log file row. There are URL to document and host-URL (author elsewhere calls it 'basic URL') and certain characters P and X. It filters out (ignores) the URLs, where certain column contains "X" or "P" (see chapter 7.1, **(4)**). "P" means that URL refers to the file named "http://.../robots.txt". It also decides whether to send hostname-URL (see chapter 7.1 **(5)**) or file-URL (see chapter 7.1 **(3)**) to a worker VM. "X" means aggressive JavaScript of a document. It then checks if document at the URL is textual by content type; if the content type is no-type or image/video/audio/css/js etc. then the URL is ignored. The program asks (through API [92]) the list of virtual machines including their IP addresses and gets to know how many CPU each has. It doubles this number of CPU and fills list in that size with valid URLs for sending this list to the particular worker. It collects this predefined number of URLs into a list and sends different lists to every virtual machine (worker.) The program uses POST method for sending accepted URLs to the cloud of workers.

It waits until all workers have finished its work and downloads generated RDF graph files into master VM's local file system. It then downloads PDF and Excel files from workers. Then it downloads generated json-formatted metadata files and error files from GCE Cloud Storage.

Figure 8: Process in the master VM if it is started by a human user; continues in figure 11a, in worker VM.


Master Server Environment
client
index.php
auth.py
select logfile and post its path
upload logfile
functions in upload_logfile.php
logfile is uploaded
receive POST request, sendlogfile location to auth.py
authenticate google API client (gAPIc)
get list of worker servers
list of worker servers
save statistics for downloads
read lines
for every line in query result
list of URLs
connect to metadata bucket, download document contents; archive document contents, download metadata jsons and errors
content typeand URL are valid?
number of CPUs in worker server
validate URL content type
YES
NO
save URL into list
dont process, take next in loop
POST list to worker server
wait until RDF creation is ready
download and merge RDF files; download Excel and PDF files
empty UrRLs list
until listsize is 2x(nr of CPUS)
save RDF gener. statistics
Powered by bizagi Modeler

6.7.1.2 Evoking Monthly Updates

In master server, there is a cron job (task scheduler), that runs a file that is responsible for checking whether the data have changed and then updates RDF files. The diagram is shown in Figure 9. The same takes place when a human user pushes button for simulating updates (browser view of this is in Figure 19).

Process in Figure 9

sendMonthlyUpdateTasks.py

is the same as auth.py, but it reads URLs from master VM local storage, from the folder of json-formatted metadata files. It additionally does an initial detecting of changes of documents at URLs. For that it opens document at URL and then reads it, then calculates its hash value and compares with the value in metadata file.

6.7.1.3 Downloading Documents

After workers have finished their work, the master downloads (GET request) and aggregates the generated RDF files from each worker into its file system.

Based on the information that is saved into metadata (the structure of it, and how it supports datasets downloading, is explained in chapter 4), the master also downloads the files into its local file storage. In case of HTML pages, it only downloads the textual content, without including media files and link content. It also downloads json-formatted metadata files and error files from GCE Storage.

6.7.1.4 Querying RDF Files (SPARQL Endpoint)

User (person) can use graphical SPARQL endpoint (browser view of it is in Figure 15) to query these RDF files, against web page, location, person names or organization. There are also web services for querying these RDF files (for machines). The web service takes in SPARQL query sentence as POST parameter "sparql" value and sends the response either json format or in RDF/XML format.

6.7.1.5 Getting Datasets

The term “datasets” refers here to the downloaded documents. According to the saved metadata, the downloaded files can be displayed, using GUI, by downloading date, content type and document's URL (browser view of it is in Figure 17).

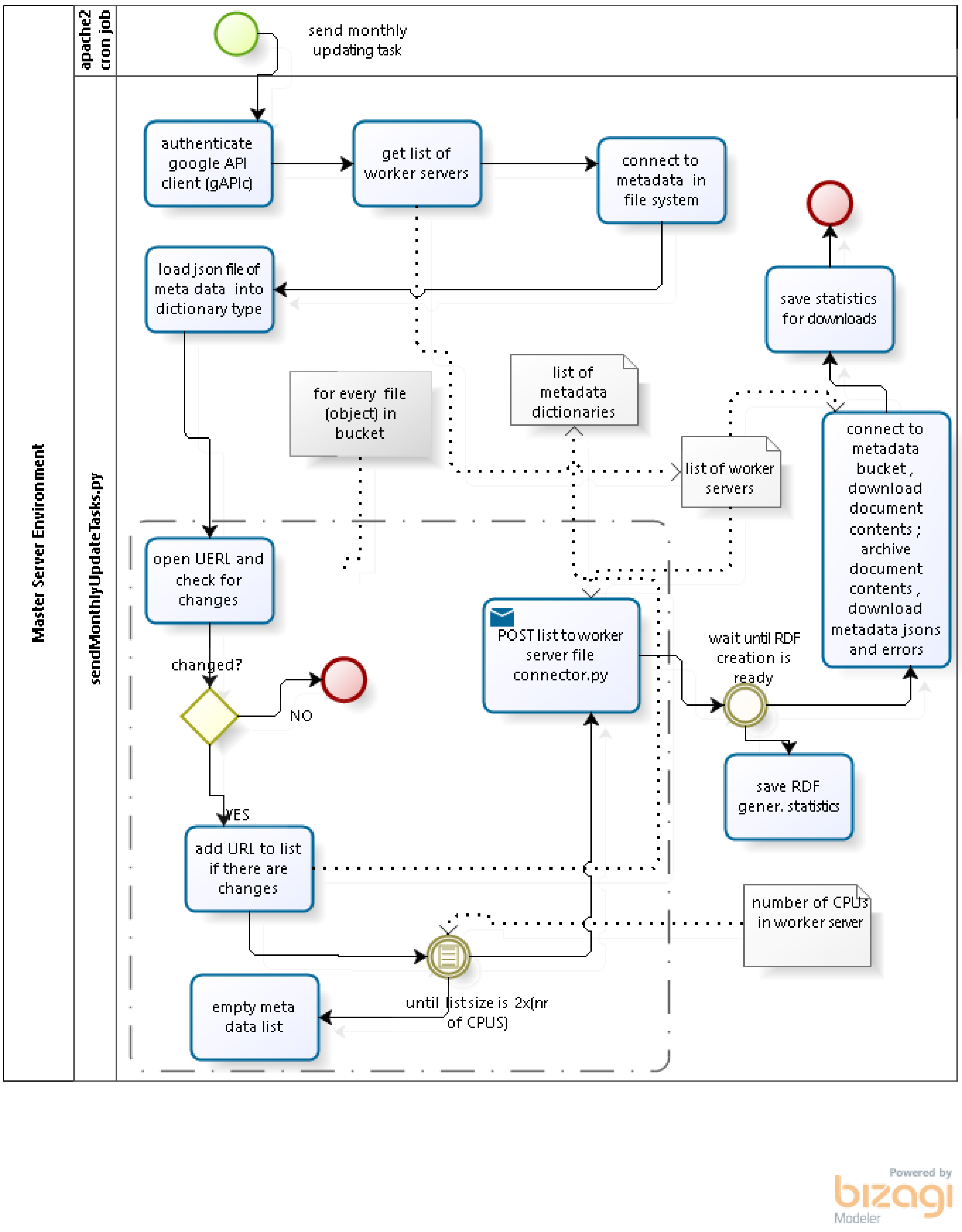
6.7.1.6 Logging of Statistics

The time spent on generating RDF files and on downloading documents is measured and saved into statistics directory in the master server file system (browser view of it is in Figure 16 and 19).

6.7.1.7 Logging of Errors

While reading the log file, metadata, downloading documents, several errors may rise. These errors are continuously saved into errors repository, where a separate error file is created for each step. In each file, at each line the time, line number, error type and the name of the method, where the error happened, is recorded (browser view of it is in Figure 20).

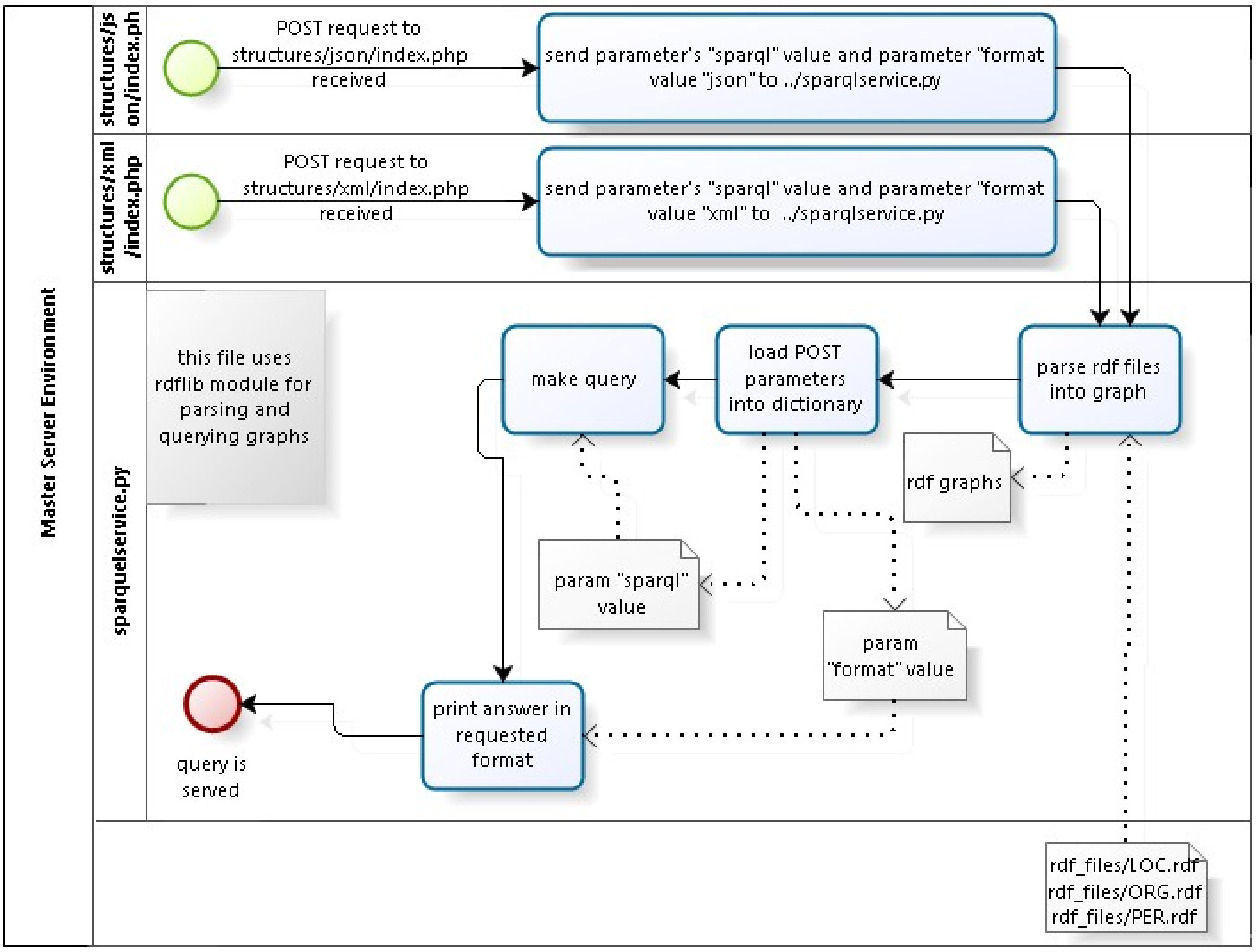
Figure 9 Process in the master server if the cron job started monthly update process


Master Server Environment
apache2 cron job
sendMonthlyUpdateTasks.py
send monthly updating task
authenticate google API client (gAPIc)
get list of worker servers
connect to metadata in file system
load json file of meta data into dictionary type
save statistics for downloads
for every file (object) in bucket
list of metadata dictionaries
list of worker servers
connect to metadata bucket, download document contents; archive document contents, download metadata jsons and errors
open UERL and check for changes
changed?
NO
YES
POST list to worker server file connector.py
wait until RDF creation is ready
save RDF gener. statistics
add URL to list if there are changes
number of CPUs in worker server
empty meta data list
until listsize is 2x(nr of CPUS)
Powered by bizagi Modeler

6.7.1.8 SPARQL endpoint

In Figure 10 the process of SPARQL web services is shown for both, json and RDF/XML formats.

Figure 10: Process for SPARQL services.


Master Server Environment
structures/json/index.php
structures/xml/index.php
sparquelservice.py
POST request to structures/json/index.php received
send parameter's "sparql" value and parameter "format value "json" to ../sparqlservice.py
POST request to structures/xml/index.php received
send parameter's "sparql" value and parameter "format value "xml" to ../sparqlservice.py
this file uses rdflib module for parsing and querying graphs
make query
load POST parameters into dictionary
parse rdf files into graph
rdf graphs
param "sparql" value
param "format" value
print answer in requested format
query is served
rdf_files/LOC.rdf
rdf_files/ORG.rdf
rdf_files/PER.rdf

There is also a Python file **owlyQuery2.py** that receives task and query sentence from file located in the same parent folder **`SPARQLendpoint/index.php`**. **owlyQuery2.py** uses Python package RDFlib for quering RDF graph files.

6.7.2Cloud of Workers

Every virtual machine (VM) in the cloud of workers has exactly the same software. Process in a single VM starts, when the list of URLs with other relevant data is received. The diagram is split into two figures.

Figure 11a: 1st half of a process in worker server in the cloud of VMs (human started)

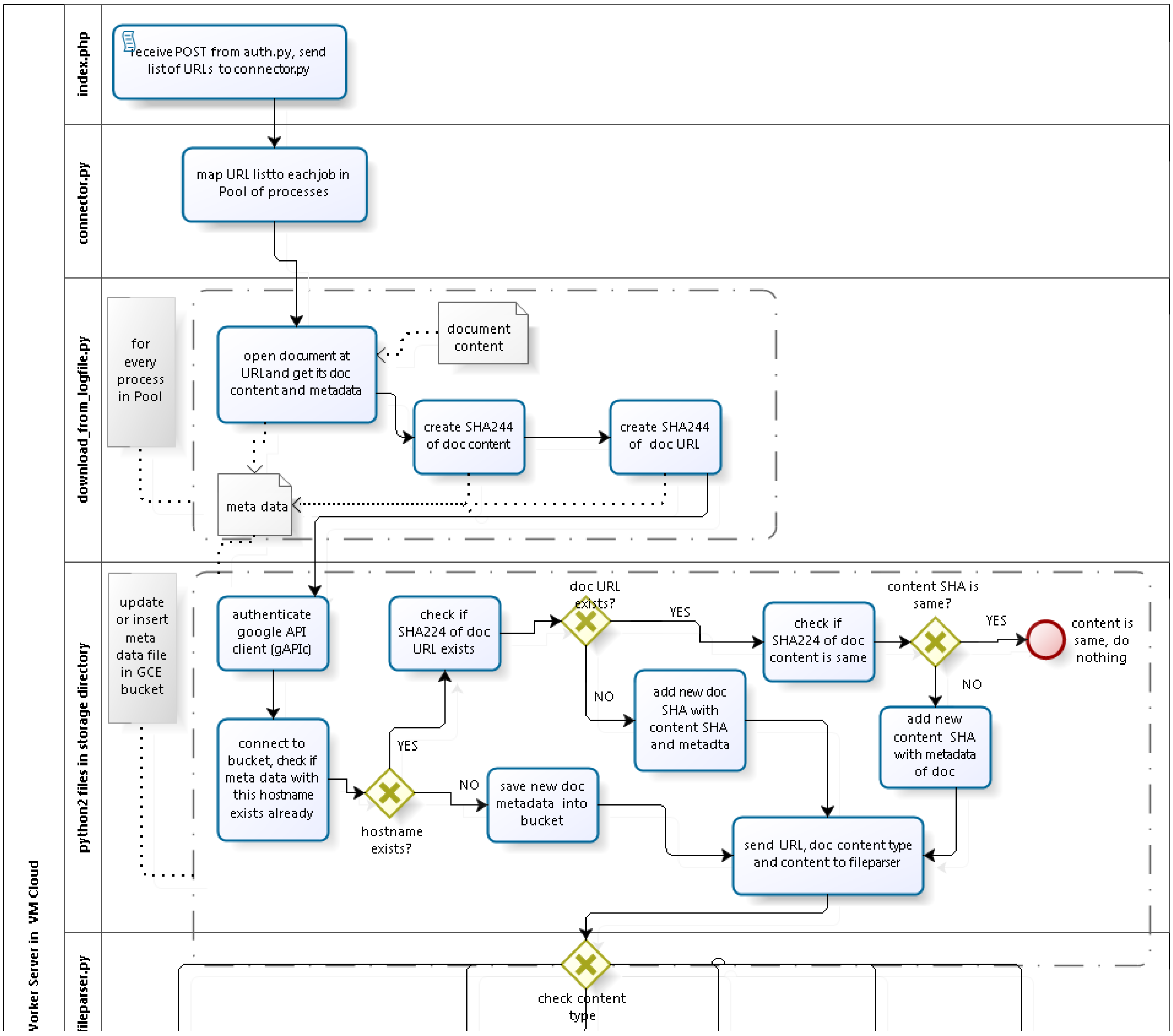
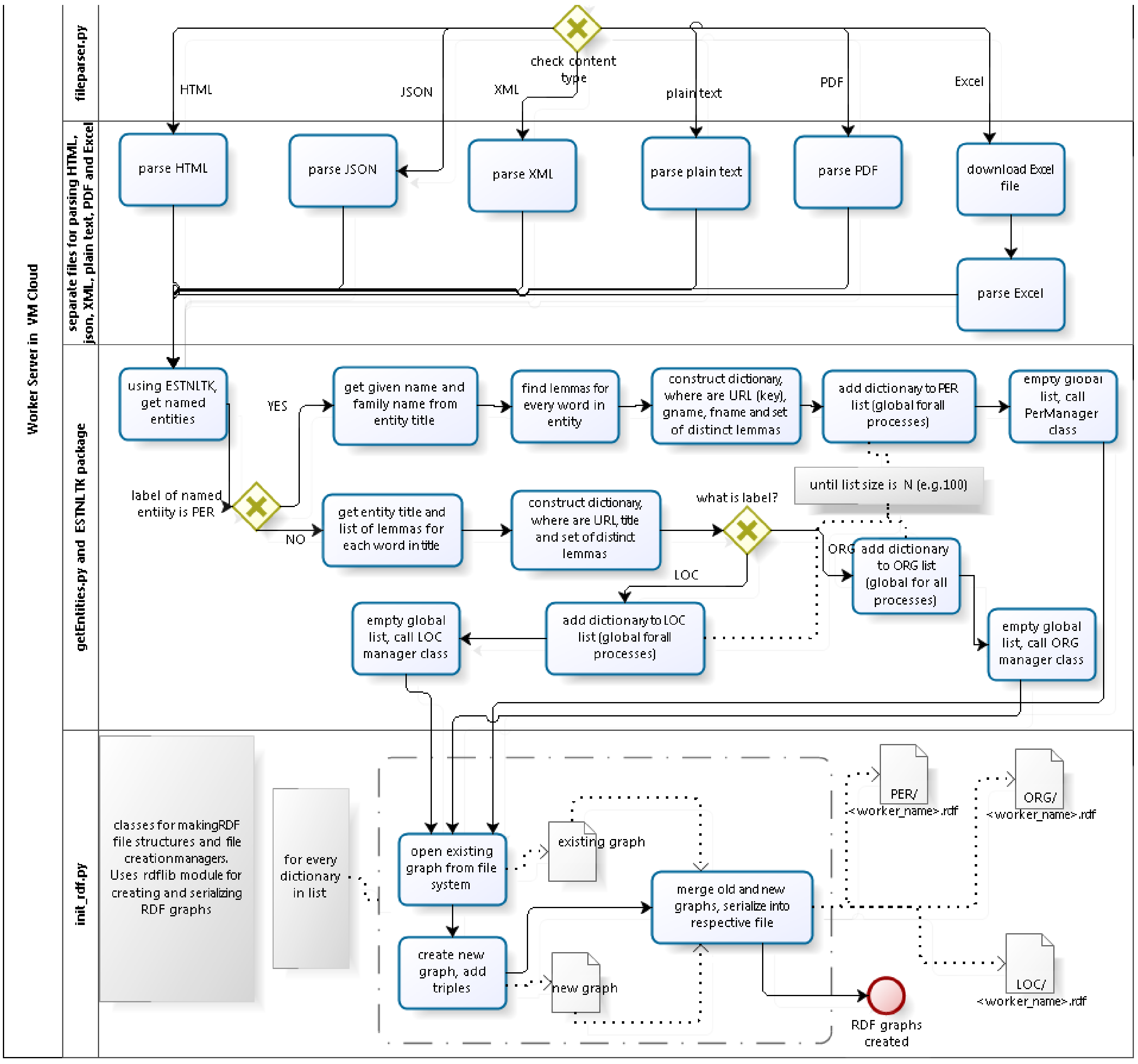

Worker Server in VM Cloud
index.php
receive POST from auth.py, send list of URLs to connector.py
connector.py
map URL list to each job in Pool of processes
download_from_logfile.py
for every process in Pool
open document at URL and get its doc content and metadata
document content
create SHA244 of doc content
create SHA244 of doc URL
meta data
python2 files in storage directory
update or insert meta data file in GCE bucket
authenticate google API client (gAPIc)
connect to bucket, check if meta data with this hostname exists already
hostname exists?
YES
NO
check if SHA224 of doc URL exists
save new doc metadata into bucket
doc URL exists?
YES
NO
add new doc SHA with content SHA and metadta
check if SHA224 of doc content is same
content SHA is same?
YES
NO
content is same, do nothing
add new content SHA with metadata of doc
send URL, doc content type and content to fileparser
fileparser.py
check content type

Figure11b: 2nd half of process in worker server in the cloud of VMs


Worker Server in VM Cloud
fileparser.py
separate files for parsing HTML, json, XML, plain text, PDF and Excel
getEntities.py and ESTNLTK package
init_rdf.py
check content type
HTML
JSON
XML
plain text
PDF
Excel
parse HTML
parse JSON
parse XML
parse plain text
parse PDF
download Excel file
parse Excel
using ESTNLTK, get named entities
label of named entiity is PER
YES
NO
get given name and family name from entity title
find lemmas for every word in entity
construct dictionary, where are URL (key), gname, fname and set of distinct lemmas
add dictionary to PER list (global for all processes)
empty global list, call PerManager class
until list size is N (e.g.100)
get entity title and list of lemmas for each word in title
construct dictionary, where are URL, title and set of distinct lemmas
what is label?
ORG
LOC
add dictionary to ORG list (global for all processes)
empty global list, call ORG manager class
add dictionary to LOC list (global for all processes)
empty global list, call LOC manager class
classes for makingRDF file structures and file creationmanagers. Uses rdflib module for creating and serializing RDF graphs
for every dictionary in list
open existing graph from file system
existing graph
create new graph, add triples
new graph
merge old and new graphs, serialize into respective file
PER/ <worker_name>.rdf
ORG/ <worker_name>.rdf
LOC/ <worker_name>.rdf
RDF graphs created

In Figure 11a, the main tasks are validating incoming data, opening document at certain URL, getting its metadata (see chapter 4, “MetaData Model”), validating metadata (see “MetaData Model”), storing metadata (see “MetaData Model”) into GCE Cloud Storage, and sending to the fileparser (figure 11b), where document contents are distributed by their content type into different parsers. Parser ignores texts that include XML-invalid and RDFlib URI-invalid characters (see ch. 6.7.2.3 “Parsing Documents”). Then process continues with using Estnltk packages and extracting named entities. Then the entities are inserted into RDF graph.

The process view of a “cron job”-started process is the same in the worker server.

6.7.2.1 Validating the Incoming Data

Every VM gets different input data, then makes a choice whether to use initial URL or redirected URL. It takes the last redirected URL if redirection took place. Then the program validates content type. The document at given URL is accepted, if its content type refers to textual content. Images, videos, font-files, script files like css and js etc. are ignored. Welcomed file types are

- Excel;
- json;
- plain text;
- HTML;
- PDF;
- XML.

There is a pool of processes (which size equals to the size of current VM instance's number of CPUs) in every VM, and accepted URLs are spread into these processes, different URL into each process. So the processes work separately and in parallel.

6.7.2.2 Saving Metadata

Next step is checking whether this document is already processed, based on the document's metadata. Saving and using metadata is exactly described in chapter 4, "Metadata Model". If there is no metadata for this document or the content has changed, new metadata is saved or is updated into json-formatted structure and posted to the repository. After that, the process of parsing the document's text starts; after parsing, the extracting named entities for locations, persons and organizations starts.

6.7.2.3 Parsing Documents

There is a difference in parsing Excel, PDF content and other content types. Using Python language, Excel and PDF documents should be downloaded before program can start parsing it. This is why the master server, after VMs have finished, downloads Excel and PDF documents from VMs, in addition to downloading RDF files. During parsing, the strange non-ASCII letters are filtered out. Also texts that include XML-invalid characters [93, chapter 2.2] and the control characters invalid for URI [94] are ignored. It is done by comparing the list of these invalid character and all characters inside the text. This is mainly because RDF graphs are saved into format RDF/XML and result in invalid RDF graph files. Unfortunately, it slows down the performance. Although the treatment of poorly defined document encoding remains beyond of this system's scope, this service tries to convert some characters (the converting happens before ignoring texts with invalid characters for XML and URI), following characters are replaced:

"Ãµ" replaced with "õ"
"Ã•" replaced with "Õ"
"Ã^Õ" replaced with "Õ"
"Ã¼" replaced with "ü"
"Ã^Ü" replaced with "Ü"
"Ãœ" replaced with "Ü"
"Ã¤" replaced with "ä"
"Ã„" replaced with "Ä"
etc.

A character that cannot belong to a name, is replaced by a space, e.g. "`<numeric value>`" and characters

„¦“, „;“, „:“, „(“, „)“, „?“, „!“, „,“, „|“, „&“, „@“, „˙“, „°“, „ˇ“, „^“, „ˆ“, „/“, „\\“, „{“, „}“, „[“, „]“, „¬“, „_“, „~“, „#“, „%“, „<“, „>“, „=“, „+“, „*“, „˝“, „"“, „¨“, „»“, „`“.

Also, in case of HTMLand XML documents, the text between markers “<” and “>” is skipped. These are HTML/XML tags and may be JavaScript programs.

Unfortunately, these replacement operations also slow down the performance.

6.7.2.4 Extracting Named Entities, Serializing RDF Graphs

For extracting named entities for people’s names, organizations and locations, Estnltk is used. During development of this system it appeared, that it does not always recognize names of organizations, with included strings 'kogu', 'selts', 'ansambel', 'keskus', 'ühendus', 'ühing', 'mtü', 'oü', 'as', 'klubi', 'asutus', 'keskus', 'fond' in the entity’s text. Then this program’s part itself changes entity’s label to „ORG“. The same goes for a string that reference to locations: 'vabarii', 'maakond'. After entities are extracted, there are 3 global lists (accessible for all processes in current VM) for every entity type (locations, organizations and people). There is also a global predefined numeric amount, how large the list can be. This is for decreasing the number of times, when RDF graph files are opened, updated and closed. If the list length exceeds this amount, the content in this list is sent to the next task for merging entities to the respective graph file, either LOC.rdf, PER.rdf or ORG.rdf.

6.7.2.5 Generating RDF Files

After that all entities are added to the respective graph: locations' names to the locations' graph, person's names into peoples' graph and organizations' names into organizations' graph. These graphs are then serialized into respective ”`<workerVMname>.rdf`“ files, which are saved into three directories `LOC`, `ORG` and `PER`. Every worker VM saves this RDF file into its own file system. The master knows, when the jobs in VM are ready (all URLs have been sent), and downloads all RDF files into its (Master's) local file system and aggregates them into the compound of three files ”`LOC.rdf`“, ”`ORG.rdf`“ and ”`PER.rdf`“.

6.7.2.6 Logging of Errors

At every program step several errors may rise while reading documents, parsing, extracting entities, serializing graphs etc. These errors are continuously saved into errors repository-GCE Cloud Storage-, where a separate error file is created for each step. In each file, at each line the time, line number, error type and method name, where the error happened, is recorded.

6.8 Use Cases

Resulting RDF/XML files are LOC.rdf for locations, ORG.rdf for organizations, PER.rdf for persons' names, recognized in Estonian Open Data. Every recognized entity includes predicate „mentionedAt" that links to the open document.

6.8.1General Using Scenarios

One example how the data can be useful, is to consume these files for search engine: one can search for people, organizations and locations and will be provided with links to the documents of those mentioned at. Also, vice versa is possible: by inserting web address of document - the recognized organizations, locations and people at this address will be delivered. This approach is currently implemented in this project's SPARQL endpoint.

These RDF/XML files can also be integrated with other linked data. For example resolving recognized entities with actually existing entities in DBpedia. Although DBpedia's link in English http://dbpedia.org/resource/Estonia is semantically many times richer than http://dbpedia.org/resource /Eesti. One solution could be first extracting Estonian Wikipedia *Vikipeedia* into RDF and then resolve recognized entities.

These RDF/XML may also be integrated into applications: wherever an entity in RDF file is encountered in an application, the web addresses that include this entity may be listed. This way the user of the application is provided with more rich information on this entity: the user can obtain more information following the provided links. This system checks the processed data sets every month: when something has changed, then it stores the new version of the document to the repository. User can then read these stored documents: it is especially helpful when the user cannot find certain entity from the document under its provided link.

6.8.2Use Case Diagrams

There are several use cases (Figures 12-13) that can start the process of RDF generation: firstly when a human uses web application in intention to generate RDF files for extracted named entities; secondly when a "cron job" (Figure 13) in Apache2 server (in master VM) starts process at certain moment monthly. Another type of use case is for human user, who can also select the log file for processing and use web application for querying RDF graph. There is also a web service use case, for machines, for querying RDF graphs (Figure 14).

Figure 12: Use cases for human user.

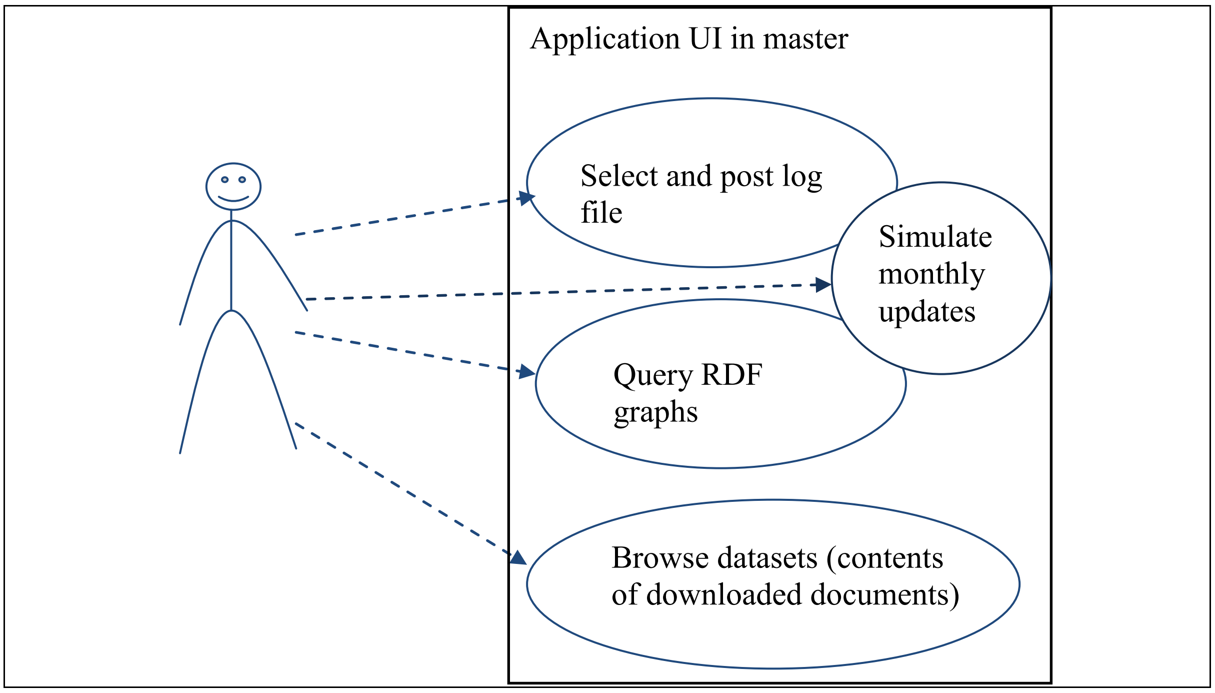


Figure 13: Use case for „cron job“.

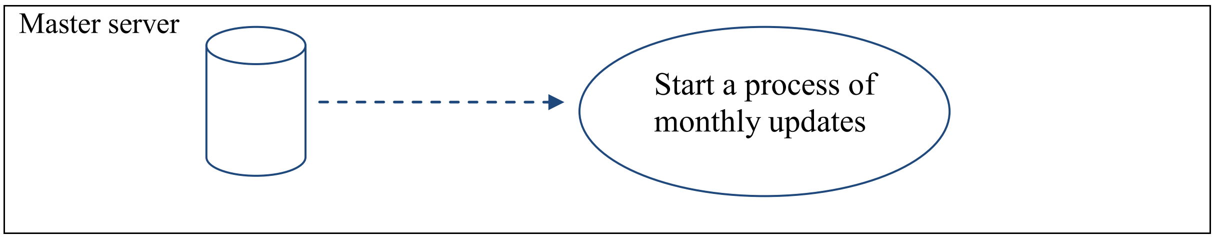
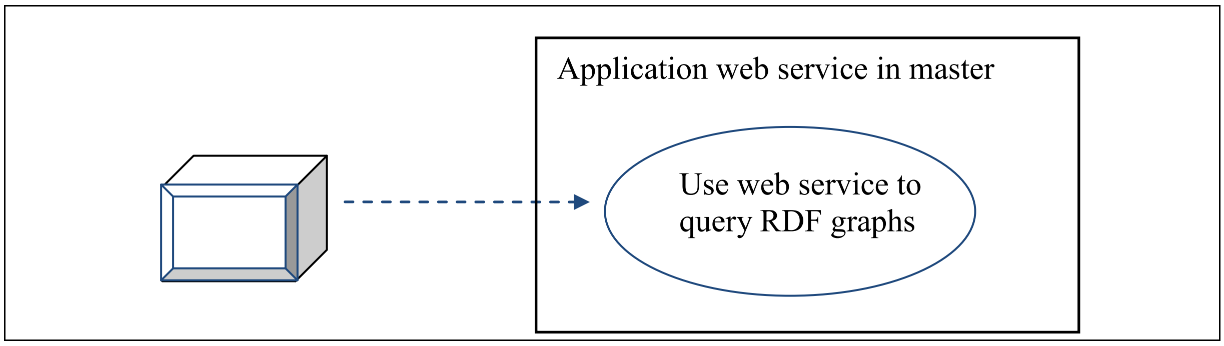


Figure 14: Use case for machines.


Application web service in master
Use web service to query RDF graphs

6.8.3Using SPARQL endpoint

In Figure 15 the SPARQL endpoint is shown together with the constructed query help boxes developed particularly for this system. This figure shows query result for organization “Bmw”.

There are 3 columns for human user case:

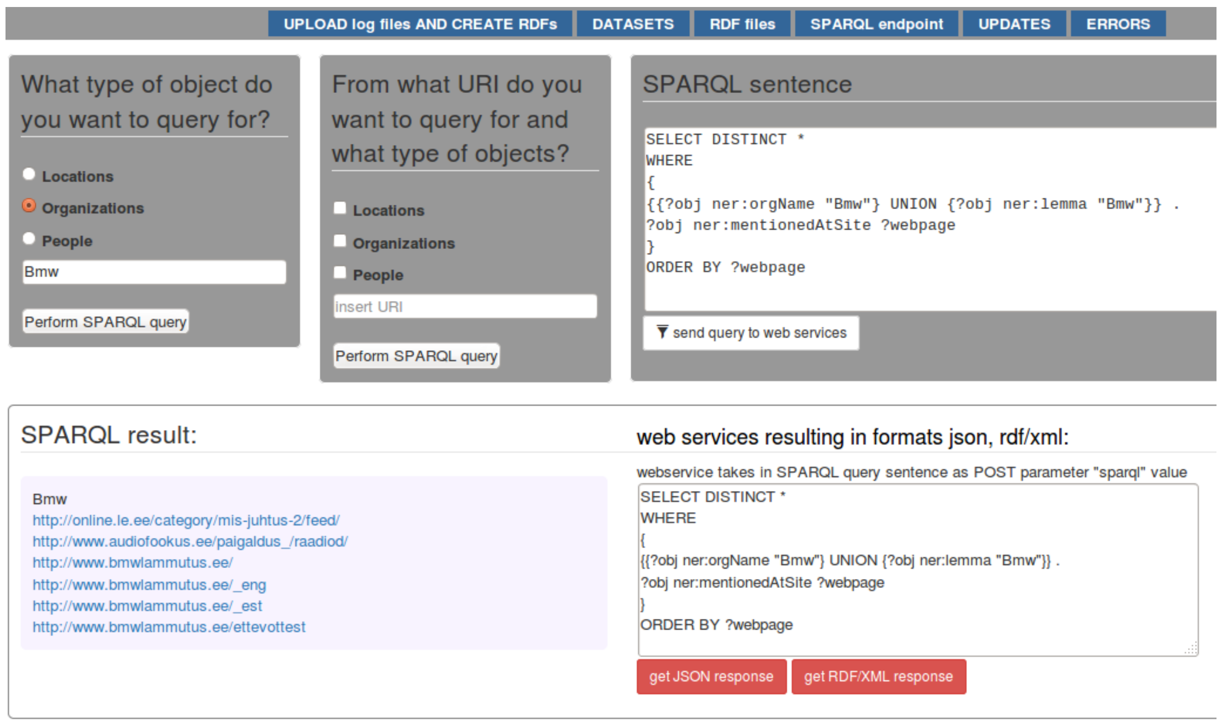
In the left the user can select whether he/she wants to query people names, locations or organizations at a certain web page. In the middle column the user can select all three entity types or one of them and the compulsory field is the web page. In the rightmost column there is a SPARQL query sentence displayed.

Under the third column there are two red buttons representing web service for machines, for sending post request for getting response either in json or RDF/XML format. The json webservice is at the address http://<master’s-ip-address>/SPARQLendpoint/structures/json and the for RDF/XML webservice http:// <master’s-ip-address>/SPARQLendpoint/structures/xml.

The `POST` parameter `'sparql'` should contain SPARQL query sentence `“SELECT … WHERE …”.` In case organization “Bmw” was queried, it is

```
SELECT DISTINCT *
WHERE
{
{{?obj ner:orgName "Bmw"} UNION {?obj ner:lemma "Bmw"}} .
?obj ner:mentionedAtSite ?webpage
}
ORDER BY ?webpage
```
`This searches over organization names and organization “lemmas”(explained in chapter 4.2.2).`

Figure 15 SPARQL endpoint GUI



What type of object do you want to query for?

- Locations
- [✓ Organizations]
- People

Bmw

From what URI do you want to query for and what type of objects?

- Locations
- Organizations
- People

insert URI

SPARQL sentence

```
SELECT DISTINCT *
WHERE
{
{{?obj ner:orgName "Bmw"} UNION {?obj ner:lemma "Bmw"}} .
?obj ner:mentionedAtSite ?webpage
}
ORDER BY ?webpage
```

SPARQL result:

Bmw
http://online.le.ee/category/mis-juhtus-2/feed/
http://www.audiofookus.ee/paigaldus_/raadiod/
http://www.bmwlammutus.ee/
http://www.bmwlammutus.ee/_eng
http://www.bmwlammutus.ee/_est
http://www.bmwlammutus.ee/ettevottest

web services resulting in formats json, rdf/xml:

webservice takes in SPARQL query sentence as POST parameter "sparql" value

```
SELECT DISTINCT *
WHERE
{
{{?obj ner:orgName "Bmw"} UNION {?obj ner:lemma "Bmw"}} .
?obj ner:mentionedAtSite ?webpage
}
ORDER BY ?webpage
```

In Figure 15, under the two gray boxes in the left, the **HTML-result** is shown: the list of links, where “Bmw” was found:

http://online.le.ee/category/mis-juhtus-2/feed/
http://www.audiofookus.ee/paigaldus_/raadiod/
http://www.bmwlammutus.ee/
http://www.bmwlammutus.ee/_eng
http://www.bmwlammutus.ee/_est
http://www.bmwlammutus.ee/ettevottest

Here is the webservice's response when making POST request to **json format**:

```
  {"head": {"vars": ["webpage", "obj"]}, "results": {"bindings":
[{"webpage": {"type": "uri", "value":
"http://online.le.ee/category/mis-juhtus-2/feed/"}, "obj":
{"type": "uri", "value":
"http://www.estnernet.ee/nerdatanet/organizations#bmw"}},
{"webpage": {"type": "uri", "value":
"http://www.audiofookus.ee/paigaldus_/raadiod/"}, "obj": {"type":
"uri", "value":
"http://www.estnernet.ee/nerdatanet/organizations#bmw"}},
{"webpage": {"type": "uri", "value":
"http://www.bmwlammutus.ee/"}, "obj": {"type": "uri", "value":
"http://www.estnernet.ee/nerdatanet/organizations#bmw"}},
{"webpage": {"type": "uri", "value":
"http://www.bmwlammutus.ee/_eng"}, "obj": {"type": "uri",
"value":
"http://www.estnernet.ee/nerdatanet/organizations#bmw"}},
{"webpage": {"type": "uri", "value":
"http://www.bmwlammutus.ee/_est"}, "obj": {"type": "uri",
"value":
"http://www.estnernet.ee/nerdatanet/organizations#bmw"}},
{"webpage": {"type": "uri", "value":
"http://www.bmwlammutus.ee/ettevottest"}, "obj": {"type": "uri",
"value":
"http://www.estnernet.ee/nerdatanet/organizations#bmw"}}]}}
```

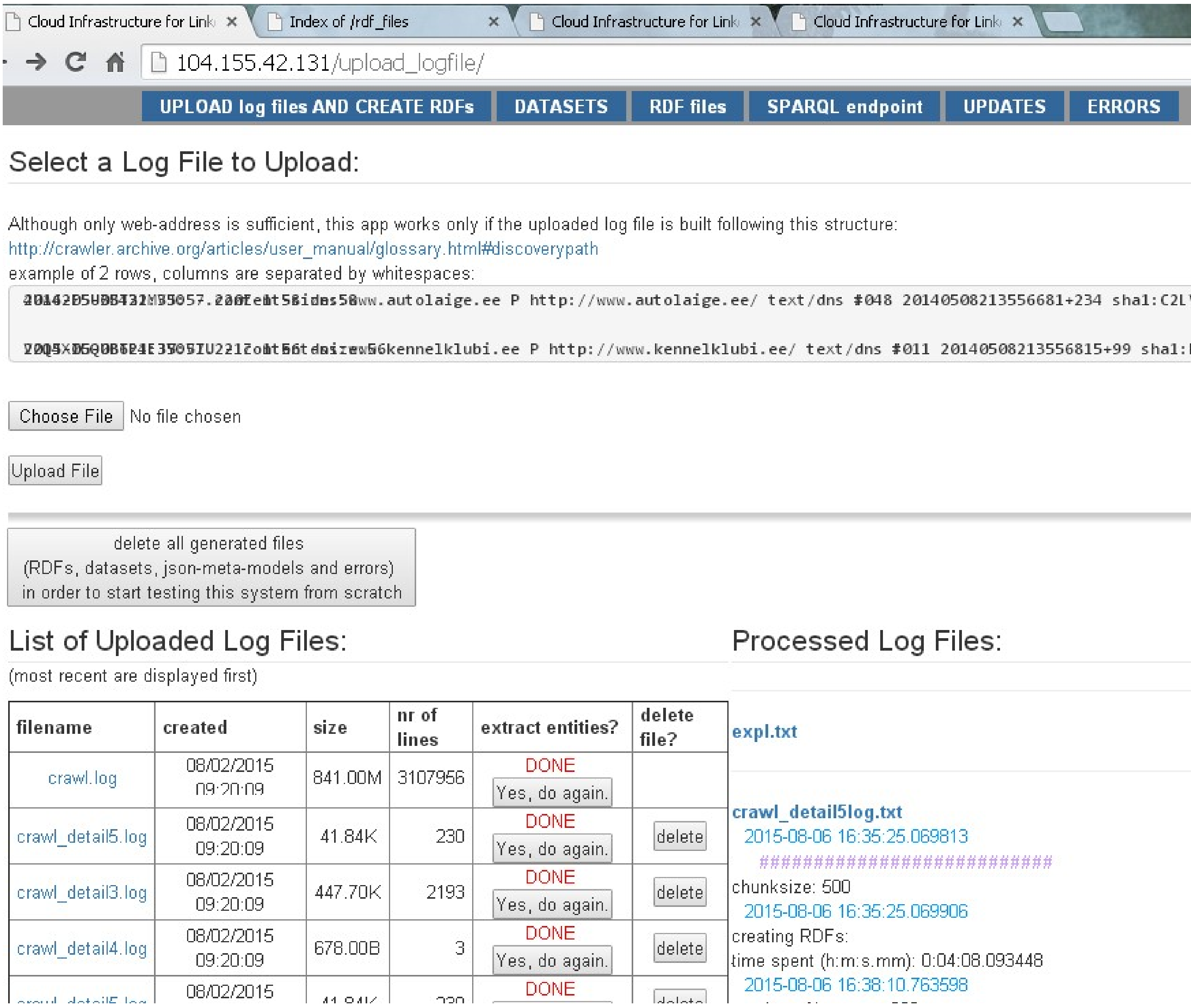
And here is the webservice’s response when making POST request to **RDF/XML format**:

```
  <sparql:sparql>
     <sparql:head>
          <sparql:variable name="webpage"/>
          <sparql:variable name="obj"/>
     </sparql:head>
     <sparql:results>
          <sparql:result>
               <sparql:binding name="webpage">
                    <sparql:uri>
                    http://online.le.ee/category/mis-juhtus-
2/feed/
                    </sparql:uri>
               </sparql:binding>
               <sparql:binding name="obj">
                    <sparql:uri>

     http://www.estnernet.ee/nerdatanet/organizations#bmw
                       </sparql:uri>
               </sparql:binding>
          </sparql:result>
          <sparql:result>
               <sparql:binding name="webpage">
                    <sparql:uri>

     http://www.audiofookus.ee/paigaldus_/raadiod/
                    </sparql:uri>
               </sparql:binding>
               <sparql:binding name="obj">
                    <sparql:uri>

     http://www.estnernet.ee/nerdatanet/organizations#bmw
                    </sparql:uri>
               </sparql:binding>
          </sparql:result>
          ...
     <sparql:results>
  <sparql:sparql>
```

6.8.4Uploading Log File and Starting RDFizing Process

In Figure 16 is shown the browser of view of use cases where human user can upload logfile, start the RDFizing process, delete all generated files (this functionality is useful during development phase) and see statistics of processed log files.

Figure 16: Upload log file and start RDFizing process.



Cloud Infrastructure for Link
Index of /rdf_files
Cloud Infrastructure for Link
Cloud Infrastructure for Link
104.155.42.131/upload_logfile/
UPLOAD log files AND CREATE RDFs
DATASETS
RDF files
SPARQL endpoint
UPDATES
ERRORS

Select a Log File to Upload:

Although only web-address is sufficient, this app works only if the uploaded log file is built following this structure:
http://crawler.archive.org/articles/user_manual/glossary.html#discoverypath
example of 2 rows, columns are separated by whitespaces:

```
[illegible]ww.autolaige.ee P http://www.autolaige.ee/ text/dns #048 20140508213556681+234 sha1:C2L

[illegible]kennelklubi.ee P http://www.kennelklubi.ee/ text/dns #011 20140508213556815+99 sha1:
```

Choose File No file chosen

Upload File

delete all generated files
(RDFs, datasets, json-meta-models and errors)
in order to start testing this system from scratch

List of Uploaded Log Files:

(most recent are displayed first)

filename	created	size	nr of lines	extract entities?	delete file?
crawl.log	08/02/2015 09:20:09	841.00M	3107956	DONE Yes, do again.	
crawl_detail5.log	08/02/2015 09:20:09	41.84K	230	DONE Yes, do again.	delete
crawl_detail3.log	08/02/2015 09:20:09	447.70K	2193	DONE Yes, do again.	delete
crawl_detail4.log	08/02/2015 09:20:09	678.00B	3	DONE Yes, do again.	delete
crawl_detail5.log	08/02/2015	41.84K	230	DONE	delete

Processed Log Files:

expl.txt

crawl_detail5log.txt
2015-08-06 16:35:25.069813
############################
chunksize: 500
2015-08-06 16:35:25.069906
creating RDFs:
time spent (h:m:s.mm): 0:04:08.093448
2015-08-06 16:38:10.763598

6.8.5 Browsing Datasets

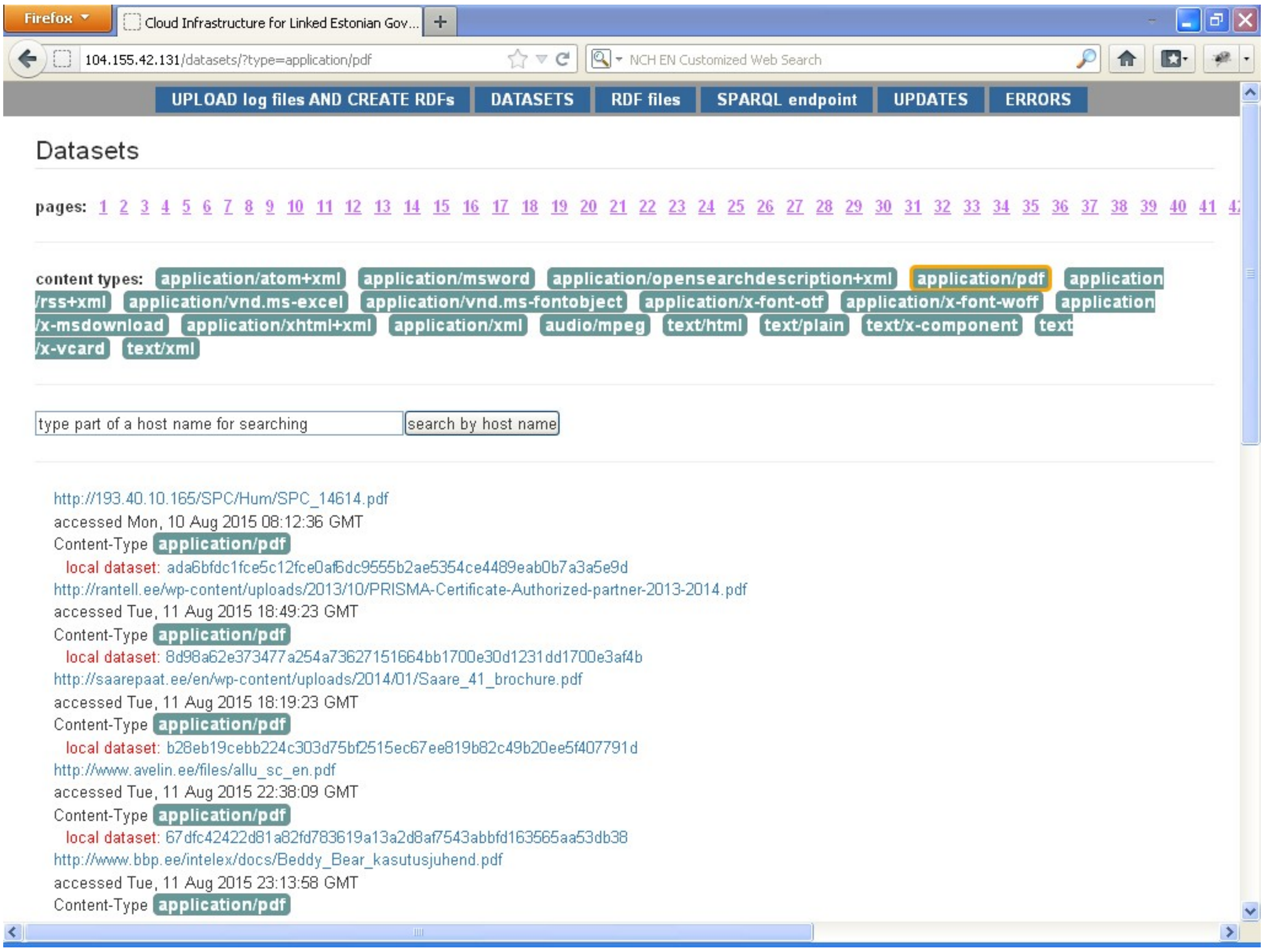
Metadata model is used for faceted search/filtering for enabling more intelligent browsing of archived datasets. At current moment, searching datsets by

- the sequence of characters in a host name;
- content type;
- alphabetically paging

are implemented.

In Figure 17 is shown the moment after user have pushed the content-type button **application/pdf**.

Figure 17: Browsing of datasets: content type „application/pdf“ is searched.



Firefox | Cloud Infrastructure for Linked Estonian Gov...

104.155.42.131/datasets/?type=application/pdf

NCH EN Customized Web Search

UPLOAD log files AND CREATE RDFs | DATASETS | RDF files | SPARQL endpoint | UPDATES | ERRORS

Datasets

pages: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42

content types: application/atom+xml application/msword application/opensearchdescription+xml application/pdf application/rss+xml application/vnd.ms-excel application/vnd.ms-fontobject application/x-font-otf application/x-font-woff application/x-msdownload application/xhtml+xml application/xml audio/mpeg text/html text/plain text/x-component text/x-vcard text/xml

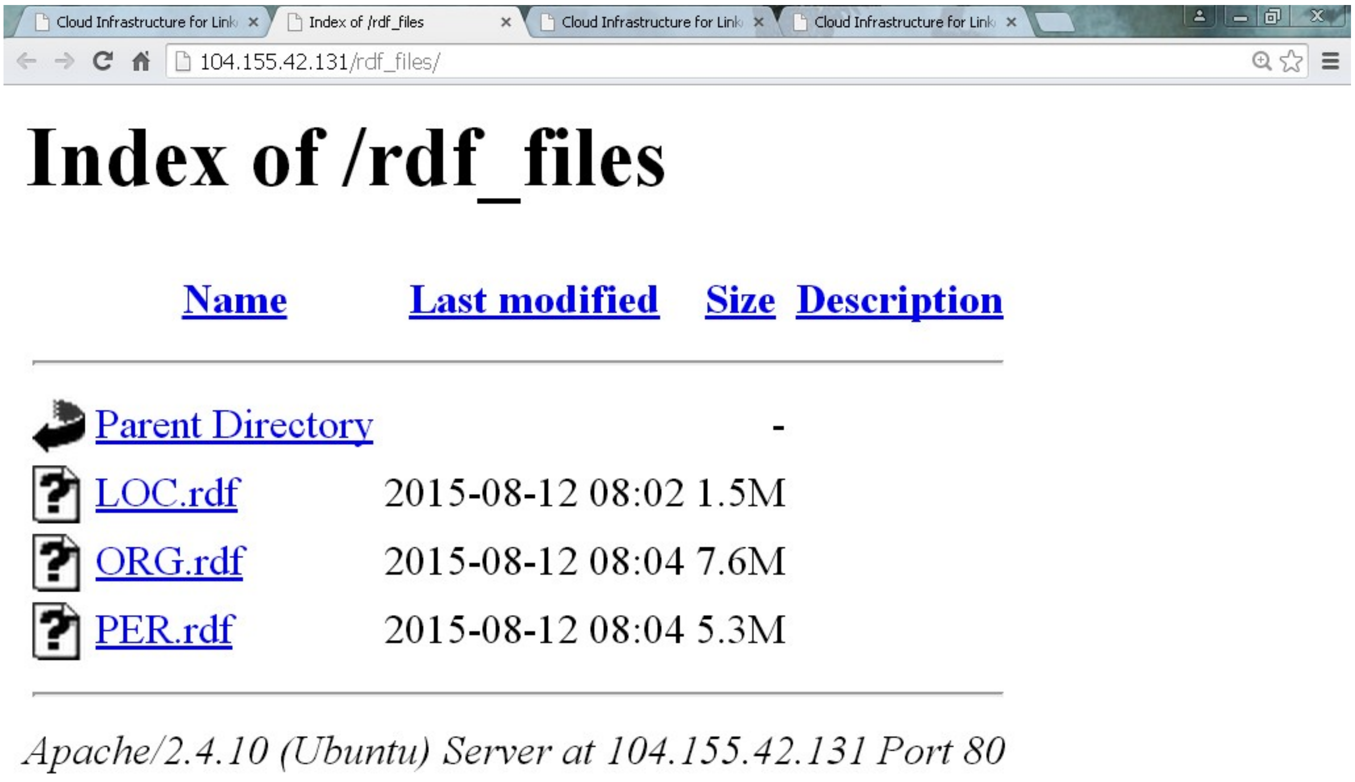
type part of a host name for searching | search by host name

http://193.40.10.165/SPC/Hum/SPC_14614.pdf
accessed Mon, 10 Aug 2015 08:12:36 GMT
Content-Type application/pdf
local dataset: ada6bfdc1fce5c12fce0af6dc9555b2ae5354ce4489eab0b7a3a5e9d
http://rantell.ee/wp-content/uploads/2013/10/PRISMA-Certificate-Authorized-partner-2013-2014.pdf
accessed Tue, 11 Aug 2015 18:49:23 GMT
Content-Type application/pdf
local dataset: 8d98a62e373477a254a73627151664bb1700e30d1231dd1700e3af4b
http://saarepaat.ee/en/wp-content/uploads/2014/01/Saare_41_brochure.pdf
accessed Tue, 11 Aug 2015 18:19:23 GMT
Content-Type application/pdf
local dataset: b28eb19cebb224c303d75bf2515ec67ee819b82c49b20ee5f407791d
http://www.avelin.ee/files/allu_sc_en.pdf
accessed Tue, 11 Aug 2015 22:38:09 GMT
Content-Type application/pdf
local dataset: 67dfc42422d81a82fd783619a13a2d8af7543abbfd163565aa53db38
http://www.bbp.ee/intelex/docs/Beddy_Bear_kasutusjuhend.pdf
accessed Tue, 11 Aug 2015 23:13:58 GMT
Content-Type application/pdf

6.8.6 Raw View of RDF files in a Browser

In Figure 18 is shown the browser view of a page of the RDF files.

Figure 18: Browser page of RDF files.


Cloud Infrastructure for Link
Index of /rdf_files
Cloud Infrastructure for Link
Cloud Infrastructure for Link
104.155.42.131/rdf_files/

Index of /rdf_files

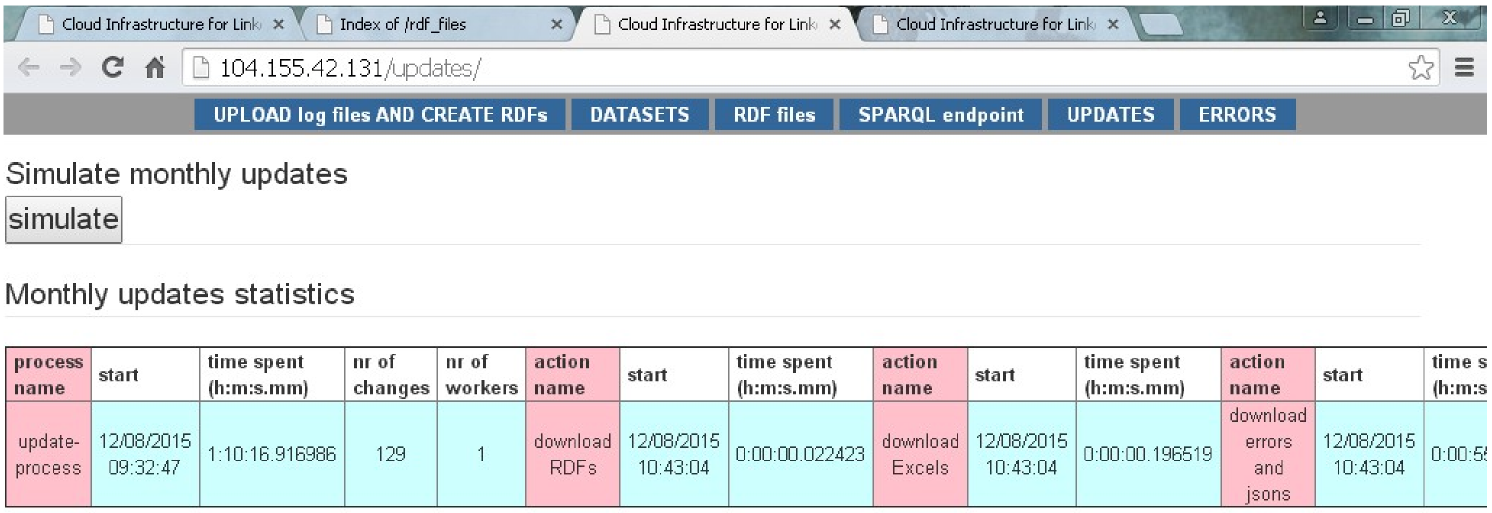
Name	Last modified	Size	Description
Parent Directory		-	
LOC.rdf	2015-08-12 08:02	1.5M	
ORG.rdf	2015-08-12 08:04	7.6M	
PER.rdf	2015-08-12 08:04	5.3M	

Apache/2.4.10 (Ubuntu) Server at 104.155.42.131 Port 80

6.8.7 View of Statistics of Processes of Monthly Updates

In Figure 19 is shown the table of monthly updates. Additionally, user can simulate this update-process manually.

Figure 19: Browser view of simulating monthly updates and statistics of performed updates.


Cloud Infrastructure for Link
Index of /rdf_files
Cloud Infrastructure for Link
Cloud Infrastructure for Link
104.155.42.131/updates/
UPLOAD log files AND CREATE RDFs
DATASETS
RDF files
SPARQL endpoint
UPDATES
ERRORS
simulate

Simulate monthly updates

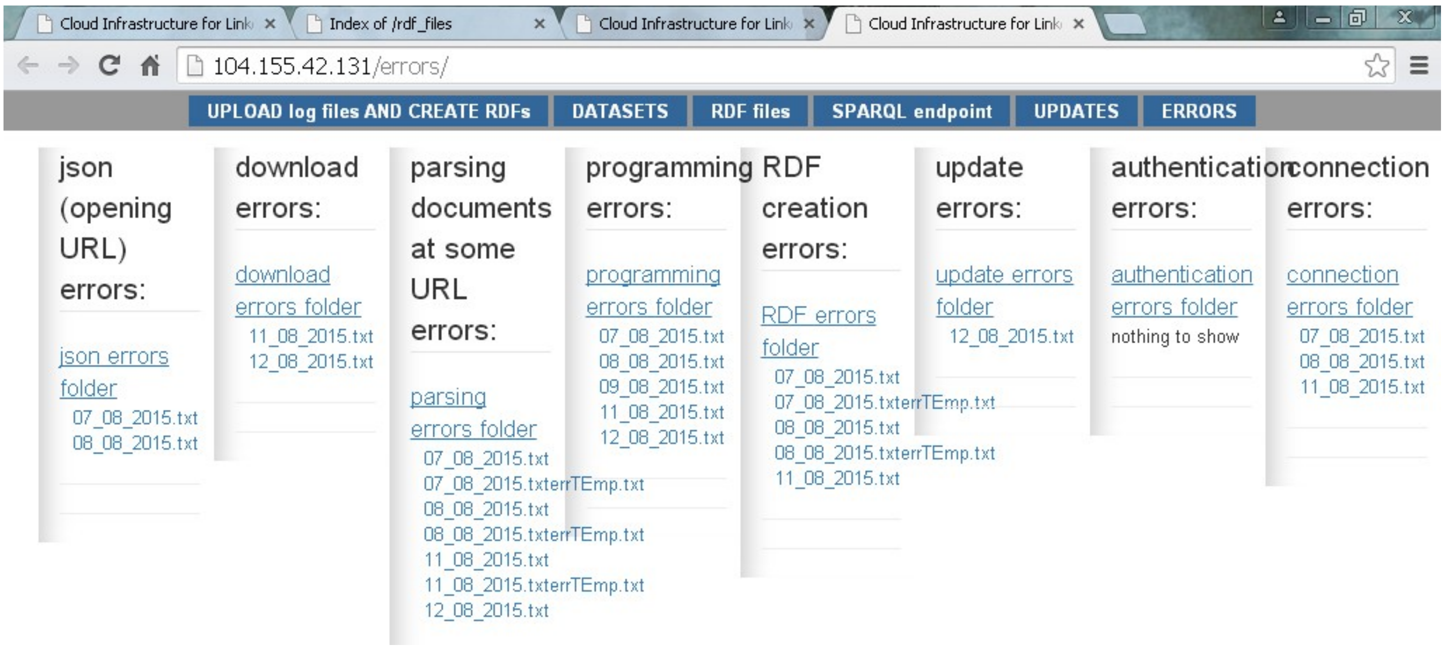
Monthly updates statistics

process name	start	time spent (h:m:s.mm)	nr of changes	nr of workers	action name	start	time spent (h:m:s.mm)	action name	start	time spent (h:m:s.mm)	action name	start	time s (h:m:s
update-process	12/08/2015 09:32:47	1:10:16.916986	129	1	download RDFs	12/08/2015 10:43:04	0:00:00.022423	download Excels	12/08/2015 10:43:04	0:00:00.196519	download errors and jsons	12/08/2015 10:43:04	0:00:5

6.8.8 View for Reading Errors

In Figure 20 is presented the browser view of several errors.

Figure 20: Browser view of generated errors.


Cloud Infrastructure for Link
Index of /rdf_files
Cloud Infrastructure for Link
Cloud Infrastructure for Link
104.155.42.131/errors/
UPLOAD log files AND CREATE RDFs
DATASETS
RDF files
SPARQL endpoint
UPDATES
ERRORS

json (opening URL) errors:

json errors folder
07_08_2015.txt
08_08_2015.txt

download errors:

download errors folder
11_08_2015.txt
12_08_2015.txt

parsing documents at some URL errors:

parsing errors folder
07_08_2015.txt
07_08_2015.txterrTEmp.txt
08_08_2015.txt
08_08_2015.txterrTEmp.txt
11_08_2015.txt
11_08_2015.txterrTEmp.txt
12_08_2015.txt

programming errors:

programming errors folder
07_08_2015.txt
08_08_2015.txt
09_08_2015.txt
11_08_2015.txt
12_08_2015.txt

RDF creation errors:

RDF errors folder
07_08_2015.txt
07_08_2015.txterrTEmp.txt
08_08_2015.txt
08_08_2015.txterrTEmp.txt
11_08_2015.txt

update errors:

update errors folder
12_08_2015.txt

authentication errors:

authentication errors folder
nothing to show

connection errors:

connection errors folder
07_08_2015.txt
08_08_2015.txt
11_08_2015.txt

7 Evaluation

In this chapter the structure and essence of input data are introduced. Then, in the next subchapter, the measured results of the performance tests and fulfilment of requirements are given.

7.1 Input Data

Author got the input data file from her supervisor. It is a log file with over 3 107 956 lines in it with the size of ~1 GB (886 7552 KB). This log file is a result of a web crawler program and its structure is explained in [95] and [96].
An example of a row in a log file; the columns are separated by white spaces:

```
  2014-05-08T21:35:57.220Z       1       58       dns:www.autolaige.ee       P
http://www.autolaige.ee/       text/dns       #048       20140508213556681+234
sha1:C2LVXEPH6TEQROI4UA62E5U35432MVJO - content-size:58
```

Every line in the log file is divided into columns; the following list is a log file row structure description:

- (0) timestamp
- (1) status code
- (2) size of the downloaded document in bytes
- **(3)** URL of downloaded file (document, in chapter 6.7.1.1 it is **(4)**)
- **(4)** R - Redirect
    - E - Embed
    - X - Speculative embed (aggressive/Javascript link extraction)
    - L - Link
    - P - Prerequisite (like for DNS or robots.txt before another URI)
- **(5)** basic url, reference (in chapter chapter 6.7.1.1 it is **(1)**)
- (6) content type of basic url, column **(5)**
- (7) the id of the worker thread that downloaded this document
- (8) timespan+download_time, difference to column (0)
- (9) sha1
- (10) -
- (11) content-size

Necessary columns for this service are

- **(3)**URL of downloaded file (document)
- **(4)**
    - R - denotes whether URL was redirected or not
    - reject lines that in this column include X or P
- **(5)** basic url
- **(6)** content type

7.2 *Statistics and Performance*

The author made several tests in Google Compute Engine environment for measuring performance of a whole service and the performance of the updating process. Also the sizes of RDF graph files as a product of this service and amount of downloads are presented. In the first subchapter the virtual machine features used in these tests are given.

7.2.1Machine types

This service used master VM with 1 CPU, 3 worker VMs with 2 CPU in each and one worker VM with 1 CPU. The performance measures are based on following virtual machine features:

name: master
machineType (GCE notation): n1-standard-1 (1 vCPU, 3.75 GB memory)
disk size: 1000 GB
Estimated performance:

Operation Type	Read	Write
Sustained random IOPS limit	300	1,500
Sustained throughput limit (MB/s)	120	90

names: worker1, worker2, worker3
machineType (GCE notation): n1-highcpu-2 (2 vCPUs, 1.8 GB memory)
disk size: 2800 GB
Estimated performance:

Operation Type	Read	Write
Sustained random IOPS limit	840	4,200
Sustained throughput limit (MB/s)	180	120

name: worker4
machineType (GCE notation): n1-standard-1 (1 vCPU, 3.75 GB memory)
disk size: 840 GB
Estimated performance:

Operation Type	Read	Write
Sustained random IOPS limit	252	1,260
Sustained throughput limit (MB/s)	100.8	75.6

7.2.2Test Results of the Whole Service

These results include reading log file, sending valid URLs to workers, RDFizing process in workers, collecting RDF graph files from workers, downloading Excel and PDF files from workers, downloading json-formatted metdata files from GCE Storage, downloading Error files from GCE Storage. The test results are presented in Table 5.

Test started: 2015-08-11 03:12

Generating RDF files:

Time spent: 8.5 hours

Size of

ORG.rdf:	7816 KB ~ 7.6 MB
PER.rdf:	5440 KB ~ 5.3 MB
LOC.rdf:	1568 KB ~ 1.5 MB
total :	14824 KB ~ 14.5 MB

Number of triples in:

ORG.rdf:	112 555
PER.rdf:	88 199
LOC.rdf:	23 444
total :	224 198

Importing-aggregating RDF files:

time spent (h:m:s.mm): 0:09:00.799224

Downloading Excel and PDF files from worker VMs,
json-formatted metadata files and error files from GCE Cloud Storage:

Time spent (h:m:s.mm): 1:17:53.424858

Number of downloaded datasets: 4405

Number of json-formatted datasets: 295

7.2.3Test Results of the Update Process

This test was ran after the previous , described in chapter 7.2.3 finished, on the same day. The timespans are small because of little nr of changes.

Generating RDF files during the update process:
Start: 12/08/2015 09:32:47
Time spent (h:m:s.mm): 1:10:16.916986
Nr of changes: 129

Downloading of RDFs:
Start: 12/08/2015 10:43:04
Time spent (h:m:s.mm): 0:0:00.0

Downloading of Excel and PDF files:
Start: 12/08/2015 10:43:04
Time spent (h:m:s.mm): 0:00:00.196519

Downloading of errors and metadata **files:**
Start: 12/08/2015 10:43:04
Time spent (h:m:s.mm): 0:00:55.049540

7.2.4Performance Measures of SPARQL Endpoint

It would be better, when the response to the query were faster, but it also depends on the client's bandwidth. The results include displaying response on web page. The results show, that time spent on receiving response, depends on RDF graph.

Tests for every SARQL endpoint functionality:

organization „Bmw“

```
        SELECT DISTINCT *
        WHERE
        {
        {{?obj ner:locationName "Bmw"} UNION {?obj ner:lemma "Bmw"}} .
        ?obj ner:mentionedAtSite ?webpage
        }
        ORDER BY ?webpage
```

The query took 14 seconds.

Person Angelica Udeküll

```
SELECT DISTINCT *
        WHERE
        {
        {{?obj foaf:givenName ?gname} UNION {?obj ner:lemma ?gname}} .
        {{?obj   foaf:familyName   "Udeküll"}   UNION   {?obj   ner:lemma
"Udeküll"}} .
        ?obj ner:mentionedAtSite ?webpage
        }
ORDER BY ?webpage
```

The query took 14 seconds.

Person Angelica

```
        SELECT DISTINCT *
        WHERE
        {
        {{?obj   foaf:givenName   "Angelica"}   UNION   {?obj   ner:lemma
"Angelica"}} .
        {{?obj foaf:familyName ?fname} UNION {?obj ner:lemma ?fname}} .
        ?obj ner:mentionedAtSite ?webpage
        }
ORDER BY ?fname ?webpage
```

The query took 17 seconds.

Person Udeküll

```
        SELECT DISTINCT *
        WHERE
        {
        {{?obj foaf:givenName ?gname} UNION {?obj ner:lemma ?gname}} .
```

```
	{{?obj   foaf:familyName   "Udeküll"}   UNION   {?obj   ner:lemma
"Udeküll"}} .
	?obj ner:mentionedAtSite ?webpage
	}
ORDER BY ?gname ?webpage
```

The query took 20 seconds.

Location Sillamäe

```
	SELECT DISTINCT *
	WHERE
	{
	{{?obj   ner:locationName   "Sillamäe"}   UNION   {?obj   ner:lemma
"Sillamäe"}} .
	?obj ner:mentionedAtSite ?webpage
	}
ORDER BY ?webpage
```

The query took 2 seconds.

All organizations

```
	SELECT DISTINCT *
	WHERE
	{
	?obj ner:orgName ?name .
	?obj ner:mentionedAtSite ?webpage
	}
ORDER BY ?name ?webpage
```

The query took 30 seconds.

All people

```
	SELECT DISTINCT *
	WHERE
	{
	?obj foaf:familyName ?fname .
	?obj foaf:givenName ?gname .
	?obj ner:mentionedAtSite ?webpage
	}
ORDER BY ?fname ?gname ?webpage
```

The query took 24 seconds.

All locations

```
	SELECT DISTINCT *
```

```
        WHERE
        {
        ?obj ner:locationName ?name .
        ?obj ner:mentionedAtSite ?webpage
        }
ORDER BY ?name ?webpage
```

The query took 7 seconds.

Query locations on web document "http://www.wws.ee/service/cruise/essentials/"

```
        SELECT ?name
        WHERE
        {
        ?obj                                        ner:mentionedAtSite
<http://www.wws.ee/service/cruise/essentials/> .
        ?obj ner:locationName ?name .
}
```

The query took 3 seconds.

Query people on web document "http://online.le.ee/"

```
        SELECT ?gname ?fname
                WHERE
                {
                ?obj foaf:givenName ?gname .
                ?obj foaf:familyName ?fname .
                ?obj ner:mentionedAtSite <http://online.le.ee/>
        }
```

The query took 6 seconds.

Query organizations on web document "http://online.le.ee/"

```
        SELECT ?name
                WHERE
                {
                ?obj ner:mentionedAtSite <http://online.le.ee/> .
                ?obj ner:orgName ?name .
        }
```

The query took 12 seconds.

Query all names on web document http://online.le.ee/

```
        SELECT ?name
                WHERE
                {
                ?obj ner:mentionedAtSite <http://online.le.ee/> .
                ?obj ner:orgName ?name .
```

```
}

SELECT ?name
      WHERE
      {
      ?obj ner:mentionedAtSite <http://online.le.ee/> .
      ?obj ner:locationName ?name .
}

SELECT ?gname ?fname
      WHERE
      {
      ?obj foaf:givenName ?gname .
      ?obj foaf:familyName ?fname .
      ?obj ner:mentionedAtSite <http://online.le.ee/>
}
```

The query took 30 seconds.

Query organizations and locations on web document "http://online.le.ee/"

```
SELECT ?name
      WHERE
      {
      ?obj ner:mentionedAtSite <http://online.le.ee/> .
      ?obj ner:orgName ?name .
}

SELECT ?name
      WHERE
      {
      ?obj ner:mentionedAtSite <http://online.le.ee/> .
      ?obj ner:locationName ?name .
}
```

The query took 12 seconds.

Query locations and people on web document "http://online.le.ee/"

```
SELECT ?gname ?fname
      WHERE
      {
      ?obj foaf:givenName ?gname .
      ?obj foaf:familyName ?fname .
      ?obj ner:mentionedAtSite <http://online.le.ee/>
}

SELECT ?name
      WHERE
      {
      ?obj ner:mentionedAtSite <http://online.le.ee/> .
      ?obj ner:locationName ?name .
}
```

The query took 18 seconds.

7.2.5Fulfilment of Requirements

In this chapter the fulfilment of settled requirements are brought in Table 6.

id	result achieved?	comment
upIN	yes	Implemented file uploading into local (master) file system, currently uploading to BigQuery is manual
genMD	yes	Metadata stored in Cloud STorage bucket
updateMD	yes	-
readMD	yes	-
genRDF	yes	RDF files are stored in master file system
updateRDF	yes	-
readRDFm	yes	-
readRDFh	yes	Reading is implemented in SPARQL endpoint GUI and webservice
downD	yes	Currently downloads to maste server file system, may consider other storage option
readD	yes	Datasets are sortable due to the matadata model
genE	yes	Errors are stored into GCE Cloud Storage bucket
readE	yes	-

Table 6: Fulfilment of requirements.

8 Conclusions and Future Work

Eventually the development resulted in a cloud service that reads the log file and manages sending jobs to workers, where the process of reading OGD documents, extracting entities and generating RDF graphs takes place. The service can perform monthly updates. This cloud service was tested in Google Compute Engine environment.

Author is quite satisfied with the performance test results: using 7 CPUs in the cloud of workers it took approximately 9.5 hours to process 3 107 956 lines of a log file. It could be even better when more CPUs in the workers' cloud is applied, but then a balance must be found between the cost and performance requirements. The Google Compute enables using only a few CPUs (8) during the free trial period (2 months) setting constraints for the project's performance.

The resulting value of this system processes are 3 RDF files for organizations, locations and people. The graphs can be used in SPARQL endpoint by human users (graphical UI) or by machines (web services that result in json or RDF/XML format). The RDF graphs are linkable to other RDF graphs and data in order to increase the quality, transparency, accessibility and reliability of Estonian Open Government Data.

In future this system should be tested without POST and GET requests, but using worker servers as if they were a part of LAN. It may reduce the processing time of the whole system. There is a Python package developed for such kind of data transfer: "Pyro" [97].

It would be also interesting what the performance measures would be when adapting this system into MapReduce technology.

Furthermore, there are much more entities in OGD documents than people, organizations, locations. These may be also extracted into RDF graphs in future.

As time goes on, the RDF graph files grow larger and a special system for storage and more time-effective query architecture becomes necessary.

The software is available at GitHub https://github.com/Mailis/EstNer/tree/master/cloud.

9 References

[1] Open Government Data. [http://opengovernmentdata.org/]. (Accessed in December, 2014.)

[2] W3C. RDF. [http://www.w3.org/RDF/]. (Accessed in December, 2014).

[3] ScienceDaily. Web crawler. [http://www.sciencedaily.com/terms/web_crawler.htm]. (Accessed in December, 2014).

[4] T.Petmanson, A.Tkachenko. [https://github.com/estnltk/estnltk]. (Accessed in December, 2014).

[5] GitHub. RDFLib. [https://github.com/RDFLib/rdflib]. (Accessed in January, 2015.)

[6] Google. Google Cloud Platform. Compute Engine. [https://cloud.google.com/compute/]. (Accessed in March, 2015.)

[7] Open Data HandBook. What is Open Data? [http://opendatahandbook.org/guide/en/what-is-open-data/]. (Accessed in July, 2015).

[8] Open Government Data. Why Open Government Data? [http://opengovernmentdata.org/#sthash.gkPL2Geh.dpuf]. (Accessed in July, 2015).

[9]W3C. Semantic Web. [http://www.w3.org/2001/sw/]. (Accessed in June, 2015.)

[10] T. Berners-Lee, J. Hendler, O. Lassila. The Semantic Web. Scientific America, 2001.

[11] T. Berners-Lee. [http://www.w3.org/DesignIssues/LinkedData.html] 2006. (Accessed in October, 2014).

[12] N. Shadbolt, W. Hall, T. Berners-Lee. The Semantic Web Revisited. IEEE Intelligent Systems. [http://eprints.soton.ac.uk/262614/1/Semantic_Web_Revisted.pdf]. (Accessed in June, 2015.)

[13] T.Heath, C.Bizer. Linked Data: Evolving the Web into a Global Data Space. [http://linkeddatabook.com/editions/1.0/#htoc40]. (Accessed in February, 2015.)

[14] W3C. Ontologies. [http://www.w3.org/standards/semanticweb/ontology]. (Accessed in June, 2015.)

[15] FOAF Vocabulary Specification 0.99. [http://xmlns.com/foaf/spec]. (Accessed in November, 2014.)

[16] W3C. RDF Schema 1.1. [http://www.w3.org/TR/rdf-schema/]. (Accessed in January, 2015.)

[17] Dublin Core. DCMI Metadata Terms. [http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=elements#description]. (Accessed in November, 2014.)

[18] SIOC Core Ontology Specification. [http://rdfs.org/sioc/spec/]. (Accessed in June, 2015.)

[19] W3C. OWL Web Ontology Language Reference. [http://www.w3.org/TR/owl-ref/] . (Accessed in November, 2014.)

[20] Nerd. Ontology. [http://nerd.eurecom.fr/ontology]. (Accessed in July, 2014).

[21] W3C. Category: RDF Generator. [http://www.w3.org/2001/sw/wiki/Category:RDF_Generator]. (Accessed in June, 2015.)

[22] Karlsruher Institut für Technologie. LODifier. [http://www.aifb.kit.edu/web/LODifier]. (Accessed in June, 2015.)

[23] Fondazione Bruno Kessler, Data & Knowledge Management. The KnowledgeStore Project. [https://knowledgestore.fbk.eu/index.html]. (Accessed in June, 2015.)

[24] E.Agichtein, L.Gravano. Snowball: extracting relations from large plain-text collections. Department of Computer Science, Columbia University. [http://www.cs.columbia.edu/~gravano/Papers/2000/dl00.pdf]. (Accessed in June, 2015).

[25] Thomson Reuters. Open Calais. [http://new.opencalais.com/].(Accessed in June, 2015.)

[26] Ontos. OntosLDIW. [http://www.ontos.com/]. (Accessed in June, 2015.)

[27] DBpedia. DBpedia Spotlight. [http://wiki.dbpedia.org/projects/dbpedia-spotlight]. (Accessed in June, 2015.)

[28] M.H. Butler, J. Gilbert, A. Seaborne, K. Smathers. Data conversion, extraction and record linkage using XML and RDF tools in Project SIMILE. Digital Media Systems Laboratory, 2004. [http://www.hpl.hp.com/techreports/2004/HPL-2004-147.pdf]. (Accessed in June, 2015.)

[29] W3C. ConverterToRdf. [http://www.w3.org/wiki/ConverterToRdf]. (Accessed in June, 2015.)

[30] C. Lange. Krextor – An Extensible XML→RDF Extraction Framework
. Heraklion, Greece, May 31, 2009. [http://ceur-ws.org/Vol-449/ShortPaper2.pdf]. (Accessed in June, 2015.)

[31] J.Tscherrig, P.Cudr´e-Mauroux, E.Mugellini, O.A.Khaled, M.Sokhn. SemantiConverter: A Flexible Framework to Convert Semi-Structured Data into RDF. University of Fribourg, Switzerland. [http://exascale.info/papers/Json2RDFSConverter.pdf]. (Accessed in June, 2015.)

[32] Wikimedia Foundation, Inc. Wiki markup. [https://en.wikipedia.org/wiki/Wiki_markup]. (Accessed in June, 2015.)

[33] W3C. A Direct Mapping of Relational Data to RDF. [http://www.w3.org/TR/rdb-direct-mapping/]. (Accessed in June, 2015.)

[34] A. Tkachenko. Named Entity Recognition for the Estonian Language. Master thesis.Tartu 2010.

[35] T. Petmanson. Estnltk — Open source tools for Estonian natural language processing . [http://tpetmanson.github.io/estnltk/index.html]. (Accessed in December, 2014).

[36] Estnltk. Named entity recognition. [http://tpetmanson.github.io/estnltk/tutorials/ner.html]. (Accessed in November, 2014).

[37] Estnltk. Paragraph, sentence and word tokenization. [http://tpetmanson.github.io/estnltk/tutorials/tokenization.html]. (Accessed in November, 2014).

[38] Estnltk. Morphological analysis. [http://tpetmanson.github.io/estnltk/tutorials/morf_analysis.html]. (Accessed in November, 2014).

[39] W3C. RDF/XML Syntax Specification (Revised). [http://www.w3.org/TR/REC-rdf-syntax/]. (Accessed in January, 2015.)

[40] W3C. 3. N-Triples. [http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/#ntriples]. (Accessed in January, 2015.)

[41] W3C. SPARQL Query Language for RDF. [http://www.w3.org/TR/rdf-sparql-query/]. (Accessed in December, 2014.)

[42] EASY RDF. A PHP library designed to make it easy to consume and produce RDF. [http://www.easyrdf.org/]. (Accessed in June, 2015.)

[43] GitHub. RDFLib. [https://github.com/RDFLib/rdflib]. (Accessed in January, 2015.)

[44] J.Sequeda. Introduction to: Triplestores. Dataversity, 2013. [http://www.dataversity.net/introduction-to-triplestores/].

[45] W3C. LargeTripleStores. [http://www.w3.org/wiki/LargeTripleStores]. (Accessed in June, 2015.)

[46] Linking Open Government Data. Installing and Managing Virtuoso SPARQL Endpoint. [https://logd.tw.rpi.edu/tutorial/installing_using_virtuoso_sparql_endpoint]. (Accessed in June, 2015.)

[47] Oracle Spatial and Graph: Benchmarking a Trillion Edges RDF Graph. Oracle White Paper, 2014. [http://download.oracle.com/otndocs/tech/semantic_web/pdf/OracleSpatialGraph_RDFgraph_1_trillion_Benchmark.pdf]. (Accessed in June, 2015.)

[48] Franz Inc. AllegroGraph. [http://franz.com/agraph/allegrograph/]. (Accessed in June, 2015.)

[49] Complexible Inc. Stardog. [http://stardog.com/]. (Accessed in June, 2015.)

[50] T.Heath, C.Bizer. Linked Data: Evolving the Web into a Global Data Space. [http://linkeddatabook.com/editions/1.0/#htoc87]. (Accessed in February, 2015.)

[51] W3C. Linked Data Platform Use Cases and Requirements, 2014. [http://www.w3.org/TR/ldp-ucr/]. (Accessed in July, 2015).

[52] European Commission. Linked Study on business models for Linked Open Government Data, 2013. [http://ec.europa.eu/isa/documents/study-on-business-models-open-government_en.pdf/]. (Accessed in July, 2015).

[53] Google Cloud Platform. Cloud Storage. [https://cloud.google.com/storage/]. (Accessed in March, 2015.)

[54] Google Cloud Platform. Frequently Asked Questions. [https://cloud.google.com/compute/docs/faq#disks]. (Accessed in March, 2015.)

[55] Google Cloud Platform. Google BigQuery. [https://cloud.google.com/bigquery/]. (Accessed in March, 2015.)

[56] Google Compute Engine. Persistent Disks. [https://cloud.google.com/compute/docs/disks/persistent-disks#pdperformance] . (Accessed in March, 2015.)

[57] Google Compute Engine. HTTP/HTTPS Load Balancing. [https://cloud.google.com/compute/docs/load-balancing/http/]. (Accessed in March, 2015.)

[58] Google Compute Engine. Load Balancing Health Checks. [https://cloud.google.com/compute/docs/load-balancing/health-check]. (Accessed in March, 2015.)

[59] DBpedia. About. [http://wiki.dbpedia.org/about] (Accessed in June, 2015.)

[60] DBpedia. Mapping et. DBpedia Estonian needs your help! [http://mappings.dbpedia.org/index.php/Mapping_et]. (Accessed in July, 2015.)

[61] DBpedia. Mapping Statistics for et. [httphttp://mappings.dbpedia.org/server/statistics/et/]. (Accessed in July, 2015.)

[62] The Trustees of Princeton University. WordNet. [http://wordnet.princeton.edu/wordnet/]. (Accessed in April, 2015.)

[63] The Trustees of Princeton University. WordNet RDF. [http://wordnet-rdf.princeton.edu/]. (Accessed in April, 2015.)

[64] Research Group of Computational Linguistics, University of Tartu. Teksaurus. [http://www.cl.ut.ee/ressursid/teksaurus/]. (Accessed in June, 2014).

[65] Max Planck Institute for Informatics. YAGO: A High-Quality Knowledge Base. [http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/]. (Accessed in July, 2015.)

[66] Cycorp. OpenCyc. [http://www.cyc.com/platform/opencyc/]. (Accessed in July, 2015.)

[67] Openlink Software. Virtuoso Programmer's Guide - RDF Middleware ("Sponger"). [http://virtuoso.openlinksw.com/whitepapers/sponger%20cartridge%20programmers%20guide%20rdf%20middleware.html#mozTocId114]. (Accessed in April, 2015.)

[68] GitHub. openlink/Virtuoso-RDFIzer-Mapper-Scripts. Readme. [https://github.com/openlink/Virtuoso-RDFIzer-Mapper-Scripts]. (Accessed in April, 2015.)

[69] Openlink Software. Virtuoso Programmer's Guide - RDF Middleware ("Sponger"). [http://virtuoso.openlinksw.com/whitepapers/sponger%20cartridge%20programmers%20guide%20rdf%20middleware.html#mozTocId119]. (Accessed in April, 2015.)

[70] Openlink Software. Virtuoso Programmer's Guide - RDF Middleware ("Sponger"). [http://virtuoso.openlinksw.com/whitepapers/Virtuoso%20Sponger.html]. (Accessed in April, 2015.)

[71] Freebase. [https://plus.google.com/109936836907132434202/posts/3aYFVNf92A1]. (Accessed in April, 2015.)

[72] Google Official Blog. Deeper understanding with Metaweb. [http://googleblog.blogspot.com/2010/07/deeper-understanding-with-metaweb.html]. (Accessed in April, 2015.)

[73] Google Official Blog. Introducing the Knowledge Graph: things, not strings. [http://googleblog.blogspot.com/2010/07/deeper-understanding-with-metaweb.html]. (Accessed in April, 2015.)

[74] GeoNames . The GeoNames Ontology. [http://www.geonames.org/ontology/documentation.html]. (Accessed in July, 2015.)

[75] TWC LOGD. Linking Open Government Data. [https://logd.tw.rpi.edu/home]. (Accessed in April, 2015.)

[76] Poolparty. PoolParty Extractor. [http://www.poolparty.biz/portfolio-item/poolparty-extractor/]. (Accessed in July, 2015.)

[77] Github. Metafacture-core. [https://github.com/culturegraph/metafacture-core/blob/master/README.md]. (Accessed in May, 2015.)

[78] BBC. [http://www.bbc.com/]. (Accessed in May, 2015.)

[79] Europeana Labs. Europeana Linked Open Data. [http://labs.europeana.eu/api/linked-open-data/introduction/]. (Accessed in March, 2015.)

[80] European Commission. Health an food safety. [http://ec.europa.eu/dgs/health_food-safety/index_en.htm]. (Accessed in April, 2015.)

[81] Health and Food Safety. DG Health and Food Safety Data. [http://ec.europa.eu/dgs/health_food-safety/information_systems/index_en.htm]. (Accessed in May, 2014).

[82] CKAN. [http://ckan.org/]. (Accessed in December, 2014.)

[83] UK Government. data.gov.uk. [http://data.gov.uk/]. (Accessed in November, 2014.)

[84] publicdata.eu. Search Europe's Public Data. [http://publicdata.eu/]. (Accessed in May, 2015.)

[85] Helsinki Region Infoshare online service. [http://www.hri.fi/en/]. (Accessed in November, 2014.)

[86] International Aid Transparency Initiative. [http://iatiregistry.org/]. (Accessed in May, 2015.)

[87] CKANv2.2. FileStore and file uploads. [http://docs.ckan.org/en/ckan-2.2/filestore.html#setup-file-uploads]. (Accessed in December, 2014.)

[88] P.Kruchten. The 4+1 View Model of Architecture. IEEE Software, Volume 12, Number 6, pp. 42-50. 1995.

[89] StackExchange. Mapping between 4+1 architectural view model & UML. [http://programmers.stackexchange.com/questions/233257/mapping-between-41-architectural-view-model-uml]. (Accessed in January, 2015.)

[90] Google Cloud Platform. Compute Engine. API Reference. [https://cloud.google.com/compute/] (Accessed in March, 2015.)

[91] Google Cloud Platform. Compute Engine. Using the Python Client Library. [https://cloud.google.com/compute/docs/tutorials/python-guide]. (Accessed in March, 2015.)

[92] Google Compute Engine. API Reference. [https://cloud.google.com/compute/docs/reference/latest/?hl=en_US]. (Accessed in February, 2014).

[93] W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition). 2.2 Characters. [http://www.w3.org/TR/REC-xml/#NT-Char]. (Accessed in August, 2015.)

[94] RDFlib. RDF terms in rdflib. URIRefs. [http://rdflib.readthedocs.org/en/latest/rdf_terms.html#urirefs]. (Accessed in August, 2015.)

[95] Crawler. Analysis of jobs. [http://crawler.archive.org/articles/user_manual/analysis.html]. (Accessed in May, 2015.)

[96] Crawler. Discovery path. [http://crawler.archive.org/articles/user_manual/glossary.html#discoverypath]. (Accessed in December, 2014.)

Resümee

Siin magistritöös kirjeldatakse pilveteenust, mille eesmärk on muuta struktureerimata avalikud dokumendid seotud graafiks, mille andmeid on võimalik pärida ja töödelda lisaks inimesele ka masinatel. Selleks toodab pilvesüsteem RDF graafid, mis on semantiliselt hästi kirjeldatud, mille semantika on avalikult loetav/kättesaadav ontoloogiates ja sematilistes sõnavarades. Pilve infrastruktuuri jaoks töötati välja süsteem, mis kasutab nn. meister-virtuaalmasinat ja töölis-virtuaalmasinate pilve. Seda testiti Googli Compute Engine keskkonnas. Meister-viruaalmasin loeb ridu, mida on üle 3,1 miljoni, logifailist ja saadab sealt igalt realt leitud URLid töölis-virtuaalmasinatele. Töölis-serverid loevad dokumentide sisu ja eraldavad neist isiku nimed ning organisatsiooni ja koha nimetused. Need nimed seotakse selle veebilehega RDF graafides, mida saab SPARQL pärimiskeele abil küsitleda/otsingut teostada. SPARQL päringuks arendati nii graafiline kasutajaliides kui veebiteenused masinatele lugemiseks formaatides json ja RDF/XML.

RDF kolmikuid tuli kokku 224 198 tükki, neist organisatsioonide nimetuste komikuid 112 555, kohanimetuste omi 23 444 ja isikunimede kolmikuid 88 199.

Üks oluline väljakutse lisaks pilveteenuse enda arendamisele oli see, et suures logifailist leitud suuremahuliste/vähemmahuliste töötlemine toimuks võimalikult kiiresti. Tulemuseks tuli, et umbes 3,1 miljoni rea töötlemine võttis 7 CPUga tööliste pilves aega ligikaudu 9,5 tundi. SPARQL päringule vastuse ootamine kestab 2-30 sekundit, sõltuvalt RDF graafi faili suurusest ja sellest, kui täpne-üldine päringu lause saadeti.

Saadud RDF graafe saab kasutada linkimiseks andmetega mujal ja integreerida teiste RDF graafidega.