KTH Computer Science
and Communication

Semantic parsing of Swedish laws

Using RDF to format legal documents in accordance with applicable standards to contribute to a more semantic web

MIKAEL FALGARD

Master's Thesis at NADA
Supervisor: Sten Andersson
Examiner: Anders Lansner

TRITA xxx yyyy-nn

Abstract

This project has been carried out as a thesis project at the School of Computing at the Royal Institute of Technology in Stockholm (KTH) on behalf of Notisum AB. The project aims to investigate whether Notisum's current process to parse and format downloaded legal documents can be improved by a more modern process. Part of the process is to mark up the data in the legal documents according to Rättsinformationsprojektets proposed standard. This will contribute to a more semantic web where all data is meaningful to both humans and computers. The Resource Description Framework (RDF), is a framework that is used to describe or model information from web sources, have been used to produce a prototype to parse and format documents (mainly laws) from Swedish Constitution (SFS) in the programming language Python. This work also included research into the phenomenon of "The Semantic Web", what it prejudge and how to proceed in order to meet its requirements. Finally, it proposes recommendations on how Notisum can proceed with the use of the developed method for parsing legislations in a meaningful way and also some improvements that could be implemented.

Referat

Utnyttja RDF för att tolka och formatera juridiska dokument i enlighet med tillämpliga standarder för att bidra till en mer semantisk webb

Detta projekt har utförts som ett examensarbete på skolan för datavetenskap och kommunikation på Kungliga Tekniska Högskolan i Stockholm (KTH) på uppdrag av Notisum AB. Projektets syfte är att undersöka huruvida Notisums nuvarande process för att formatera nerladdade juridiska dokument kan förbättras genom en modernare process. En del av processen är att märka upp datat i dokumenten enligt Rättsinformationsprojektets föreslagna standard. Detta ska bidra till en mer semantisk webb där all data är meningsfull för både människor och datorer. Resource Description Framework (RDF), ett ramverk som används för att beskriva eller modellera information från webbkällor har använts för att ta fram en prototyp för att formattera dokument (lagar) från Svensk Författnigsamling i språket Python. I arbetet ingick även forskning kring fenomenet "Den Semantiska Webben", vad den innbär och hur man kan gå till väga för att uppfylla dess krav. Slutligen föreslås rekommendationer för hur Notisum kan gå vidare med användandet av den framtagna metoden för att märka upp lagar på ett meningsfullt sätt och även vissa förbättringar som skulle kunna genomföras.

Contents

Part I

Introduction

Chapter 1

Introduction

1.1 Legal information online

According to *Rättsinformationsförordningen* (1999:175) basic legal information has to be provided bot to the public administration, and to private individuals in electronic form.[1] For example the collection of statutory law, the *Svensk Författningssamling* (SFS) is published in the Government Offices legal databases. These databases are old and hard to navigate since they only consist of plain text documents.[1]

1.2 Uniform standards

One of the goals with *Rättsinformationsförordningen* is that all legal information should be coherent, searchable from a single location and have a uniform presentation. Today's decentralized system where authorities are responsible for their own documents, leads to a couple of problems. One of them being that the documents don't follow the same format standards.

In 2006 a project called *Rättsinformationsprojektet* was commenced to develop the legal information system further. A first step was to assure that the document that is going to be published contains the required meta data for that type of document. The next step is to assure that the information is delivered in a correct format.[2] For example, date data must be unambiguously expressed in machine-readable form.

[1] `http://www.notisum.se/rnp/sls/lag/19990175.htm`

[2] Guidelines for publisher, developers etc are found on `http://dev.lagrummet.se/dokumentation/`

1.3 Rättsnätet

Rättsnätet is a free web service on `www.notisum.se`[3] that provides legal information. Rättsnätet's content consists of information gathered from authority databases and is processed in several steps from pure text information into a rich XML file. Notisum also provides a premium service that includes additional information and services.

1.4 Lagen.nu

A similar website to Notisum is lagen.nu, they also provide all Swedish laws as a free web service. A difference is that lagen.nu follows standards regarding the Semantic Web[4] in a better way than Notisum's Rättsnätet does. Lagen.nu is created by an individual with voluntary interest in law, the Semantic Web and is also involved in *rättsinformationsprojektet.*

The source code is open source[5] and the author encourages the use and reuse of it. I have gained a lot of inspiration from that project and in terms of architecture I have reused parts of it.

1.5 Problem

Today Rättsnätet use processing steps that downloads, analyzes and transforms authority information from plain text to structured and marked up XML information. This process is a collection of programs written in Pascal and C#. These programs, especially those that have been written in Pascal, are old and difficult to maintain. The conversion parts are constructed using a traditional technique similar to that used to write compilers.

Instead of using compiler technology lagen.nu use regular expressions to parse legal documents. The conversion programs lagen.nu use are written in object-oriented Python, which is more suitable for the task than Pascal that Notisum use today. Additionally lagen.nu is based on a document model from the Semantic Web, with the Resource Description Framework, RDF, as a base. Because of this the code base in lagen.nu has greater flexibility and is more about standards than Notisum.

The problem that this thesis focus on is how to modernize and replace Rättsnätet's current platform to a more modern process. A process where the result should be

[3] Notisum AB is the company that I am writing this thesis at.

[4] The Semantic Web is a collaborative movement that promotes common data formats and meaningful data online.

[5] Link to lagen.nu's license file `http://svn.lagen.nu/svnroot/trunk/LICENSE`

based on the Semantic Web and follow standards proposed by Rättsinformationsprojektet. Some aspects of the problem to keep in mind are:

- The amount of documents, since there is over 200.000 legal documents, the program requires a fast process.
- Legal source texts contains references that should be interpreted differently depending on the context, and to be marked up automatically with a certain margin of error.
- Cross-references between the 200,000+ documents require theoretically, an array of 40 billion nodes.
- The output from the process needs to be structurally similar to Notisums current process output so the post process (generating html to publish to endusers) can be used with as little tweaking as possible.

1.6 Limitations

The implementation part of the thesis will be limited to the parsing step of the process. It will not cover the downloading or the final transformation to html that is ready to be published.
To make this project feasible it is limited to handling only SFS data. Other types of data sources that a complete process would need to handle are for example:

- Propositions
- Supreme Court summaries
- Decisions of the Courts of Appeal, and the Labour Court
- European legal regulations and directives

The goal is of course to create a generic and loosely coupled code base, so that it is easy to implement these modules in a later stage.

1.7 Abbreviations and Vocabulary

To be able to understand this report, the reader needs to understand the following abbreviations. Given the complexity of a legal document[6] I will also present a vocabulary of how words are defined in the thesis.

[6] For example "a paragraph" in a Swedish legal document does not correspond to a regular paragraph in the English language.

SFS

The Swedish Code of Statutes “Svensk författningssamling” (SFS) is the official publication of all Swedish laws enacted by the Riksdag and ordinances issued by the Government[7]. Every law and ordinance has an SFS number, it consists of a four digit year, a colon, and then an incrementing number by year. For instance the Ordinance on tattooing dyes have the SFS number 2012:503[8].

SFSR

The Statute Register (SFSR) includes registry information of Swedish Code of Statutes (SFS), amendments, references to preparatory work, etc.

SFST

Statutes in full text (SFST) includes Swedish Code of Statutes (SFS) in full text, ie all applicable laws and regulations.

URI

Short for Unified Resource Identifier. Uniquely identifies resources in a collection[9], for example an hypertext transfer protocol url specifies a webpage on the internet.

XML

eXtensible Markup Language (XML) defines a set of rules for encoding documents in a format that is readable for both humans and machines[10].

HTML

HyperText Markup Language (HTML) is the main language for displaying web pages and other information that can be displayed in a web browser[11].

XHTML

eXtensible HyperText Markup Language (XHTML) is HTML written as XML. Documents written in XHTML needs to be well formed, elements must be properly nested and they always have to be closed. Because of this, XHTML can be parsed using standard XML parsers, unlike HTML, which requires a lenient HTML-specific parser. XHTML files are saved with the extension .xht.

[7] SFS - `http://en.wikipedia.org/wiki/Swedish_Code_of_Statutes`

[8] PDF copy of 2012:503 - `http://notisum.se/rnp/sls/sfs/20120503.pdf`

[9] URI - `http://www.w3.org/TR/uri-clarification/`

[10] XML - `http://en.wikipedia.org/wiki/XML`

[11] HTML - `http://en.wikipedia.org/wiki/Html`

RDF

Resource Description Framwork (RDF) is a general method to describe or model information that is implemented in web resources[12].

RDFa

Resource Description Framework - in - attributes (RDFa) provides a syntax, which basically is a set of attributes for expressing RDF properties in HTML, XHTML and XML documents.

UTF-8

A encoding that can represent every character in the Unicode character set. UTF-8 has become the dominant character encoding for the World-Wide Web.

ISO-8859-1

A character encoding that is generally intended for "Western European" languages, it encodes what it refers to as "Latin alphabet no 1" consisting of 191 characters from the Latin script.

N3

Notation3, or N3 is a shorthand non-XML serialization of RDF models, designed with human-readability in mind which makes it more compact and readable than XML RDF notation.

Dublin Core

The Dublin Core metadata terms are a set of vocabulary terms which can be used for simple resource description, or combining metadata vocabularies of different metadata standards in Semantic web implementations.[3]

The Semantic Web

The word semantic stands for "the meaning of". The semantic of something is the meaning of something. The Semantic Web = a Web with a meaning.[13]

[12] More detailed information about RDF in the ‘Background’ section

[13] Intro to the Semantic Web (helpful video) `http://www.youtube.com/watch?v=OGg8A2zfWKg`

Chapter 2

Background

2.1 The Semantic Web - A Web with a meaning

The Semantic Web is a collaborative movement led by W3C[1] that promotes common data formats on the world wide web by encouraging semantic content in web pages.[4] W3C's "Semantic Web Vision" is a future where:

- Web information has exact meaning
- Web information can be understood and processed by computers
- Computers can integrate information from the web

Natural languages have ambiguous meanings and even a human reader may in some cases have problems understanding the correct meaning of a text. If someone ask me "Do you know Zlatan?" they may refer to if I know him as a friend or if Iknow who he is. It is the same for computers when they are trying to understand the meaning of a text. By adding labels to the text we can make it easier to interpret and thus creating a semantic web, formed so that software can collect and analyze data. The aim is that system can present the answer to a query instead of a page where the answer can be found, and answer queries where the answer is spread over several documents. The semantic web is a web of data, where the data can be stored in documents in various ways. Using RDF is a way to structure data, so that it can be linked to other data with information and realations to objects.

2.1.1 RDF

Resource Description Framwork (RDF) is a general method to describe or model information that is implemented in web resources. It is based upon the idea of making statements about resources in the form of triples of subject-predicate-object.[5] The subject denotes the resource (it can be anything that can have a URI), the predicate denotes traits or aspects of the resource and also expresses a relationship

[1] The international standards body, the World Wide Web Consortium (W3C)

between the subject and the object. For example: "The sky" (subject) "has the color" (predicate) "blue" (object) could be an RDF triple.

Let's take a look at an example[2] of an RDF object describing two CDs.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:cd="http://www.recshop.fake/cd#"/>

  <rdf:Description
  rdf:about="http://www.recshop.fake/cd/Empire Burlesque"/>
    <cd:artist>Bob Dylan</cd:artist>
    <cd:country>USA</cd:country>
    <cd:company>Columbia</cd:company>
    <cd:price>10.90</cd:price>
    <cd:year>1985</cd:year>
  </rdf:Description>

  <rdf:Description
  rdf:about="http://www.recshop.fake/cd/Hide your heart"/>
    <cd:artist>Bonnie Tyler</cd:artist>
    <cd:country>UK</cd:country>
    <cd:company>CBS Records</cd:company>
    <cd:price>9.90</cd:price>
    <cd:year>1988</cd:year>
  </rdf:Description>
</rdf:RDF>
```

The two lines at the top, **xmlns:rdf** and **xmlns:cd** define from which namespace elements with the rdf and cd prefix are from. The **rdf:Description** element contains the description of the resource identified by the **rdf:about** attribute. Finally the elements **cd:artist, cd:country, cd:company** etc. are properties of the resource.

2.1.2 RDFa

RDFa is a syntax for embedding an RDF graph in an XHTML document by using XHTML attributes for expressing RDF properties. It was proposed by Mark Birbeck[3] in the form of a W3C note entitled XHTML and RDF[6].

[2] Full example can be viewed at `http://www.w3schools.com/rdf/rdf_example.asp`

[3] `http://markbirbeck.com/`

RDFa is based on attributes that are used to carry metadata in an XML/HTML/XHTML language. There are some attributes that have been reused, for example "href" and "src" that comes from the HTML syntax. Then there is also a set of new attributes that are specified for RDFa, the attributes are:[7]

- **about** - A URI specifying the resource the metadata is about
- **rel and rev** - A relationship and reverse-relationship with another resource, respectively
- **src, href and resource** - The partner resource
- **property** - A property for the content
- **content** - optional attribute that overrides the content of the element when using the property attribute
- **datatype** - optional attribute that specifies the datatype of text specified for use with the property attribute
- **typeof** - optional attribute that specifies the RDF type(s) of the subject or the partner resource

Not all (X)HTML validators recognize and know how to validate these attributes, however this is usually not a problem when using a browser to view the document since most browsers ignore attributes that they not recognize. This means that the RDFa attributes does not have any effect on what is displayed to the user.

Below we have a simple example[4] where RDFa is used to markup basic information about a book. Here we are using Dublin Core to add metadata to this document, it is declared at the top so later on in the text we can use the prefix "dc" followed by the Dublin Core data element for the element we are describing (for example title).

```
<div xmlns:dc="http://purl.org/dc/elements/1.1/"
  about="http://www.example.com/books/wikinomics">
  <span property="dc:title">Wikinomics</span>
  <span property="dc:creator">Don Tapscott</span>
  <span property="dc:date">2006-10-01</span>
</div>
```

Essentially what we are trying to do is to bridge the gap between what the computer sees and what a human reader sees. Giving the computer a way to know that headline expresses a title or that the sub-headline indicates the author, hints that a human reader understands while the computer only sees headline and sub-headline.

[4] For more RDFa examples see http://en.wikipedia.org/wiki/RDFa

2.2 Linked Data - Connect data across the Web

The Semantic Web isn't just about putting data on the web. It is about creating connections through links, so that when you have some of it, you can find other, related, data.

The term Linked data describes a method of publishing structured data so that it can be interlinked and become more useful.[8][11] This is done by following a data model which is based on resources and that describe their properties, including relationships to other resources. By using standardized names for resources and property types, data sets from many different providers integrated. By expressing the relationships between resources in different data sets, a user may discover additional relevant data.
The foundation of linked data are the technologies HTTP and URIs. Today we use them (and many other services) to present human readers with web pages, linked data extends this to share information in a way that can be read, connected and queried automatically by computers. This is as a concept is not new, and have been used in other similar situations such as database network models and headings in library catalogs.[9]

Tim Berners-Lee,[5] the father of the World Wide Web coined the term *"Linked Data"* and outlined four principles of linked data:[10]

1. Use URIs to identify things.

2. Use HTTP URIs so that these things can be referred to and looked up ("dereferenced") by people and user agents.

3. Provide useful information about the thing when its URI is dereferenced, using standard formats such as RDF/XML.

4. Include links to other, related URIs in the exposed data to improve discovery of other related information on the Web.

A community project called *Linking Open Data*[6], organized by W3C aims to extend the Web by publishing various open datasets as RDF on the Web by creating RDF links between data items from different data sources. In October 2007, the datasets consisted of over two billion[7] RDF triples, which were interlinked by over two million RDF links. By September 2011 this had grown to 31 billion RDF triples, interlinked by around 504 million RDF links.

The image below shows datasets that have been published in Linked Data format,

[5] `http://en.wikipedia.org/wiki/Tim_Berners-Lee`

[6] WC3 Semantic Web Education and Outreach group's - Linking Open Data community project

[7] `http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`

by contributors to the Linking Open Data community project and other individuals and organizations. The purpose of this image is merely to depict the extent of such a project, not to present the details of the data sets.[8]

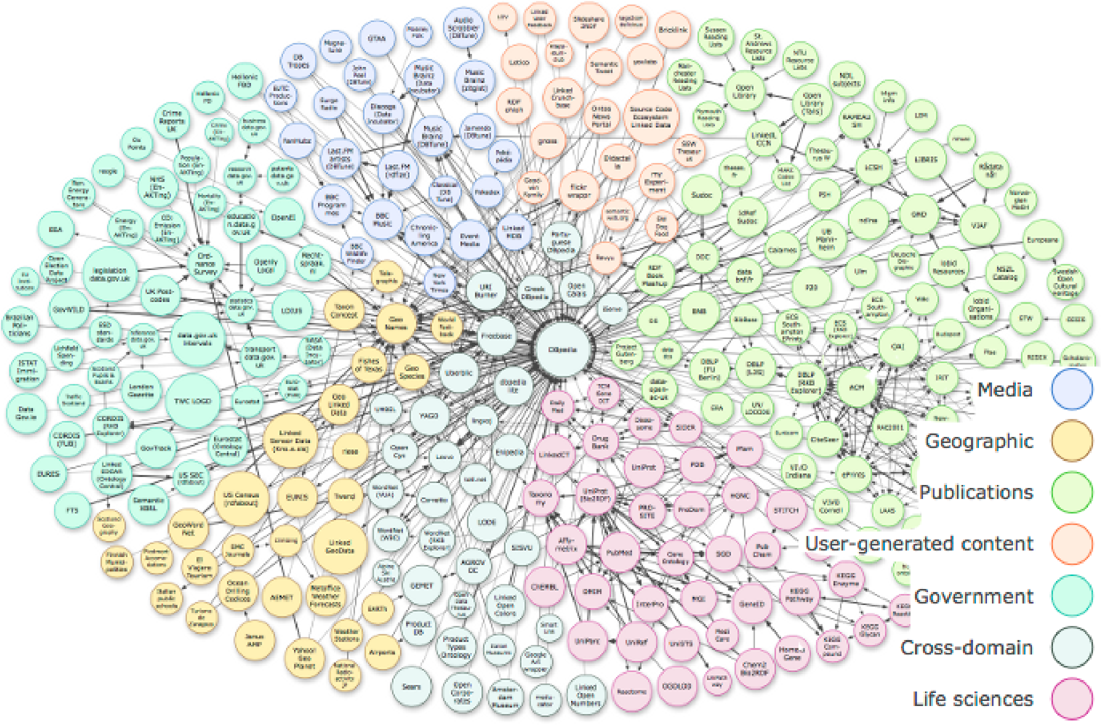

Figure 2.1: Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.

Linked data is essential to actually connect the semantic web, common sense considerations determine when to make a link and when not to.[13]

2.3 Swedish law

The main legislative powers are the parliament (Riksdagen) and the government (Regeringen), these institutions can adopt statutes which are published in the main official collection of statutory law, the *Svensk Författningssamling* (SFS). The statutes enacted by the parliament are referred to as laws, and the statutes enacted by the government as ordinances.[2]

When a statute is changed, this is done by adopting a new statute (the change statute) that states what sections of the old statute (the base statute) are to be

[8] See `http://lod-cloud.net/` for a more detailed image with links to all the contributors.

changed, and how. There is not a new merged version published of the statute in SFS, only the base statute and change statute(s).

Two recurring definitions in this paper are legal sources and legal source documents. A legal source is a type of legal information such as a constitution in SFS or verdicts from one of the Swedish courts. A legal source document is a specific document for a certain legal source, such as *Personuppgiftslagen* (SFS 1998:204).

2.3.1 Law structure

Today there's 3000[9] constitutions but there's no well defined structure that describes how a law should look like. One thing that is common is that each law has an SFS number, including legislations amending already existing laws. The SFS number consists of a four digit year, a colon and a serial number assigned in chronological order of the date of issue.

2.3.2 Rättsinformationssystemet

Rättsinformationssystemet is the state's system to make legal information such as laws, legislative histories, court cases, etc. - available to the public via the Internet. This information is produced by a number of different authorities. Today, each agency is responsible for publishing "their" information via their own website. It is all tied together by the website `http://www.lagrummet.se` which is referring to each agency's respective legal information page.
This decentralized approach has its advantages. It gives the authorities a lot of freedom to design their information in a way that fits them. A downside is that they use this freedom to invent their own standards. *Rättsinformationsförordningen*[10] is not very detailed on how the information should be presented, only that it should be published.

An additional purpose with the system is the ability for the private sector to re-use government information to create added value services[11]. Legal information is of great value, that could be even better utilized if it was easier to re-use the information.

The project's aim is to establish a central place where all legal documents can be searched and retrieved. To achieve this all authorities that are publishing legal documents need to share a common technical standard and use well-defined meta-

[9] Actually there's some 8000 statutes but many of them are change statutes so only around 3000 are active.

[10] Rättsinformationssystemet is regulated in this ordinance (1999:175)

[11] Services like Rättsnätet from Notisum and lagen.nu

data terms.[12] The project imagines the end-product to look something like the figure[12] below.

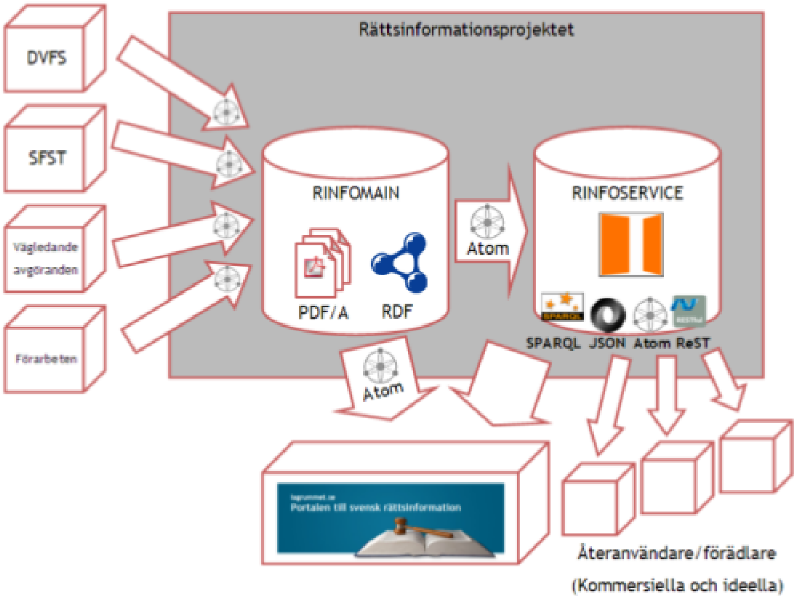

Figure 2.2: Rättsinformationsprojektet - A central place for legal information

As the figure shows legal documents should be retrieved from various government databases, checked to make sure that they are in a correct format (XHTML/RDFa). Once this is done, the data will be stored, documents in PDF format and metadata (RDF files) on disk.

From here (rinfomain) it will be possible to search and retrieve all documents as pdf files, but there will also be a parallel service (rinfoservice). It will have a richer interface where a user can access documents in different formats such as JSON, Atom and HTML in a REST API[13] or even ask SPARQL-queries[14] against the RDF data.

[12] The figure is from rättsinformationprojektets website `http://dev.lagrummet.se/dokumentation/`

[13] A web service implemented using HTTP and the principles of REST.

[14] RDF query language, that is, a query language for databases, able to retrieve and manipulate data stored in Resource Description Framework format.

Part II

Results

Chapter 3

Method

The work with this thesis can be divided into two parts, the first being reading litterature regarding the Semantic Web, Linked Data and related subjects. The litterature also included reading many articels, blog post and watching podcasts,[1] that were great to get a better understaning of the concept and importance of a more semantic web.

The second part was to create a prototyp. Before I started to work on that I looked at the existing system, and tried to get some understanding of what steps the process went through. At this point a I got a lot of guidence from Magnus at Notisum, the author of the existing code. He explained the necessary steps and why they are important for the end result. However I didn't spend to much time on the old code, since I needed to create something different (and take factors such as semantics into mind).
With no background or knowledge in law it can be quite hard to understand the structure of laws and references between them. That is something I picked up as work went along with the prototype, many times I had to stop and try to figure out what I tried to do and why.

3.1 Methodology

I devloped the prototype in several iterations, a simplfied version of agile.[2] The first step was to get the program to read downloaded documents and save them as a new file. After that was in place, I always had a running prototype that would go through all files in a specified folder and apply transformations to them before saving them with a new exstension. This way I could add new functionality and try it out right away. I tried to put general functions and helpers seperate from

[1] References can be found in the bibliography and in footnotes when mentioned throughout the thesis.

[2] `http://en.wikipedia.org/wiki/Agile_software_development`

SFS-specific logic so that it is possible to reuse a lot of the code when other legal sources needs to be implemented.

3.2 Existing system

Below follows a short description of the steps the process goes through today. The steps in blue bubbles (first and last) are written in C# and could with minor modifications be used together with the prototype. The two steps in pink bubbles (second step which is split in two different parts and the third step) are programs written in Pascal and will be replaced by the prototype. Then there's a green bubble (fourth step) which also with some modifications, could be reused with the output from the prototype.

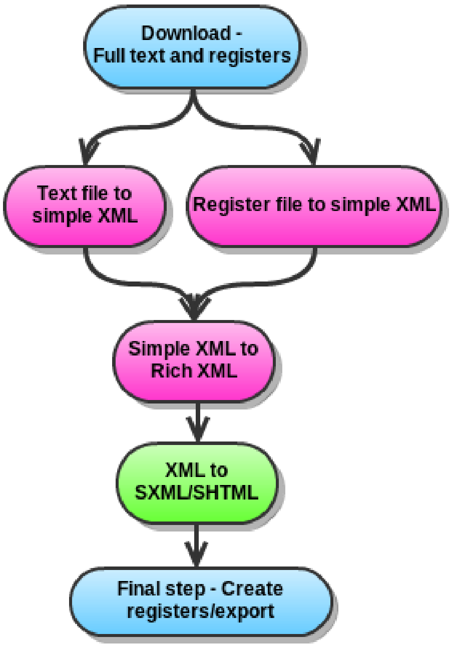

Figure 3.1: Chart describing the existing process

For the sake of simplicity lets say we are only handling SFS laws[3]. First we have a module that handles the actual downloading of the legal source files, (full text and register files for each law) this part is written in C# and could be reused[4] for the prototype. Next we have two programs that for each law converts the downloaded files to very simple XML files, one for the full text and one for the registerfile. The next step is also an XML transformation, it takes the simple XML files and turns them into more comprehensive XML files. Now it's time to create files that can be viewed in a browser, this is done in several steps where data

[3] In reality the program handles all kind of legal source documents.

[4] This part just have to download the files, then the prototype can do it's magic on those files.

structures containing information regarding the laws are created with information like paragraph references and links. In the end a SHTML[5] file that is ready to be published with this information is created. The final step is to export these files (and other legal source documents) to databases and update the website to display the newly created or updated files.

3.2.1 Output

The resulting file will look something like the code example below, that is describing paragraph 4, section 1 from the *Home Guard regulation* (1997:146).[6]

```
<a name="4"></a>
<p><a name="P4"></a>
<a name="P4S1"></a><b>4</b>
Hemvarnets personal bestar av hemvarnsman och personal
med avtal for tjanstgoring i hemvarnet enligt forordningen...
</p>
```

As we can see the only markup that is done to the document is regular HTML markup that can be used to link within the document for example. In the Result section we will look at examples of how this statute will look when it is marked up with RDF.

3.3 Prototype

The main differences between the old system and the prototype will be:

- Language, the prototype is written in Python[7] which is a object oriented[8] languge that is well suited for tasks like this one. It is easy to divide the code into separate modules that handle different tasks, create relationships that allows objects to inherit from other objects. Furthermore it is convinient to work with (apply transformations etc.) objects such as whole documents, paragraphs etc.

- Follows standards proposed by Rättsinformationsprojektet for legal documents.

- The generated files will be marked up with RDF to contribute to a more semantic web.

[5] http://www.computerhope.com/jargon/s/shtml.htm

[6] This statute will return as an example later on in the thesis

[7] www.python.org

[8] http://www.webopedia.com/TERM/O/object_oriented_programming_OOP.html

3.3.1 Delimitations

The prototype does not have to handle the downloading of the legal source documents, these can be presumed exist on the computer running the prototype.
The aim for the prototype is to deliver XML files, the final step where these files are transformed into HTML files is simple and needs to be controlled by Notisum. However this step is done by the prototype but might not be the way Notisum will do it, if they use the prototype.

Chapter 4

Implementation

In this chapter I will discuss how the implementation was done and describe some implementation ideas and some parts of the prototype.

4.1 Interface

In order to interact with the program, read status and error information some sort of user interface is needed. Most common is a graphic user interface[1] that for example uses icons and menus. However, this falls outside the scope of this project, which focuses on the core of the program rather than the user experience. Instead, a rudimentary command line interface[2] was implemented, where the user can specify arguments at the start of the program (which is done via a terminal). During the parse and transformation process status updates will be presented in the terminal, for example if something fails the user will be notified. Also when the program is done the user will receive information about what was processed and the outcome.

The main file is called Controller.py and to run the program from a terminal an argument specifying what the program should do is needed. The intresting arguments for this thesis are 'ParseAll' and 'GenerateAll' other possible arguments are 'DownloadAll' and possible candidates to implement would be only 'Parse' or 'Generate' followed by a specific source, for example 'Parse 1997:240'. Also one can specify what types of legal sources that should be run, (will be relevant when more sources than SFS are implemented.) default behavior is to run for all types. There's also two available flags to use, '-d' for debug mode and '-h' or '–help' for help instructions.

[1] A graphic user interface is a human-computer interface i.e., a way for humans to interact with computers.

[2] Command line interface (CLI), an interface which use only text and are accessed solely by a keyboard. For example via an terminal in Linux.

```
kungen@dell-desktop:~$ python Controller.py --help
Usage: pyhton Controller.py [-d | -h] [arg]
Available flags are: -d (debug), -h--help (help)
kungen@dell-desktop:~$ python Controller.py
No arguments given
Valid arguments are: DownloadAll, GenerateAll, ParseAll
kungen@dell-desktop:~$ python Controller.py ParseAll
```

4.2 Main modules

The main modules consists of the 'Controller' and 'Source' that includes the logic to run the program. Then there's the different legal type modules that contains their specific logic.

4.2.1 Controller.py

The controller handles validation of user input (arguments and flags) if everything is correct it fetches all available legal source types (in this case just SFS) and then run the command given as an argument, ex. 'ParseAll' for each type.

4.2.2 Source.py

This module is a blueprint of how modules inheriting from it needs to look like. It contains the base classes for Controller and Parser, these classes contains functions that child classes need to implement and functions that can be overridden. Example of general functions are: checking if a file is up-to-date, trimming filenames or returning the XML or XHTML name for a certain file.
However most of the code will be specific for each legal source type and be implemented in the child modules. To support a new type of legal source for example 'EG Court cases'[3], create an 'EG' module that inherits Source.Controller and Source.Parser, that is all that is needed for it to function with the rest of the program.

4.2.3 SFS.py

This is the module specific for SFS documents, basically it takes a list of all the downloaded files[4] and parse the files one by one.
The largest and most complex class is 'SFSParser' that parses a document and creates objects for each part of the document. All objects are defined as classes[5] here, for example 'Rubrik', 'Stycke', 'Paragraf' etc. One of the parser's duties is

[3] The Court of Justice of the European Union `http://curia.europa.eu/jcms/jcms/j_6/`

[4] The path to the base directory is specified in a config file, then it appends a path like '/sfs/dl/sfsr/1997' for example for all SFS full text documents from 1997

[5] These classes inherit from base classes in DataObjects.py.

to figure out what type of object a certain part of the text should be represented as. This is achieved by first guessing what it should be based on for example the previous section, then running functions to verify or reject that type.

4.3 Helper modules

The helper modules are used to simplify (mostly) for the Parser by defining native python types, providing different types of text readers etc.

4.3.1 DataObjects.py

Makes it possible to build an object model for each legal source document, by creating simple data objects. These objects are base data types that inherit from native python types such as unicode, list, dictionary. The data types have added support for other properties that can be set when instantiated.

4.3.2 Reference.py

Parses plaintext and finds references to other legal source documents and returns a list with "Link-objects"[6]. Depending on with which properties it is initialized, it can find different types of references (Other laws, 'Rättsfall', 'EG-lagstiftning' etc..).

4.3.3 TextReader.py

A helper class to read text files in different ways, by line, paragraph etc.[15]

4.3.4 Util.py

A few small help functions mostly related to checking and or renaming directories and files.

4.4 Unicode

Unicode is a standard for consistent encoding, representation and handling of text. It can be implemented by different character encodings[17][7].
Python's Unicode string type stores characters from the Unicode character set. Unicode characters don't have an encoding; each character is represented by a number which is called its code point. However we work with text files, which contains encoded text, not characters. Each character in the text is encoded as one or more bytes in the file. If you read a line of text from a file, you get bytes, not characters.

[6] A Link-object is a base data type that inherits from 'Unicode Structure' in DataObjects.py, basically a unicode string with a 'URI' property.

[7] Example of common character encodings are, UTF-8 and ISO-8859-1, see the vocabulary in chapter 1 for more info.

To solve this, we need to decode the text strings that we read from files, to do this, we can use Python's **decode()** method on the string, and pass it the name of the encoding like so:

```
encoding = "iso-8859-1"

raw = file.readline()
txt = raw.decode(encoding)
```

When we need to save a Unicode string to a file, we have to do the opposite conversion and encode it. The **encode()** method converts from Unicode to an encoded string.

```
out = txt.encode("utf-8")
```

There is a lot of files and text to handle when parsing laws, to avoid issue with different encodings and representations of characters we try to only work with Unicode in the code. Also some of the third party libraries only work with Unicode which is another reason to stick with that. The content is converted[8] to Unicode when it is loaded into the program. A Unicode string in Python looks like this *u'string'* instead of a just *'string'* and that is the format that is used to represent strings when needed.
The source files we work with are saved as ISO-8859-1 and the generated files we create will be saved in UTF-8.

4.5 3rd party libraries

There's a few third party libraries that are used in the program to perform certain tasks.

4.5.1 BeautifulSoup

BeautifulSoup is a really helpful library when you need to parse HTML documents (even with malformed markup, i.e. non closed tags). It creates a DOM[9] for parsed pages that can be used to extract data, for example you can ask it to extract "The first row in the second table on the page". This comes very handy and have been to a great assistance during my work.

[8] The conversion is done by the Util module.

[9] Document Model Object (DOM) is a convention for representing and interacting with objects in HTML, XHTML and XML documents.

4.5.2 SimpleParse

SimpleParse allows you to generate parsers directly from EBNF[10] grammar. This library has also been very helpful during my work since it allowed me to specify rules (for different types of documents, references, headlines etc.) and then create a parser with those capabilities.

4.5.3 RDFLib

RDFLib is a library for working with rdf that contains an RDF/XML parser and serializer. It also contains graph data structures to represent data. In my case it was for example used to read and represent data from an external file with URIs to all the SFS laws.

4.5.4 Genshi

Genshi is a library that provides components for parsing, generating, and processing HTML, XML or other textual content for output generation on the web. The main feature that is used in this project is a template language. It is used to convert the parsed and marked up files to XHTML files. A nice feature is that template instances are cached to avoid having to parse the same template file more than once.

4.6 Version control

GitHub[11] is used for web hosting and revision control, the code used for this report is open source and available to checkout from the link below. On the github page there's also an issue tracker with the current open issues, bugs and some enhancements.

Project: `https://github.com/Sup3rgnu/lawParse`
Checkout[12]: `https://github.com/Sup3rgnu/lawParse.git`

[10] Extended Backus-Naur Form is a formal way to describe the grammar of languages. It consists of rules which are restrictions governing how symbols can be comined.

[11] GitHub is a web-based hosting service for software development projects that use the Git revision control system. `http://github.com`

[12] For more info regarding how to checkout, see `http://www.kernel.org/pub/software/scm/git/docs/git-checkout.html`

Chapter 5

Analysis

This section will look at the different steps in the process flow, what kind of input we expect, how the parsing is done and finally what kind of output we get.

5.1 Process flow

Let's run the program and follow one SFS statute through the process flow to see what steps it goes through and how it changes. The statute chosen[1] for this excerise is the *Home guard regulation*[2] (1997:146). A transcript of the log created (when the 'debug' flag is turned on) during the parsing of this file can be found in the appendix, if the reader wants to follow the process steps.

5.1.1 Input files

The SFS statutes published in the Government Offices legal databases consists of two files for each statute, one *Statute register file* (SFSR) and one *Statute in full text* (SFST). The program expects the downloaded files to be saved in a specific tree structure to be able to read, create and save files.

[1] The statute was chosen randomly, it does not have any specific properties that makes it more suitable as an example.

[2] Hemvärnsförordningen `http://notisum.se/rnp/sls/sfs/20050819.pdf`

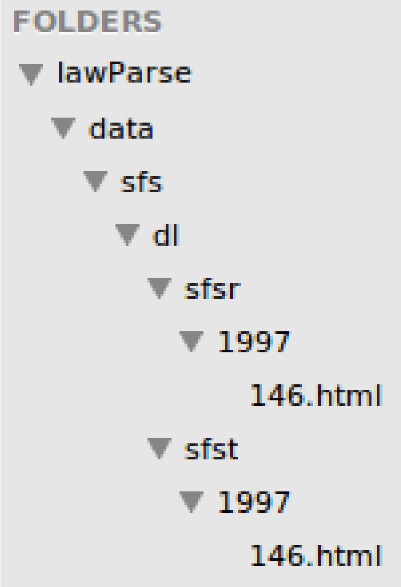

Figure 5.1: Tree structure for downloaded files

Under the root ’lawParse’ there is a directory called ’data’ which contains all different types of legal documents, here we have a ’sfs’ directory where we store the downloaded, (’dl’) parsed and eventually the generated files respectively. In all these directories we have another level for ’sfsr’ and ’sfst’ files, here the statutes are sorted according to the year they were created.

5.1.2 Parsing

The first step is to scan the directory containing the files, this will give us a list of files, we will loop through the list one statute at the time. For our exampel (1997:146) statute we we will have the following set of files:

```
All files connected to this file:
        {’sfst’: [u’../lawParse/data/sfs/dl/sfst/1997/146.html’],
         ’sfsr’: [u’../lawParse/data/sfs/dl/sfsr/1997/146.html’]}
```

We then transform and trim the file names so that they will be on the format ’1997/146’ instead of ’../1997/146.html’. The actual parsing step then begins if the file met these three requirements:

1. If we have the file parsed already and it is newer than the incoming raw file, don’t parse it again.

2. Filter out documents that are not proper SFS documents. They will have a name like ‘N1992:31’[3]

[3] It happens that other agencies publish documents to SFS, this is handled by giving them a SFS number preceded with an ‘N’

3. Skip parsing the documents that have been or will be revoked, they are marked as *"The constitution is repealed / shall be repealed"*[4]

The parsing begins with creating a SFS specific Parser, that has a set of properties and rules defined such as regular experssions created for parsing SFS statutes. Like this regular expression for matching a SFS number:

```
reSimpleSfsId = re.compile(r’(\d{4}:\d+)\s*$’)
```

The raw string (’r’) notation keeps the regular expression clean, without it every backslash would have to be prefixed with another on to escape it. Then we are looking for four (4) digits followed by a colon(:) and one (1) or more digits. The ’s’ at the end is there to match white spaces (behaves differently with or without the unicode flag turned on). Finally the dollar sign ($) states that the match have to be the end of the string. The other regular expressions are created in a similar fashion, in total there are about 30 expression matching everything from chapter ids and numbered lists to the definition of a revoked date.[16]

When the SFSParser class is initialized and have all it’s properties created, we can divide the parsing into two steps. One where we handle the register file (SFSR) and one for the full text (SFST). We begin with creating a registry with *registerposts* containing meta information and other information regarding changes to the statute.

```
Registerpost:
{u’Ansvarig myndighet’: Link(’u’Forsvarsdepartementet’’,
        uri=u’http://lagen.nu/org/2008/forsvarsdepartementet’),
 u’Ikraft’: DateSubject(1997, 7, 1),
 u’SFS-nummer’: u’1997:146’}

Registerpost:
{u’Omfattning’: [
        u’andr. ’,
        Link(’u’1 \xa7’’,
        uri=u’http://rinfo.lagrummet.se/publ/sfs/1997:146#P1’)],
 u’Ikraft’: DateSubject(2012, 7, 16),
 u’Rubrik’: u’Forordning (2012:334) om andring i
        hemvarnsforordningen (1997:146)’,
 u’SFS-nummer’: u’2012:334’}
```

The first registerpost created for 1997:146 describes basic properties such as responsible authority, entry into force and SFS number. The second[5] registerpost states that there has been some changes to the statute. We can find out if it is a

[4] "Författningen är upphävd/skall upphävas"

[5] There’s actually three register posts for 1997:146, I left a ’change post’ out to save space

change or if some part has been revoked[6], also we can see the force into entry and the change statute's SFS number. All these values are represented with different data types discussed in the 'Implementation' section, for ex. 'Link', 'DateSubject' or a unicode 'u' string.

The second part is to deal with the full text part of the statute, we do this by creating an intermediate text file with just the raw[7] text. This file is saved in a directory called 'intermediate' on the same level as the downloaded, 'dl' directory in the file tree structure.

The file we have downloaded consist of a header with information similar to the register file. We could probably reuse some information[8] but we have everything set up for parsing and it is a fairly simple procedure so we go ahead and save it again. This information will be part of the header in the HTML file when that is created later on. For the information in the header we create a corresponding object, in the example below we create *UnicodeSubjects*[9], with the values *SFS number* and *Headline*, their predicates will be an URI to that object's definition.

```
if key == u'Rubrik':
        meta[key] = UnicodeSubject(val, predicate=self.labels[key])
elif key == u'SFS nr':
        meta[key] = UnicodeSubject(val, predicate=self.labels[key])
```

Here is the values for the variables above when the header infromation is created:

```
key:SFS nr
val:1997:146
self.labels[key]:
        http://rinfo.lagrummet.se/taxo/2007/09/rinfo/pub#fsNummer
key:Rubrik
val:Hemvarnsforordning (1997:146)
self.labels[key]:
        http://purl.org/dc/terms/title
```

To mark up the full text we call a number of methods on the raw text file, to first of, find out what the part we are looking at should be represented as and then to create an object with that type's properties. The main method is **makeForfattning** when it is called it will in turn call several other methods like **makeKapitel,**

[6] Omfattning can be for example "Change" or "Revoked" ("Ändr." or "Upph.")

[7] It is not the complete downloaded file, we get rid of the "HTML" mark up of the document so that we only have text.

[8] SFS number and responsible authority could be reused for example.

[9] We will look closer at this in the next section 'RDF markup'

makeParagraf and **makeTabell** and so on. The logic to figure out what each part should be marked up as uses a 'statehandler' to keep track of what part we just parsed and to guess what should come next. If we are inside the **makeStycke** method possible states/methods we can invoke are different types of lists and tables. Then when we reach the end of the current state we start over again and look what state the next part should be, this part can probably not be a list or a table, those states can only be reached from within certain states. Following that logic for example a paragraph can not be called within a chapter, each type has its place in the hierarchy.

All objects have an **"isObject"** and **"makeObject"** method, and they are for example used like the example below shows.

```
def guesState(self):
        try:
                if self.reader.peekLine() == '':
                        handler = self.blankLine
                elif self.isAvdelning():
                        handler = self.makeAvdelning
                elif self.isKapitel():
                        handler = self.makeKapitel
                elif self.isParagraf():
                        handler = self.makeParagrafnd{minted}
                ...
```

When we are done with the parsing the text body we add some additional information to the law's meta information such as time created and preliminary work. To find out if there is any preliminary work we check the *registerposts* in the register we have created, and save the URI to that document if existing.

5.1.3 RDF markup

During all the different parsing steps, I have mentioned that we create data objects with specialized properties for the different parts of the statute. Let's take a closer look at one of the objects that are created during our parse example.
In the previous section we look at example code for finding a statute's SFS number and headline, they are both of the type *UnicodeSubject*. The class *UnicodeSubject* it self does not do anything as we can see in the class definition below.

```
class UnicodeSubject(PredicateType, UnicodeStructure):
        pass
```

However the class inherits from two other classes, *PredicateType* and *UnicodeStruc-*

ture. We are going to look at what our *PredicateType* objects looks like. In the second chapter we stated that RDF is based upon the idea of making statements about resources in the form of triples of subject-predicate-object. Here we are obviously looking at the ’Predicate’ part, as the class definition below reads; Inheriting from this class gives the child class a predicate attribute that describes the RDF predicate to which the class is the RDF subject.

```
class PredicateType(object):
    """Inheriting from this class gives the child class a
    predicate attribute that describes the RDF predicate to
    which the class is the RDF subject"""
    def __init__(self, *args, **kwargs):
        if ’predicate’ in kwargs:
            self.predicate = kwargs[’predicate’]
            shorten = False
            for (prefix, ns) in Util.ns.items():
                if kwargs[’predicate’].startswith(ns):
                    predicateUri = kwargs[’predicate’]
                    kwargs[’predicate’] = kwargs[’predicate’]
                        .replace(ns, prefix + ’:’)
                    shorten = True
                else:
                    from rdflib import RDFS
                    self.predicate = RDFS.Resource
            super(PredicateType, self).__init__(*args, **kwargs)
```

When the object is initialized we compare the predicate’s URI to some common name spaces that we have stored, if that is the case we can swap the long URI for a much shorter prefix. In our case with the SFS number and the title we can substitute the title’s URI ’`http://purl.org/dc/terms/title`’ for simply ’dct:title’ and ’`http://rinfo.lagrummet.se/taxo/2007/09/rinfo/pub#fsNummer`’ becomes ’rinfo:fsNummer’. Below we find part of the name space list, as we can see we can reuse definitions that are already defined by large international organisations, it is only ’rinfo’ that is specific "Swedish-legal" definition.

```
# Common namespaces and prefixes for them
ns = {
    'dc':'http://purl.org/dc/elements/1.1/',
    'dct':'http://purl.org/dc/terms/',
    'rdfs':'http://www.w3.org/2000/01/rdf-schema#',
    'rdf':'http://www.w3.org/1999/02/22-rdf-syntax-ns#',
    'skos':'http://www.w3.org/2008/05/skos#',
    'rinfo':'http://rinfo.lagrummet.se/taxo/2007/09/rinfo/pub#',
    'xht2':'http://www.w3.org/2002/06/xhtml2/'
}
```

This way we can mark up our legal documents with short prefixes that makes the documents readable for both humans and computers in a way that contributes to the Semantic Web. In a final HTML file our title would look something like the snippet below, much more meaningful then just a header tag with a title, yet still simple enough to be understandable for human readers.

```
<h1 property="dct:title">Hemvarnsforordning (1997:146)</h1>
```

5.1.4 Generating XHTML

The way we are going to represent the parsed and marked up statute is as a XHTML file (in a later step we can transform the XHTML files to HTML to be able to display them to users in a browser). One of the reasons that the XHTML format is used is that through its extensibility and its focus on document structure it works well to represent legal documents in a semantically meaningful way.

To create an XHTML representation of the parsed statute we use a third party library called *Genshi*[10]. We use Genshi's "TemplateLoader" to load a template we have created that specifies how we want our XHTML file to look like, ex. which values goes where etc. A nice feature with Genshi is that it uses a Stream-based filtering[11] that allows us to apply various transformations as a template is being processed, without having to parse and serialize the output again.

Below is an example of how the template file renders a headline. As the code snippet shows there is two <h> tag templates to chose from, one if it is a "normal" headline and one if it is an sub headline, then an extra class is added to the tag.

[10] See the 'Implementation' section for more information regarding third party libs.

[11] http://genshi.edgewall.org/wiki/Documentation/filters.html

```
<div py:def="render_rubrik(rubrik)" py:strip="" py:choose="">
    <h py:when="rubrik.type == ’underrubrik’" py:content="rubrik"
        id="${rubrik.id}" class="underrubrik">Underrubrik</h>
    <h py:otherwise="" py:content="rubrik"
        id="${rubrik.id}">Huvudrubrik</h>
</div>
```

5.2 Output

The output we get in form of a XHTML document consists of two parts a <head> and a <body> as normal HTML document. We are going to look at those parts[12] to see what we ended up with after parsing and marking up our statute.

All documents will have several namespaces, they will be declared in the beginning of the document in the <html> element. These namespaces are mainly[13] used to express the predicate in the RDF triples that will be embedded in the document.

```
<?xml version="1.0" encoding="utf-8"?>
<html xmlns="http://www.w3.org/2002/06/xhtml2/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:dct="http://purl.org/dc/terms/"
    xmlns:rinfo="http://rinfo.lagrummet.se/taxo/2007/09/rinfo/pub#"
    xmlns:rinfot="http://example.com/terms#"
    xsi:schemaLocation="http://www.w3.org/2002/06/xhtml2/
            http://www.w3.org/MarkUp/SCHEMA/xhtml2.xsd"
    xml:base="http://rinfo.lagrummet.se/publ/sfs/1997:146"
    xml:lang="sv"
    about="http://rinfo.lagrummet.se/publ/sfs/1997:146">
```

5.2.1 Document head - metadata

The first part of the document is inside the <head> element of the document, it contains metadata regarding the document. We find the obvious information such

[12] I will only show parts of the resulting XHTML document here to save space and to focus on some of the interesting parts. The full document can be found in the appendix.

[13] The XHTML2 namespace is the default namespace and is used for elements and attributes.

as title and a URI to rättsinformationsprojektet definition of the statute. Then further down there is some elements that are marked up with the RDFa[14] attributes. There are some that use the attributes “property” and “content” describing things such as SFS number and the date when the document was fetched. Lastly we find elements marked up with the “rel” (relationship) RDFa attribute, these are links to inter alia the change statutes we found for this statute.

```
<head about="http://rinfo.lagrummet.se/publ/sfs/1997:146">
    <title>Hemvarnsforordning (1997:146)</title>
    <base href="http://rinfo.lagrummet.se/publ/sfs/1997:146"/>
    <link rel="rinfo:forfattningsamling"
        href="http://rinfo.lagrummet.se/ref/sfs"/>
    <meta property="rinfo:fsNummer" content="1997:146"/>
    <meta property="rinfoex:senastHamtad" content="2012-08-01"
        datatype="xsd:date"/>
    <meta property="rinfo:utfardandedatum" content="1997-04-10"
        datatype="xsd:date"/>
    <link rel="rinfo:konsoliderar"
        href="http://rinfo.lagrummet.se/publ/sfs/1997:146"/>
    <link rel="rinfo:konsolideringsunderlag"
        href="http://rinfo.lagrummet.se/publ/sfs/1997:146"/>
    <link rel="rinfo:konsolideringsunderlag"
        href="http://rinfo.lagrummet.se/publ/sfs/2005:819"/>
    <link rel="rinfo:konsolideringsunderlag"
        href="http://rinfo.lagrummet.se/publ/sfs/2012:334"/>
</head>
```

5.2.2 Document body - Full text

The second part of the document is the body, which consists of four parts itself, with the statute’s title being the first.

```
<h property="dct:title">Hemvarnsforordningen (1997:146)</h>
```

It is a <h> tag (header) that is marked up with the RDFa attribute "property" which is set to "dct:title", meaning that this part is a title and the definition for title can be found using the namespace that "dct" is defined as in the beginning of the document[15].

[14] See chapter 2, section 2 regarding RDFa for more information about RDFa attributes

[15] The prefix dct represents the Extended Dublin Core namespace

In XHTML it is possible to assign a role to an element, for example “contentinfo” or “main”[16]. This feature is used to declare that a certain section is the document’s actual text (main) etc. The next part of the body is marked with the role “contentinfo”. Which is including metadata relating to the document as a whole, and in the typical use case that should be visible to the user and not hidden in the document’s head. It can be thought of as a short intro with some short facts that the user might be interested in seeing right away when looking at the statute.

```
<dl role="contentinfo">
...
<dt>SFS nr</dt>
    <dd property="rinfo:fsNummer">1997:146</dd>
<dt>Utgivare</dt>
    <dd rel="dct:publisher"
        href="http://example.com/org/2008/regeringskansliet">
        Regeringskansliet</dd>
...
```

As we can see from the figure above, the elements, SFS number and publisher are marked up with RDFa attributes.

The next section of the body is where it starting to get a little bit more interesting in a semantic point of view. It is the part with the documents actual text, it has been assigned the role “main”. If we look at the two first paragraphs below[17], we see that almost everything is marked up with RDFa attributes.

[16] TODO: src

[17] The actual text includes åäö and "§" characters, but the latex plugin (minted) used to generate code snippets can not handle those characters, so I have manually removed them from this figure.

```
<section role="main">
<h id="R1">Hemvarnets andamal</h>
<section id="P1" typeof="rinfo:Paragraf" about="#P1"
    property="rinfot:paragrafnummer" content="1">
    <span rel="dct:isPartOf"
        href="http://rinfo.lagrummet.se/publ/sfs/1997:146"/>

    <p id="P1S1" about="#P1S1" typeof="rinfo:Stycke">
    <span rel="dct:isPartOf" href="#P1"/>
    <span class="paragrafbeteckning">1</span>
    Hemvarnet har till uppgift att skydda skyddsobjekt. Hemvarnet
    ska i ovrigt kunna stodja de operativa insatsforbanden och
    delta i annan verksamhet dar Forsvarsmakten medverkar.
    </p>

    <p id="P1S2" about="#P1S2" typeof="rinfo:Stycke">
    <span rel="dct:isPartOf" href="#P1"/>
    Hemvarnet ingar i Forsvarsmakten.<a rel="dct:references"
    href="http://rinfo.lagrummet.se/publ/sfs/1997:146#L2012:334">
    Forordning (2012:334).</a>
    </p>
</section>
...
</section>
```

First we have a header with the id “R1” which stands for header number one[18]. Then the first paragraph starts and that includes many RDFa attributes such as the “typeof” attribute which states that the following text is a paragraph and the definition for a paragraph can be found at the URI represented by “rinfo” followed by “Paragraf”. We can also see that this section has an id “P1”, which of course is short for “Paragraph one”. There is also a relation (the “rel” attribute), it is of the type “PartOf” which is a Dublin Core element, it is stating that this section is a part of the statute 1997:146. After that comes a similar code snippet but this time it is a “Stycke"[19] that is a part of “P1”, the first paragraph that is. It has the id “P1S1”, which stands for Paragraph one, Section one, this id notation continues by increasing the number for each element and concatenating sections as we traverse into the document tree.
The first paragraphs second section (P1S2) looks similar, it is marked up with a relation attribute with a reference to one of the change statutes (2012:819).

The last part of the document body is a section with the role “secondary”, as

[18] Rubrik in Swedish

[19] Stycke is a subsection or "subparagraph"

the name suggests it includes material that is not a part of the document text but is associated with it, typically a list of changes and transitions.

```
<section role="secondary">
<h1>Andringar och overgangsbestammelser</h1>
...
<section id="L2012:334"
    about="http://rinfo.lagrummet.se/publ/sfs/2012:334">
<dl>
<dt>Omfattning</dt>
    <dd>andr. <a rel="rinfo:ersatter"
    href="http://rinfo.lagrummet.se/publ/sfs/1997:146#P1">1</a></dd>
<dt>Ikraft</dt>
    <dd property="rinfo:Ikrafttradandesdatum">2012-07-16</dd>
<dt>Rubrik</dt>
    <dd property="dct:title">Forordning (2012:334) om andring i
    hemvarnsforordningen (1997:146)</dd>
<dt>SFS-nummer</dt>
    <dd property="rinfo:fsNummer">2012:334</dd>
</dl>
</section>

    </section>
  </body>
</html>
```

We can see that there is a section with the id “L2012:334” and a URI to that change statute. Then there is information regarding the change, what has been change, when is the entry into force date etc. All this information is of course marked up with RDFa attributes and elements from both the dct and rinfo namespaces.

5.2.3 RDF

The vocabularies that are used are mainly, Dublin Core (dct) and Rättsinformationsprojekt’s own vocabulary (rinfo). The Dublin Core vocabulary that is used is an extended version called “DCMI Terms”, some of the elements that are used from that are:

- **dct:title** - The document’s title, for ex. "Hemvärnsförordningen" (1997:146)
- **dct:publisher** - The publisher, for ex. “Regeringskansliet”
- **dct:alternate** - A shorter prefix, ex “TF” for “Tryckfrihetsförordningen”

- **dct:reference** - Used for references to another document.

The other vocabulary, *rinfo* is provided by Rättsinformationsprojektet[20]. It includes definitions such as "Ikrafttradandesdatum", "ersatter", "paragrafnummer" and other attributes that are specific to legal documents. A conceptual model that describes the document types, properties and other related matters that are Swedish Legal Information can be found in Rättsinformationsprojektet's documentation[21]

RDF Triples

If we extract RDF data from the output document, which is possible with for example RDFa Distiller[22] we get all the RDF Triples for the document. Here are a couple of the resulting triples from the examples we have looked at earlier, displayed in N3-format.[14]

```
<http://rinfo.lagrummet.se/publ/sfs/1997:146#P1S1>
<http://purl.org/dc/terms/isPartOf>
<http://rinfo.lagrummet.se/publ/sfs/1997:146#P1>
...
<http://rinfo.lagrummet.se/publ/sfs/1997:146#P1S2>
<http://purl.org/dc/terms/references>
<http://rinfo.lagrummet.se/publ/sfs/1997:146#L2012:334>
```

The first triple is stating that “P1S1 is a part of P1” that is, "Paragraph 1, Section 2" is a part of "Paragraph 1" which makes sense to a human reader but now it is also understandable to a computer. The second triple states that “P1S2 is a reference to L2012:334”, in this case, Paragraph 1, section 2 has been revoked and replaced with a new statute, 2012:334. With all these triples telling us about all sorts of things in our document, it also possible for computers to understand the meaning of the different parts, how they are connected and where to find more information about them.

[20] More information regarding Rättsinformationsprojektet's URI principles see http://dev.lagrummet.se/dokumentation/system/uri-principer.pdf

[21] Documentation in pdf format, http://dev.lagrummet.se/dokumentation/model.pdf

[22] RDFa Distiller http://www.w3.org/2007/08/pyRdfa/

5.3 Performance

Even though performance is not the main focus of this thesis, it is still a part of it. As I stated in the problem statement, we are handling a lot of documents hence the program requires a fast process. The process has been tested on two computers[23] with quite different specifications, one pretty slow machine and one well above average (for a laptop). The result shows that the performance increases significantly[24] with a more powerful machine, which is positive since then it is not the case that the program itself is a limitation on how fast the process can run.

Computer specifications for benchmark:

Make	Dell	Apple
Model	Vostro V13	MacBook Pro
Processor	Intel Celeron 743 @ 1.30 GHz	Intel Core i7 @ 2,6 GHz
Memory	1.9 GB	16 GB
OS	Ubuntu 12.04 32-bit	Mac OS X 10.8

To ensure that the performance is not disturbed by any other processes it has been run at night with all other programs shut down.

5.3.1 Total running time

As we can see from these numbers collected when running all SFS statutes[25] and what becomes even more clear in the charts below, most of the statutes are parsed and marked up very fast, the average running time on the mac is just above one second. There are a few statutes that are very slow that makes up most of the total running time.

Computer	**Dell Vostro**	**MacBook Pro**
Nr of statutes	3424	3424
Average running time	2.53s	1.06s
Max running time	113.79s	47.74s
Min running time	0.97s	0.40s
Total running time	8700.59s / 2hrs 25min	3721.26s / 1hrs 2 min

[23] Note that I am not saying that the Mac is better than the Dell, I just happend to have a 2 years old Dell and a brand new Mac (also, the price differs with a about a factor of ten, so it should be faster no matter the brand.)

[24] Almost all statutes parse in half the time on the Mac compared to the Dell

[25] All the statutes that should be marked up, that is statutes that are valid i.e. not revoked. The total number of documents that pass through the process are about twice as many as the ones that get marked up, this time is negligible, but of course included in the total running time.

5.3.2 Running time for selected statutes

Because of the number of statutes I will only show a graph depicting the statutes from a few years (1942, 1967, 2005 and 2006)[26] to get a meaningful[27] graph. The purpose of the graph is to visualize that most statutes parse fast and few 'heavy' statutes comprise the majority of the time.

Computer	**Dell Vostro**	**MacBook Pro**
Nr of statutes	270	270
Average running time	3.15s	1.24s
Max running time	71.98s	28.71s
Min running time	1.04s	0.41s
Total running time	860.59s	337.79s

Below are the charts showing the statutes on the horizontal axis and the time in seconds on the vertical axis. The longer running times (blue) are from when the statutes were run on the Dell and the shorter ones (pink) on the Mac.

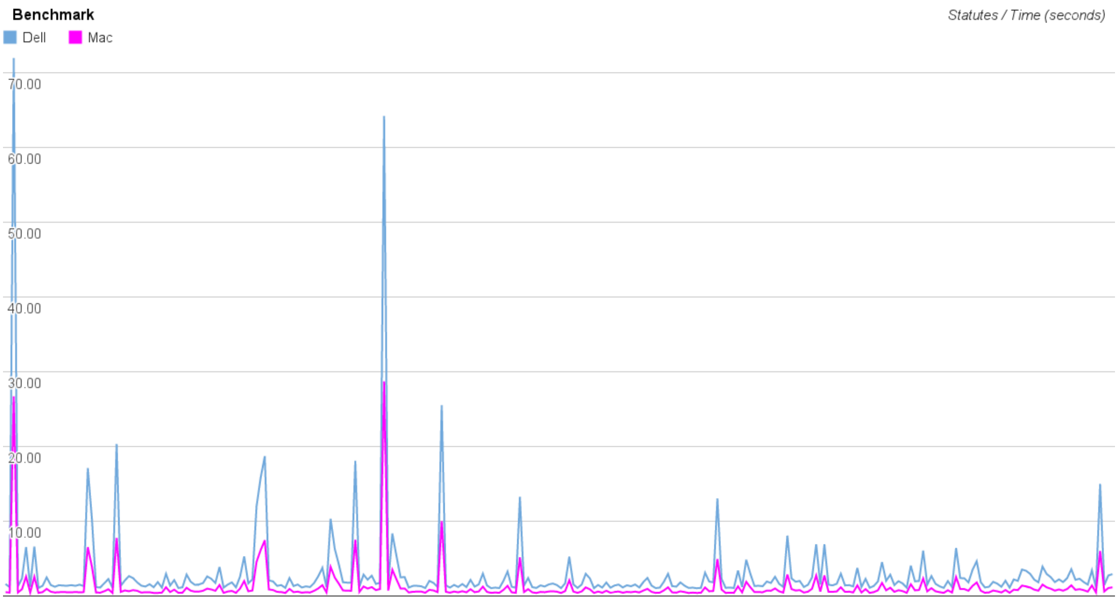

Figure 5.2: Chart showing running times for the selected statutes on both test computers

[26] These years are picked randomly, a couple more recent and a couple older years to get a good variation

[27] Now the graph is showing 270 statutes, it would be impossible to include all 3400 in a graph of this size.

The statute that took the longest time[28] (71.98s and 28.71s) is *Rättegångsbalken* (1942:740) which consist of 59 chapters and more than 100 change statutes, hence it takes a lot of time to go through and mark up all that text.

In this next chart we have a zoomed in version of the chart with running times on the Mac for the selected statutes, here it becomes even more clear that most statuts parse very fast. As we can see that most statute's running time is less than two seconds (average running time is 1.24 seconds).

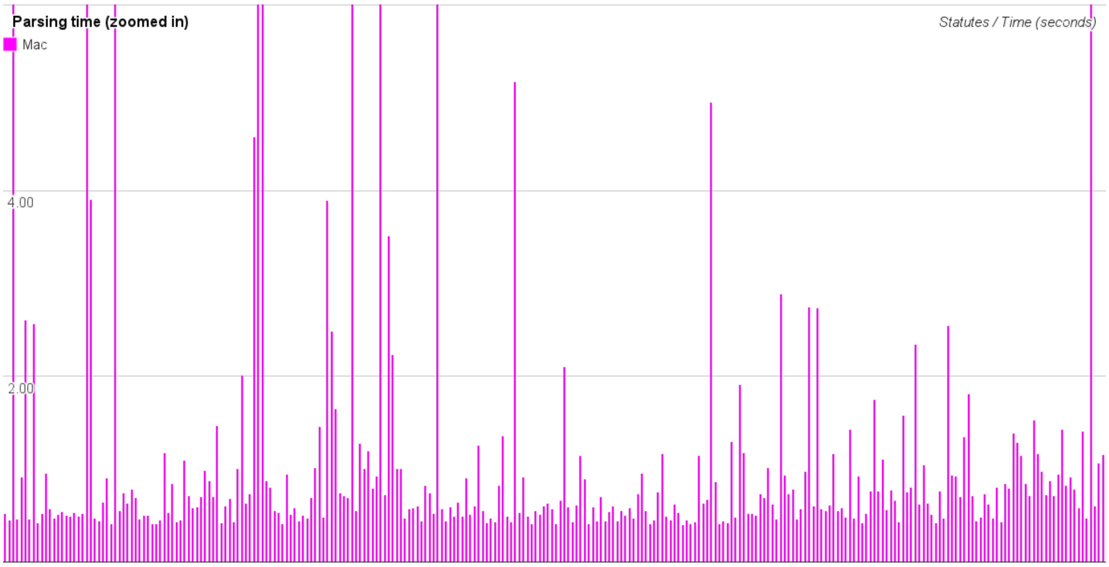

Figure 5.3: Chart zoomed in to 6 seconds showing running times on the Mac for the selected statutes

[28] It is the forth statute from the left, partially covered by the vertical axis legend

Chapter 6

Conclusions and Discussions

A prototype that handles parsing and mark up of SFS documents according to applicable standards was developed. The complete prototype was not fully functional in the sense that all the steps (downloading the documents, parse each document, markup each document with rdf, generate HTML ready to be shown in a browser) where not implemented. The interesting parts of the process was the parse and markup step and those worked and gave the sought after result.

6.1 Future development

The law parsing program was done as a prototype that Notisum could use to decide if they should rewrite their current parsing process to a more efficient one. During the development of the prototype there was not time to implement all the different types of legal documents that Notisum handles. However the prototype was designed with that in mind in a loosely coupled way which makes it easy to implement new modules and classes for other types of legal documents. The final step in a "complete process" where the formatted documents are transformed to HTML documents that have a certain layout and style sheets matching for example Notisum's site was not implemented.

6.1.1 RDF markup

Apart from the way the documents are parsed a new feature was introduced, marking up the documents with RDF, adding semantics to the documents according to ideas proposed by the Swedish government. The result of the RDF markup was successful in a sense that the resulting documents are semantically correct, we can for example extract multiple RDF triples from each legal document that are meaningful to computers and allows us to link our data in accordance with applicable standards for the Semantic web specifications. However the markup is not complete in the sense of exactly adhering to a standard or vocabulary that is being used by everyone working with Swedish laws, because there is none. Rättsinformationsprojektet was commenced to develop this standard and see to that all authorities producing legal documents would follow it, this work takes a lot of time. Since the project started in 2006 there has not really been that much progress, there has been a few enthusiasts that have put a lot of work into the project trying to get resources and support to do it full scale.

If the project would have been implemented already and all authorities were marking up their documents according to a standard vocabulary the RDF markup feature of the prototype would have been useless since the documents would already have been marked up when downloaded. But since that is not the case, the markup feature becomes even more important and hopefully it will help to shed light on Rättsinformationsprojektet and maybe speed up the work.

There are some attributes that are used in the prototype's markup that are not following the vocabulary suggested by Rättsinformationsprojektet. I felt that it was not top priority to exactly follow the suggestions and ideas proposed by Rättsinformationsprojektet that has been on hold for the last couple of years but rather marking up the documents with meaningful attributes that easily can be updated later on when hopefully, the work with Rättsinformationsprojektet is resumed and a standard is set.

6.2 Linked Data

Something that is not mentioned in this thesis is the potential dangers with linked data. It is not a subject that I have looked into that much during the work, I have focused on the positives and possibilities instead. But of course linked data can be misused by both humans and machines. It is something we have to take into mind, it might not be a problem today but might as well be in a few years as linked data grows more popular.

> "One of the defining characteristics of a successful information system is its level of exploitation by spammers." -Ian Davis[1]

[1] Taken from an article regarding threats to Linked data `http://blog.iandavis.com/2009/09/21/linked-data-spam-vectors/`

Bibliography

[1] *Regeringskansliets Rättsdatabaser*
`http://62.95.69.15/`
2012-09-09

[2] *Lagrummet.se - Portalen till svensk rättsinformation*
`http://www.lagrummet.se/`
2012-09-09

[3] *Dublin Core Metadata Element Set*
`http://dublincore.org/documents/dces/`
2012-12-25

[4] *The Semantic Web Intro, Presentation 2011-04-04*
Fariz Darari
`http://www.slideshare.net/fadirra/semantic-web-intro-040411`
2012-12-25

[5] *Introducing RDF*
`http://www.linkeddatatools.com/introducing-rdf-part-2`
2012-12-23

[6] *XHTML and RDF W3C Note 14 February 2004*
World Wide Web Consortium
`http://www.w3.org/MarkUp/2004/02/xhtml-rdf.html`
2012-12-23

[7] *RDFa in XHTML: Syntax and Processing - W3C Recommendation.*
Adida, B., et al.
`http://www.w3.org/TR/rdfa-syntax/`
2012-12-23

[8] *Introducing Linked Data And The Semantic Web*
`http://www.linkeddatatools.com/semantic-web-basics`
2012-12-23

[9] *Linked Data—The Story So Far (2011-07-05)*
Christian Bizer, Tom Heath, Tim Berners-Lee

`http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf`
2012-12-25

[10] *Linked Data (2006-07-27)*
Tim Berners-Lee `http://www.w3.org/DesignIssues/LinkedData.html`
2012-12-25

[11] *Linked Open Data: The Essentials*
Florian Bauer, Martin Kaltenböck
ISBN: 978-3-902796-05-9
`http://www.semantic-web.at/LOD-TheEssentials.pdf`
2012-12-25

[12] *Introduktion till rättsinformationssystemet*
Domstolsverket, Sveriges Domstolar
`http://dev.lagrummet.se/dokumentation/introduktion/intro-beslutsfattare.pdf`
2012-12-25

[13] *How to Publish Linked Data on the Web*
Chris Bizer, Richard Cyganiak, Tom Heath
`http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/LinkedDataTutorial/`
2012-12-23

[14] *Notation3 (N3) A readable RDF syntax*
Tim Berners-Lee
`http://www.w3.org/DesignIssues/Notation3.html`
2012-12-23

[15] *Text Processing in Python (chapter 4)*
David Mertz
ISBN: 978-0321112545
`http://gnosis.cx/TPiP/`
2006-06-07

[16] *Tutorialspoint - Python Regular Expressions*
`http://www.tutorialspoint.com/python/python_reg_expressions.htm`
2012-09-09

[17] *Uris, urls, and urns: Clarification and recommendations 1.0.*
T. Coates, D. Connolly, D. Dack, L. Daigle, R. Denenberg, M. Dürst, P. Grosso, S. Hawke, R. Iannella, G. Klyne, L. Masinter, M. Mealling, M. Needleman, and N. Walsh.
`http://www.w3.org/TR/uri-clarification/`
2012-12-23