MASTERTHESIS
Master Information Technology, Software Construction

"NEWTEC SEMANTIC WEB"

Author
Henrik Andersen
14. of June 2015

--------------------------
Henrik Andersen, Student
Student no. 20102880

Henrik Bærbak Christensen, Supervisor
Abstract

Tim Berners Lee the inventor of the WWW describes a dream in his book “Weaving the Web”. The dream is about connecting all things on the WWW in a semantic way so not only humans can interpret the information but also computer agents.

Newtec has an online service to provide information about customers' machines and products. This online service is of course targeted at the customers but it requires a lot of expert knowledge to use it and it is developed for human interaction and interpretation.

How can Tim Berners Lee’s dream and the Newtec online service be combined to add more semantics to the Newtec online service allowing both humans and computer agents to retrieve useful information without expert knowledge.

The W3C has defined a framework for developing a Semantic Web. Using these recommendations the Newtec online service can be transformed into a better semantically founded product without disturbing the human interaction and interpretation of the web interface.
# Table of contents

1 Motivation ........................................ 6

2 Problem Statement ................................. 7

3 Method .............................................. 7
   3.1 The Probabilities of the Newtec Data ........ 7
   3.2 The Semantic Web and Transformation of Newtec Data 8
   3.3 Ontology ................................... 8
   3.4 Inference .................................. 8
   3.5 Semantic web and data extraction ............ 8
   3.6 Newtec Web 2.0 Solution .................... 8
   3.7 Newtec Semantic Web solution ............... 8
   3.8 Computer Agents ............................ 8
   3.9 Overview of the Newtec Semantic Web Solution 9

4 Newtec Data ......................................... 10
   4.1 Product Data ................................ 10
   4.2 Customer Data ............................... 11
   4.3 Machine Data ............................... 11
   4.4 Batch Data ................................. 11
   4.5 Data Selection ............................... 12
   4.6 Data Collections ............................ 12
   4.7 Newtec Reports .............................. 13
   4.8 Human and Machine Data Probabilities ....... 14
      4.8.1 Customer, Machine, Batch and Product Data 14

5 Semantic Web Data Modeling ..................... 15
   5.1 Data and Meta Data .......................... 15
   5.2 The Semantic Web Data Model ............... 15
   5.3 RDF – Resource Description Framework ...... 17
      5.3.1 Newtec Product Data .................. 18
      5.3.2 Newtec Machine Data .................. 19
      5.3.3 Newtec Batch Data .................... 19
      5.3.4 Newtec Customer Data ................ 20
      5.3.5 Newtec Data RDF Graph and Triples ... 21
   5.4 Unique Identifiers ............................ 21
   5.5 Representing The Data in RDF/XML .......... 23

6 Newtec Ontology .................................... 24
   6.1 RDF type ................................... 24
   6.2 RDFS Class .................................. 25
      6.2.1 Newtec Batch Class .................... 25
      6.2.2 Newtec Customer Class ................ 25
   6.3 RDFS SubClassOf ............................. 25
1 Motivation
The invention of HTML and HTTP by Tim Berners-Lee\(^1\) in the CERN institute was an attempt to use and share information between researchers. Tim Berners-Lee developed both the protocol, markup language and the browser in the late 1990\(^2\). HTML as the name indicates is a markup language that consists of a number of tags that is interpreted by the browser to allow the user to access information. Tim Berners-Lee original idea of sharing information between researchers using the WWW's simple construct is perfect for this. The information of these static pages was typed in by the creator of the web page and viewed by users. This was an easy why of enabling researchers to share information. This was considered to be the first version of the world wide web or web 1.0. The evolution into web 2.0 came when information was shared not only from creator to user but also between users through web pages. Tim Berners-Lee was driven by a dream and here is his view of the world wide web and what it should become. Below is his dream from his book “Weaving the Web”

I have a dream for the Web... and it has two parts.

In the first part, the Web becomes a much more powerful means for collaboration between people. I have always imagined the information space as something to which everyone has immediate and intuitive access, and not just to browse, but to create. The initial World Wide Web program opened with an almost blank page, ready for the jottings of the user. Robert Cailliau and I had a great time with it, not because we were looking for a lot of stuff, but because we were writing and sharing our ideas. Furthermore, the dream of people-to-people communication through shared knowledge must be possible for groups of all sizes, interacting electronically with as much ease as they do now in person.

In the second part of the dream, collaborations extend to computers. Machines become capable of analyzing all the data on the Web -- the content, links, and transactions between people and computers. A "Semantic Web", which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy, and our daily lives will be handled by machines talking to machines, leaving humans to provide the inspiration and intuition. The intelligent "agents" people have touted for ages will finally materialize. This machine-understandable Web will come about through the implementation of a series of technical advances and social agreements that are now beginning.

Once the two-part dream is reached, the Web will be a place where the whim of a human being and the reasoning of a machine coexist in an ideal, powerful mixture. Realizing the dream will require a lot of nitty-gritty work. The Web is far from "done." It is in only a jumbled state of construction, and no matter how grand the dream, it has to be engineered piece by piece, with many of the pieces far from glamorous\(^3\).

The dream that Tim Berners-Lee describe above seems to have come true for the first part. The second part however does not seem to have fully reveled itself yet. What is the next step?, Tim Berners-Lee certainly does not hope that the evolution of the world wide web ends and the next generation WWW has already been shaped. A new era where data can be used in a more efficient way. A world where computers can dig into this vast information. W3C has defined a number of standards for the next generation web named “The Semantic Web“. These standards are the attempt to fulfill the dream of Tim Berner-Lee.

The company Newtec situated in Odense constructs machines for the farming industry. The company has been very successful of developing a machine that can grade potatoes. The machine has the capability to grade 120,000 products an hour. The system uses a vision system that allows it to evaluate each product and determining a number of attributes. The attributes includes length, diameter, shape and also quality defects it might suffer from.
The information is of course used to mechanical grade the products into different exits but a new business arose from the grading machines and the knowledge they had of each product. For the first time farmers had complete knowledge of each product and its attributes. This knowledge has many benefits when the goal is to create as many sell able products as possible. It is a key element to profit optimize for the farmers. In an attempt to gather the information Newtec developed a web service interface that allowed all the information from the machines to be collected. Newtec also build a front to the web service interface that allowed customers to view the information. The web interface is made so humans can interpret them in the best web 2.0 fashion. The new challenge for Newtec is the demand from customers to access the raw information or interpret the current information accessible through the web interface using computer agents. Combining the current Newtec Web solution with the semantic web seems to be a much better solution and opens new doors to the ways the information can be used by both humans and computer agents without developing proprietary systems that requires export knowledge.

2 Problem Statement

The semantic web seems like the wonder of the next generation WWW and how can this be transformed into something that a company like Newtec can use. With the two part dream of Tim Berners-Lee in mind, the data that Newtec gathers should be accessible for both human and machine interpretation. Since the launch of Newtec' web interface a reoccurring question has been asked by the customers. Whether each customer could dig into this vast information for data mining purposes or generation of customized reports. The problem with the Newtec interface is of course that limitation of resources which means that only key reports can be developed. There seems to be a need for more specialized reports and each customer seems to have their own report format.

The data flow from the Newtec machines is vast and Newtec has handpicked the data that has relevance for a research project. Newtec has over 1000 customers and over 20000 machines each delivering 10-50 batches and 1000000 products every day. The handpicked data will cover the areas of customer, machine, batch and product data. There will be data from 2 customers, 3 machines, 3 batches and 4 products.

Can the handpicked data from the Newtec web service be stored and presented in a way which allows both humans and computer agents to make use of it through a semantic web interface?

3 Method

The problem will be addressed using the methodology of the semantic web. That leads to number of key areas that needs to be analyzed and developed. The following sub sections explains each stage.

3.1 The Probabilities of the Newtec Data

The first phase of the report is to analyze the data that Newtec has provided. The data will be handpicked and selected to fit the purpose of the report. Newtec will provide a report that needs to be accessible through a web interface. To fulfill the report Newtec will provide a limited but selected sets of data. This analysis will cover the reasons behind the selection and which data Newtec has provided.
3.2 The Semantic Web and Transformation of Newtec Data
This phase will investigate the basis of the Semantic Web. This will lead to an analysis of the Newtec data and how to transform it into semantic web data. The semantic data model is one of the key elements of the semantic web which will be analyzed.

3.3 Ontology
The semantic web data that has been converted from the selected Newtec data will be accompanied with an ontology for the Newtec data the purpose is to enrich the semantic data. The ontology is one of the pillars of the semantic web and will help others understand the selected Newtec data.

3.4 Inference
The ontology that helps understand the semantic web data will also enrich the data further through inference. This phase of the report will try to determine what the Newtec ontology means for the data through the inference rules.

3.5 Semantic Web and Data Extraction
The semantic web data is only helpful if information can be extracted. Information retrieval is analyzed and the needed information for the selected Newtec report is constructed.

3.6 Newtec Web 2.0 Solution
Newtec already has a web page that works as an interface for the collected data. It is of interest to understand how an existing web page can be enriched with semantic data. A small Web 2.0 solution is developed that can serve as an interface for the selected data. This will help understand how the client and server side of the Web application has to change when the semantic data has to be introduced. The Web 2.0 solution will only cover navigation to a selected report and the generation of it.

3.7 Newtec Semantic Web solution
The Web 2.0 solution will be transformed into a Semantic Web solution. The Newtec Semantic data should be made available for web crawlers and web scrapers. This part of the report will look at how the server and client side have to change to enable this capability. Investigating which software tools are available to make the transformation on the server side. How to transform the Newtec data in accordance with the Newtec Semantic Web data discovered and the inference data from the Newtec ontology. How this data is queried with a software tool and given to the web clients upon requests. The client side has to change so the semantic data can be embedded along the normal Web page markup.

3.8 Computer Agents
The Newtec Semantic data has been made available through a normal Web interface. The data presented here is however static in the sense that the developer decides what kind of semantics to present. If the full potential of the Newtec Semantic data has to be presented it has to have an interface where the user can decide the combines of data to extract. This part will therefore have a REST interface that is exposed on the web server. The interface will be able to handle queries and respond with the Newtec data requested.
3.9 Overview of the Newtec Semantic Web Solution

The method should result in a design depicted in figure 3.1. Each subsection of the method is highlighted in the figure to give an overview of where in the model each step takes place.

Figure 3.1 – Newtec Semantic Web Solution

Picture of the Newtec Semantic Web using the method described.
4 Newtec Data

Newtec is a company that develops machines for the farming industry. The optical grader is one of the main products of Newtec. The machine has the ability to grade potatoes into a number of categories. The machine uses vision technology to record images of the products that passes under a camera. Each product is analyzed real time which results in a number of product attributes such as length, diameter, volume etc. The attributes of each product is compared to the grading criteria entered by the customer. The products are then dropped into exits selected by the customer in regards to the grading criteria. See figure 4.1.

![Figure 4.1 – Newtec Optical Grader](image)

The products passes through the machine for optical grading. The products are singulated at the first part of the machine. The second part of the machine evaluates the products using a vision system. The third part of the machine has a number of exits where the products can be placed according to the customers specification.

4.1 Product Data

The grading machine analyses the products and each product has a set of attributes. Data from each product is collected and send to Newtec’ web server. The product attributes is depicted in figure 4.2 and listed in table 4.1 below.

<table>
<thead>
<tr>
<th>Table 4.1 – List Of Product Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Attribute</td>
</tr>
<tr>
<td>Time stamp</td>
</tr>
<tr>
<td>Batch name</td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>Length</td>
</tr>
<tr>
<td>Diameter</td>
</tr>
<tr>
<td>Volume</td>
</tr>
<tr>
<td>Surface filter – Green spots</td>
</tr>
<tr>
<td>Surface filter – Black spots</td>
</tr>
</tbody>
</table>
4.2 Customer Data
Newtec has a number of customers, table 4.2 below shows the attributes associated with the customer data provided by Newtec. Each customer name is unique.

<table>
<thead>
<tr>
<th>Table 4.2 – Customer Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Data</td>
</tr>
<tr>
<td>Customer name</td>
</tr>
<tr>
<td>Contact person</td>
</tr>
</tbody>
</table>

4.3 Machine Data
The Newtec machines are unique and they each belong to one customer, table 4.3 shows the attributes associated with the machines.

<table>
<thead>
<tr>
<th>Table 4.3 – Machine Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Data</td>
</tr>
<tr>
<td>Machine ID</td>
</tr>
<tr>
<td>Customer name</td>
</tr>
</tbody>
</table>

4.4 Batch Data
The flow of products through the machine has to be labeled so the customer knows when a batch of products have started entering the machine and when the batch has finished. Ever customer uses batches for traceability and batches are products from a field or a subfield. The customer enters batch information into the grading machine when a new batch is started. The batch information indicates the start of a collection of products and the end of a previous collection. Table 4.4 below is a list of the batch attributes. The batch name is unique.
### Table 4.4 – Batch Data

<table>
<thead>
<tr>
<th>Batch Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine ID</td>
<td>A unique machine ID</td>
</tr>
<tr>
<td>Batch name</td>
<td>Batch name entered by the customer</td>
</tr>
<tr>
<td>Start time</td>
<td>The time when the batch was started in UTC</td>
</tr>
<tr>
<td>End time</td>
<td>The time when the batch was stopped in UTC</td>
</tr>
</tbody>
</table>

### 4.5 Data Selection

The samples are handpicked by Newtec, they are representative for the data flow that happens currently [April 2015]. Newtec has also provided this selection because the data samples show the relationship between customer, machines, batches and products. The three batches in table 4.5 are not randomly selected, but rather handpicked by Newtec from exactly two different customers and three grading machines. Even though the products are potatoes and may vary from customer to customer due to variety or physical growing conditions. There are no extra information from a different set of samples that will make a difference for the Newtec semantic web solution. The reason for this is that the products them self can vary in shape, color etc but all the products are represented with a number of generic attributes that does vary between potato varieties or growing conditions. This again is not by chance because it would be harder to compare batches of products from all customers if each set contained special attributes. The customer and machine data provided by Newtec covers the needed data to add the relationships between products and batches that runs through the Newtec grading machine owned by the customers.

### 4.6 Data Collections

The report will focus on the three samples of data selected and collected by Newtec. The samples will consists of a series of batches that are obtained from real customers. The samples consist of a number of products each with their set of attributes as described in section 4.1. Table 4.5 below shows a list of the three samples.

### Table 4.5 – List Of Data Samples

<table>
<thead>
<tr>
<th>Sample</th>
<th>Customer</th>
<th>Machine ID</th>
<th>Batch Name</th>
<th>Number of products</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Warnez</td>
<td>A501</td>
<td>Folva 13</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Warnez</td>
<td>A502</td>
<td>Folva 14</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Lamb Weston</td>
<td>A503</td>
<td>Innovator 1A</td>
<td>1</td>
</tr>
</tbody>
</table>

The samples are collected in files with ASC-II encoding. Newtec has made all the data available and Appendix A contains the raw data output from the machines in JSON format. The customers have been helpful running small batches for the case study as discussed in chapter 4.4. All data is obtained using the Newtec web interface from the machines. The data comes in a JSON format see example in figure 4.3 below.
Besides the information of the products and their attributes the JSON string contains a tag for the packet type and a tag for number of products. These are used to identify the packet and help to identify how many products are in each packet. This information is merely for parsing the data into a software model. The customer, machine and batch data is given in the same way using a JSON format. This data is also made available by Newtec and can be viewed in Appendix A.

4.7 Newtec Reports
The data that flows from the Newtec grading machines is currently used in a database with a web interface as the UI. The customers have access to the data through the UI and different reports can be generated. Newtec already has a vast number of reports that the customer can use and which is out of scope to cover. Newtec has therefore picked a key report that can be used in conjunction with the data samples. The report is also chosen because it provide a good foundation of introducing semantics. The content of the report is connected through the relationship of customers, machines, batches and products. This report is one of the most used and it is a batch report. The batch report generates an overview of a batch of products, below is an output of a report.

<table>
<thead>
<tr>
<th>Table 4.6 – Batch Report Data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Batch Report</strong></td>
</tr>
<tr>
<td><strong>Customer</strong></td>
</tr>
<tr>
<td><strong>Batch Number</strong></td>
</tr>
<tr>
<td><strong>Batch Elapsed Time</strong></td>
</tr>
<tr>
<td><strong>Number Of Products</strong></td>
</tr>
<tr>
<td><strong>Average length</strong></td>
</tr>
<tr>
<td><strong>Average diameter</strong></td>
</tr>
<tr>
<td><strong>Average Volume</strong></td>
</tr>
<tr>
<td><strong>Number of green products</strong></td>
</tr>
<tr>
<td><strong>Number of black products</strong></td>
</tr>
</tbody>
</table>

Figure 4.3 – Newtec Product Packet
Example of Newtec Machine Data in JSON format. The JSON string containing 2 products.

```json
{"packetType":"productListPacket",
"NoOfProducts":2,
"volume":[156,149],
"diameter":[63,63],
"length":[70,71],
"filter0":[4497,722],
"filter1":[4497,722],
```
4.8 Human and Machine Data Probabilities
The study of the semantic web and Newtec data requires some identification of what a human can interpret versus a machine. Analyzing the data and the reports it quickly becomes clear that it requires a human to understand the content of the reports and the product data provided by the grading machines. It would even be hard for most humans to link all the details together without some insight to the grading and potato industry.

4.8.1 Customer, Machine, Batch and Product Data
The product data itself containing information of each product is impossible for a third party machine to interpret, because the data itself is only raw numbers. A machine might at best find a link between the rows and the columns but the content itself is not clear or how they relate.
A human can interpret and understand the semantics of the raw data, because it is possible to look for more information if needed. Identifying the products as an example and what attributes are given for each product seems like an easy task. A computer would not have these capabilities unless a program with expert knowledge was developed. Semantics describing a “product” or a “attribute” can be quite overwhelming in term of a computer and how do they relate. Looking at the information given by Newtec it is clearly evident that the semantics are there it just need to be part of the raw data.
5 Semantic Web Data Modeling

The Newtec Data has to be transformed into semantic web information. The following section will go through the steps of transformation of the Newtec Data. A snippet of the Newtec data is selected to illustrate the transformation of customer, machine, batch and product data. See chapter 4.1 – 4.4 for information of the Newtec data. Data is often stored in relational database and lot of the comparison in this section will be between the relational data base and the transformation into the semantic web representation.

5.1 Data and Meta Data

The first step of transformation is to understand the information at hand. Data is really just values such as length or diameter and can be considered individual atoms of information but the meta data (semantics) connects the atoms to each other. This is often the job for a human to make sense of all these values and without further notice humans add the meta data using semantics. Computer systems have had the challenge for a long time to add this meta data. A good example is rational databases that have tables representing the data stored. The table column names are the meta data that explains the content of the column which of course often are just numbers or literals. The key for each row helps identify each object. The problem for a traditional relational database is that the convention or naming of the tables is often impossible to interpret for external resources without expert knowledge. This expert knowledge creates a problem when information has to be used by other computers without human intervention. To allow computers to share information automatically, data and meta data must be grouped together. In other words, without meta data there is nothing to talk about because no one knows the relationship between data. Without data there is nothing to say, since there is no information to interpret[4]. Combining data and meta data reduces the sharing problem at the semantic level. The portability of information is also easy since it is not dependent of predefined tables in relational databases, that contains the values. Treating meta data in the same way as data by grouping them simplifies tools because a single representation can be used for both type of information. These are powerful advantages for an information sharing approach and good reasons to chose the semantic web for interchanging information[4].

5.2 The Semantic Web Data Model

To understand how the semantic web data model is put together a snippet of the Newtec data is used, the raw data is available in Appendix A as JSON objects. The product, batch, machine and customer data from the Newtec data can be visualized in tabular form. Table 5.1, 5.2, 5.3 and 5.4 below shows a selection of the Newtec data.

| Table 5.1 Tabular Data Of Newtec Products
| Products
<table>
<thead>
<tr>
<th>ID</th>
<th>Time</th>
<th>Length</th>
<th>Diameter</th>
<th>Batch Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6124...</td>
<td>65</td>
<td>44</td>
<td>Folva 13</td>
</tr>
<tr>
<td>2</td>
<td>6124...</td>
<td>58</td>
<td>35</td>
<td>Folva 13</td>
</tr>
</tbody>
</table>

| Table 5.2 Tabular Data Of Newtec Machines
| Machines
<table>
<thead>
<tr>
<th>ID</th>
<th>Machine ID</th>
<th>Customer Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A501</td>
<td>Warnez</td>
</tr>
<tr>
<td>2</td>
<td>A502</td>
<td>Warnez</td>
</tr>
<tr>
<td>3</td>
<td>A503</td>
<td>Lamb Weston</td>
</tr>
</tbody>
</table>
Table 5.3 Tabular Data Of Newtec Batches

<table>
<thead>
<tr>
<th>ID</th>
<th>Batch Name</th>
<th>Machine ID</th>
<th>Start Time</th>
<th>End Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Folva 13</td>
<td>A501</td>
<td>2015-02...</td>
<td>2015..</td>
</tr>
<tr>
<td>2</td>
<td>Folva 14</td>
<td>A502</td>
<td>2015-02...</td>
<td>2015..</td>
</tr>
<tr>
<td>3</td>
<td>Innovator 1A</td>
<td>A503</td>
<td>2015-02...</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 5.4 Tabular Data Of Newtec Customers

<table>
<thead>
<tr>
<th>ID</th>
<th>Customer Name</th>
<th>Contact Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Warnez</td>
<td>Mr Warnez</td>
</tr>
<tr>
<td>2</td>
<td>Lamb Weston</td>
<td>Mr Lamb Weston</td>
</tr>
</tbody>
</table>

It is easy to understand how a relational database would represent these tables since they can almost be converted directly into database tables with the column names and the ID in the rows as keys. This is also very easy for a human to list the data in tabular form and hence why relational database are so widely used. To dig deeper into the principals of semantic web and how to convert the data above into semantic web data, a short discussion of data sharing and pairing data values with meta data needs to be explained. The explanation will take its root in the relational database and how to move to the semantic web data model. This will also help to understand how the Newtec data will be represented.

Data sharing and meta data becomes critical when data moves from a central storage facility to multiple storage facilities, which is just how the WWW can be viewed. The Newtec data example seems to be too small even to consider to talk about multiple data facilities but some of the Tim Bernes Lee ideas of the WWW and semantic web is to think about every piece of data as distributed. The Newtec data in the tables above can be divided and shared in a number of ways. The problem arises when multiple servers or sources wants to store and use the information. A lot of data is stored in relational databases, but the WWW is a completely different matter. The WWW does not have a database manager and the data available is vast. So even though it seems logical best to represent the data exactly as it is shown in table 5.1-5.4 it is not the way forward.

There are a couple of strategies to overcome the problem of splitting the data from tabular form into multiple storage or just to view the data as separate units. An obvious way of sharing data is row by row adding the meta data to each row. The problem of sharing data row by row is the need to understand what each column represents. In table 5.1 above ID is the first piece of information followed by time, length and so forth. When more servers are storing data they each need to have knowledge of the columns and any change to the columns requires a change in the other servers to preserve the compatibility, that leads some overhead. See figure 5.1 below.

Figure 5.1 – Row Strategy
Distributing data across the Web, row by row. A common schema is needed, which column is which?

Another strategy is to store data using columns, adding the meta data to each column. Each server is responsible for one or more columns. This strategy gives flexibility in a different way. In table 5.1 above this strategy would mean as an example that the columns ID, time and Length could be stored in
one server and the columns Diameter, volume and Green spots in another server. If certain data does not have relevance it can be left out, in the example above if only ID and length was needed the server or servers holding this data could be queried and the rest of the data would never be retrieved. The problem is the coordinations of entities in each row, since there has to be some general knowledge of what entity each row represents, this also leads to some overhead. See figure 5.2 below. 

A third strategy of sharing data is cell by cell. This combines the strategy of the row and column strategies described above, adding meta data from both the row and column. It also has the cost of both strategies which means a big overhead to keep track of each cell. The data in each cell has to be associated with information about the entity and the property. Data can be distributed across many servers and data can always be put back together. The W3C\(^5\) has a number of design goals for the semantic web and the RDF format it is expressed in. One of these design goals are the AAA slogan, “Anyone can say Anything about Anything”. If this has to be used in praxis the only way to store and share information is by using the cell to cell strategy\(^6\). See figure 5.3 below. 

The strategy also implies the penalty of adding entity and property information to each cell. The cost of extra information in exchange of the ability to distribute the data on a vast amount of servers is often small compared to the cost of having human knowledge to ensure compatibility of the data. The idea of splitting the data in a cell by cell manner also means that the data can be stores anywhere and be put back together. This is exactly the dream of Tom Berners-Lee\(^3\). Considering the cost of human knowledge to ensure compatibility would properly be considered small or reasonable in environments using traditional rational database. Considering the World Wide Web in relation to data compatibility a more maintenance free approach is needed and is therefore the obvious and chosen strategy for the semantic web\(^7\).

5.3 RDF – Resource Description Framework
The idea behind representation of data in the semantic web is explained in section 5.2. Every cell is therefore represented with three values - data, entity and property. W3C has defined a way of expressing the information using Resource Description Framework\(^7\) (RDF) which can be used to
identify the information at each cell. The basic building block of RDF is referred to as triples. The
identifier for each row in the table is called the subject of the triple. Which is not by coincidence
notation from elementary grammar the subject is the thing that a statement is about. The identifier for
the column is called the predicate of the triple. Again no coincidence since predicate is seen as a
property that a subject has or is characterized by. The value in the cell is the object of the triple.
Triples can be represented in a number of ways, a human depiction as a graph. See figure 5.4 below for
the graph representation.

Figure 5.4 – RDF Graph
RDF graph representation
of a triple.

A computer system however cannot work with graphs and a range of formats are available. Turtle and
XML/RDF are two formats recommended from W3C. Below is an example of the triple
representation of both formats, stripped. Turtle and RDF/XML will be used throughout the report.

Turtle:
<Subject>
  Predicate Object;

XML/RDF:
<Description about:"Subject">  
  <Predicate>Object</Predicate>
</Description>

5.3.1 Newtec Product Data
Identification of each cell with both subject and predicate has to be unique. In the example of the
Newtec product data, table 5.1, every row has a unique ID identifying each product and every column
has a unique name representing the attributes of the products. Using the unique ID and the column
names from table 5.1 a snippet of the Newtec product data can easily be transformed into triples.
Table 5.5 and figure 5.5 shows the cells containing length, diameter and batch name from Table 5.1
being transformed. It is worth to notice that the batch name is not only literal, it actually represents
another subject from the batch table.

Table 5.5 Sample Triples - Product

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID:1</td>
<td>Length</td>
<td>65</td>
</tr>
<tr>
<td>ID:1</td>
<td>Diameter</td>
<td>44</td>
</tr>
<tr>
<td>ID:1</td>
<td>Batch name</td>
<td>Folva 13</td>
</tr>
<tr>
<td>ID:1</td>
<td>Time</td>
<td>2015-02...</td>
</tr>
<tr>
<td>ID:2</td>
<td>Length</td>
<td>58</td>
</tr>
<tr>
<td>ID:2</td>
<td>Diameter</td>
<td>35</td>
</tr>
<tr>
<td>ID:2</td>
<td>Batch name</td>
<td>Folva 13</td>
</tr>
<tr>
<td>ID:2</td>
<td>Time</td>
<td>2015-02...</td>
</tr>
</tbody>
</table>

Figure 5.5 – RDF Graph
RDF graph representation
of sample triples from
the Newtec product
data table 5.1
product ID:1.
The RDF graph representation above is quite easy to understand. Squares represent values such as 65 for the length, and the arrow represent the predicate. The ellipsoids represent subjects and notice that the predicate “Batch name” does not point to a value (square), it points to another subject “Folva 13” (ellipsoid) which is represented in the batch table, table 5.3.

5.3.2 Newtec Machine Data
The Newtec machine data from table 5.3 can also be transformed using the machine ID which is unique. Considering that the customer owns a number of machines which means that the subject would be the machine name which is unique and the predicate would be “HasOwner”. The object would then be the customer. Figure 5.3 depicts the table 5.6 in a RDF graph. The unique subject is good to use for human readability but the name A501 is getting lost. The name A501 now has to be represented as a literal, which is also depicted in figure 5.6. The name is made more specific by calling it “MachineName”.

<table>
<thead>
<tr>
<th>Table 5.6 Sample Triples - Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
</tr>
<tr>
<td>A501</td>
</tr>
<tr>
<td>A501</td>
</tr>
</tbody>
</table>

Figure 5.6 – Machine RDF Graph
RDF graph representation of the customer table 5.2 for A501.

The transformation makes sense in semantic way and it good practice to make the triples human readable for development purposes. The transformation above is good example of better semantics than a traditional table would give\[4\].

5.3.3 Newtec Batch Data
Table 5.3 Newtec batch data can also be transformed using a similar direct tabular transformation approach as Newtec product data. The question is if this is the best way to go. The design pattern that “Anyone can say Anything about Anything” can be used to create better semantics. Transforming the batch names into triples can be done if it is realized that the ID column is not needed, since every machine ID is unique and there cannot be two of the same batch names. The unique ID is merely identification and database uses them as keys to connect tables together. A human already has an understanding that the machines are responsible for running the batches of products. Further more the description “batch name” could be viewed from the machine point of view and named “Ran batch” that way it is more readable and easier to understand. It would mean that the column with the batch names could be transformed as shown in table 5.7 and table 5.8, where a better semantic transformation is taken place. Figure 5.7 depicts RDF graph.
Having identified the batch name as the unique key, the rest of table 5.3 can be transformed like the Newtec product data. Figure 5.8 below depicts the RDF graph of the transformation. Since the batch name is used as subject it necessary to add the name of the batch as a literal. The name of the batch is called “BatchName” to make it specific.

Table 5.9 shows a sample triple of the batch data from table 5.3. In worth noticing that the figure 5.4 both the object “Folva 13” is the subject in figure 5.8. This shows the relationship between the triples.

Table 5.9 Sample Triples - Batch

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Folva13</td>
<td>Start time</td>
<td>2015-02</td>
<td>Folva14</td>
<td>Start time</td>
<td>2015-02</td>
<td>Innovator1A</td>
<td>Start time</td>
<td>2015-02</td>
</tr>
<tr>
<td>Folva13</td>
<td>End time</td>
<td>2015-02</td>
<td>Folva14</td>
<td>End time</td>
<td>2015-02</td>
<td>Innovator1A</td>
<td>End time</td>
<td>2015-02</td>
</tr>
<tr>
<td>Folva13</td>
<td>BatchName</td>
<td>Folva 13</td>
<td>Folva14</td>
<td>BatchName</td>
<td>Folva 14</td>
<td>Innovator1A</td>
<td>BatchName</td>
<td>Innovator 1A</td>
</tr>
<tr>
<td>A501</td>
<td>Ran batch</td>
<td>Folva 13</td>
<td>A502</td>
<td>Ran batch</td>
<td>Folva 14</td>
<td>A501</td>
<td>Ran batch</td>
<td>Innovator 1A</td>
</tr>
</tbody>
</table>

5.3.4 Newtec Customer Data
The Newtec customer data from table 5.4 can be transformed like the batch data. The customer name is unique. Figure 5.9 depicts the table in RDF graph and table 5.10 shows a sample triple of the customer data. Like the batch and machine data the name of the machine also has to be added using the predicate “CustomerName”
5.3.5 Newtec Data RDF Graph and Triples
The transformation of the Newtec Data from table 5.1 – 5.4 into triples can be viewed as one big RDF graph shown in figure 5.10.

Even though the triples from the sample can be viewed as one big graph it is obvious that the graph quickly becomes unmanageable. The graph also gives the illusion that the data is connected, that is not the case. The graph is merely constructed from the triples and there deductible relationships.

5.4 Unique Identifiers
The RDF graph in figure 5.10 shows how the triples can be connected. The connection of the triples however assumes that the different identities in the triples are in fact the same resource. Each entity has to be identified uniquely otherwise it is not possible to know if the batch name from the the product table is the same as the batch name from the batch table. The identities of each resource is solved using unified resource identifiers (URI). The semantic web and the RDF uses the URI from foundational web technology. The widely used URL is in fact a special case of URI, and if a URI that can be resolved in browser on the WWW it is a URL. The semantic web is not only for resources on the WWW. Any resource can be described and the objects around us can all have a URI that allows the semantic web to
identify the resource. URI provides a global identification for a resource that is common across the web. This will be the case of the Newtec data since the products and batches does not live as URLs on the WWW but are merely represented as URI's in well defined name space.

It is now time to identify the Newtec URI's that make each resource unique. When that has been carried out, a few more bits of information can be added to the semantic web data found in table 5.5, 5.6, 5.7 and 5.9. For the sake of the reader a different notation than the full URI will be used and it is common to use a name spaces in web technology for this. Since Newtec already has a unique place on the WWW (www.newtec.com) which is their web page it is easy to use this in combination with the customer, machine, batch and product to uniquely identify each entity.

First the common name space is chosen that will have all the entities as shown in table 5.10 below. The table contains the URI in the abbreviation scheme called qname\[12\] to help identify the data.

<table>
<thead>
<tr>
<th>Table 5.10 URI and qname for the Newtec database</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
</tr>
<tr>
<td>qname</td>
</tr>
<tr>
<td><a href="http://www.newtec.dk/webservice/NewtecDatabaseScheme#">http://www.newtec.dk/webservice/NewtecDatabaseScheme#</a></td>
</tr>
<tr>
<td>nds</td>
</tr>
</tbody>
</table>

The sample triples earlier constructed in table 5.5, 5.6, 5.7 and 5.9 can now be put together using the qname for the Newtec data and concatenating the table name products, batches and customers. Table 5.11 shows a snippet of the concatenation of the tabular information and the name space of Newtec. The full list can be seen in appendix E.

<table>
<thead>
<tr>
<th>Table 5.11 Concatenation of Newtec name space and tabular data of products</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name space</td>
</tr>
<tr>
<td>nds (Table 5.7)</td>
</tr>
<tr>
<td>nds (Table 5.7)</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

The result of the concatenation for the products becomes nds:product(number). The batches and customers uses their names with no white spaces to uniquely identify them they become eg. nds:Folva13 and nds:Warnez. Each predicate also needs to be uniquely identified and the same naming convention is used. The ID from the product table becomes nds:ProductID and the length becomes nds:ProductLength and so fort. Table 5.12 is a snippet of the full list of triples, which can be seen in appendix E.

<table>
<thead>
<tr>
<th>Table 5.12 Triples representing the data from table 5.1 – 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
</tr>
<tr>
<td>nds:Product1</td>
</tr>
<tr>
<td>nds:Product1</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
5.5 Representing The Data in RDF/XML

Table 5.12 is a nice visualization of the triples but it cannot be used for interpretation of a computer, see
full list in appendix E. The next step is to convert the triples into a format that a computer can work
with. There are several serialization alternatives in use. N-triples, Turtle and RDF7XML are all
alternatives. RDF/XML and N-triples are raw RDF triples and not easy to read or print. Turtle is more
compact and quite suitable for the human consumption. Below is a snippet of table 5.12 serialized into
RDF/XML and Turtle.

Newtec data expressed in the semantic web language RDF output format is RDF/XML:11

```
<rdf:RDF
  xmlns:nds="http://www.newtec.dk/webservice/newtec-database-scheme#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <nds:Product rdf:about="http://www.newtec.dk/webservice/NewtecDatabaseScheme#Product1">
      <nds:ProductID>1</nds:ProductID>
      <nds:ProductLength>65</nds:ProductLength>
    </nds:Product>
  .
  <nds:Batch rdf:about="http://www.newtec.dk/webservice/NewtecDatabaseScheme#Folva 13">
  </nds:Batch>
</rdf:RDF>
```

Newtec data expressed in the semantic web language RDF output format is turtle:10

```
PREFIX nds:<http://www.newtec.dk/webservice/NewtecDatabaseScheme#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

nds:Product1 nds:ProductID "1";
  nds:ProductTime "2015-02-28 07:25:25";
  nds:ProductLength "65";

```

23
6 Newtec Ontology

Ontologies are considered one of the pillars of the Semantic Web, although they do not have a universally accepted definition[40]. A Semantic Web vocabulary can be considered as a special form of usually light-weight ontology[13]. RDFS (Resource Description Framework Schema) is the most basic schema language commonly used in the Semantic Web technology stack[14]. It is lightweight and very easy to use and get started with. In fact, many of the most popular RDF vocabularies are written in basic RDFS[15].

If external sources want to make use of the Newtec data, they need a way to interpret it. How does an external source know how to understand the Newtec data. The challenge is equal to two people with different languages that want to exchange information, they need some translator. RDFS is similar to a language translator in sense that it allows data created by different sources for different uses at different times to be connected using RDFS and Semantic Web technologies. It provides a kind of universal translation between alien languages or different semantics to be compatible. The first step down this path is to define common vocabularies. RDF is a graph database and RDFS on the other hand, is object oriented in its nature. That is, RDFS is fundamentally about describing classes of objects with its properties and there relationships. The Newtec data is no different, an example could be the products which could be instances of a class names “Product” with some attributes. The “Product” is associated with a batch and so fort. In the following sections different RDFS and OWL constructs will be used to create an ontology for the Newtec data. The chapter will use the Turtle notation to show the RDFS constructs for the Newtec data along with RDF graphs.

6.1 RDF type

```
@prefix nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

```

It will be easy to find all the products in the triple store with the rdf:type information.
6.2 RDFS Class

The basic construct for specifying sets in RDFS is called rdfs:Class. Since RDFS is expressed in RDF the class is represented with a triple of the predicate rdf:type and the object is rdfs:class. The rdf:type is enough to know that all the products can be grouped to the type “Product” as in section 6.1. It is however not very flexible if the products properties such as length and diameter needs to be related to the product. The RDFS class construct is much better for this purpose since a rdfs:class can have properties. The Newtec data have four obvious classes customer, machine and product which all have properties. Below is a Turtle representation of the Newtec rdfs:class.

```turtle
@prefix nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

nds:Customer rdf:type rdfs:class.
nds:Machine rdf:type rdfs:class.
nds:Product rdf:type rdfs:class.
```

6.2.1 Newtec Batch Class

The Newtec batches have two properties for the start and end time of the batch. These properties could be viewed as specialized classes or sub classes of the batch class. The Newtec batch would always belong to the “StartedBatch” class and if the batch has ended it will also belong to the “EndedBatch” class. The two subclasses could be named “nds:startedBatch” and “nds:endedBatch” and will be created along with the other classes. See the Turtle representation below.

```turtle
@prefix nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

nds:StartedBatch rdf:type rdfs:class.
nds:EndedBatch rdf:type rdfs:class.
```

6.2.2 Newtec Customer Class

The Newtec Customer data has a property for the contact person see table 4.2. This could be modeled as a class of its own. The reason for this is that a contact person has lots of details and even though the data only has the contact name as shown in table 4.2, it does in fact represent a human with a lot more details. See Turtle representation below.

```turtle
@prefix nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

nds:ContactPerson rdf:type rdfs:class.
```

This semantic construct leaves room for easier development in the future, where more details might have to be added.

6.3 RDF SubClassOf

The identification of specialized classes of the batch, see section 6.1, leads to the question of how to incorporate it into the semantics. The rdfs:subClassOf is the construct for this.
6.3.1 Newtec Batch SubClassOf

The batch is always semantically started, since no batch can exist without being started. The instances of the batch from table 5.3 therefore always have a “rdf:type” of the “nds:StartedBatch”. If the batch has ended it is also a “rdf:type” of the “nds:EndedBatch”. The “nds:StartedBatch” and “nds:EndedBatch” are subclasses to the “nds:Batch”. That means inheritance of the properties from the superclass in this case “nds:Batch”. The rdfs:subclassof of the batch is shown in RDF graph in figure 6.2 below along with the Turtle representation, using the data from table 5.3.

```
@prefix nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

nds:StartedBatch rdfs:subclassof nds:Batch .
nds:Folva13 rdf:type nds:StartedBatch .
nds:Folva14 rdf:type nds:StartedBatch .
nds:Innovator1A rdf:type nds:StartedBatch .
```

It can be seen that the batch “Innovator 1A” has not ended and therefore has no “rdf:type” to “nds:EndedBatch”. This will happen when the batch “Innovator 1A” has been declared ended by the machine and the information reaches the RDF store. This is another beauty of using triples, because only valid data needs to be added, in comparison to a database table which have fixed tables.

6.3.2 Newtec Contact person SubClassOf

The contact person identified as a class in section 6.2.2 “nds:ContactPerson” can also be referenced with a “rdfs:subclassof”. The Newtec ontology is not the first to model a person and the FOAF ontology has already a well used definition of a person. It would therefore make sense to subclass the FOAF ontology[^10]. The FOAF (Friend Of A Friend) is a format for supporting distributed descriptions of people and their relationships. The use of an existing ontology makes the data easier to understand and reuse[^14].

[^10]: FOAF (Friend Of A Friend) is a format for supporting distributed descriptions of people and their relationships.
[^14]: The use of an existing ontology makes the data easier to understand and reuse.
See the RDF graph in figure 6.3 and the Turtle representation below for the Newtec data from table 5.9.

![RDF Graph](image)

**Figure 6.3 – RDF Graph, customer and foaf:Person**  
RDF Graph of the rdfs:subclassof expressing the fact that the contact person from table 5.9 can be expressed with a rdf:typeof to a “nds:ContactPerson” and that the “nds:ContactPerson” is a “rdfs:subclassof” the FOAF ontology “foaf:Person”.

### 6.2 RDFS Property
The identification of the four basis classes customer, machine, batch and product makes it possible to describe how each class has a number of properties. The properties for the Newtec products can be seen in table 4.1, which contains length, diameter, volume and black/green spots. The properties can be described using rdfs:property, see Turtle representation below. The property names are taken from table 5.9.

```turtle
@prefix nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

nds:ProductLength  rdf:type rdfs:property .
nds:ProductDiameter  rdf:type rdfs:property .
nds:ProductGreenSpot  rdf:type rdfs:property .
```

### 6.3 RDFS Domain
The rdfs:property is enough to explain that the property exists but not enough to tell that it is an actually property of a “Product”. RDFS uses the rdfs:domain to state that a property belongs to a “nds:Product”, see the Turtle representation below.

```turtle
@prefix nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

```

The use of domain comes with some consideration because it specifies the class membership of
individuals and is globally asserted. This has the consequent that all future use, regardless of context of the resource (property) will be subject to the domain assertion which could introduce inflexibility. If the “nds:Product” property length was introduced as “nds:Length” and in the domain of the “nds:Product”. It would be wrong to reuse the property “Length” in an other context, as an example if the length of the machine had to added to the ontology. The reuse of “nds:Length” for the machine would imply that a machine was of the type “nds:Product” which is not correct[4].

6.4 RDFS Range
The range of a property is the set of values that the property can accept. The Newtec properties can all be made see the Turtle representation below.

```turtle
@prefix nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

nds:BatchStartTime  rdfs:range xsd:dateTime .
nds:BatchEndTime  rdfs:range xsd:dateTime .
```

6.5 Newtec Temporal data
There are a few time stamps in the data from table 4.1 and table 4.4. It includes the time stamp for the products and the start/end of the batches. This data is temporal an W3C already has an ontology for managing time governed by the Time Ontology in OWL[18].

The temporal data can be described using this ontology and the Turtle below shows an example of the product and Batch time stamps using the Time Ontology.

```turtle
@prefix nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix time: <http://eee.w3.org/2006/time#> .

nds:ProductTime a time:Instant .
nds:BatchStartTime a time:Instant .
nds:BatchEndTime a time:Instant .
```

6.6 Newtec Measurements and Units
The Newtec products comes with some properties that uses standard units like millimeter and gram. Looking at table 4.1 and figure 4.2 the description of the properties are expressed with units. These units cannot be found in table 5.1 which is a tabular example of some the products, resulting in a loss of semantics.

6.6.1 Length, Diameter and Volume Units
The solution is not straightforward because this is not easily modeled. The problem is that the measurement is a value and a unit as one block and there is a problem to express this without tension
between expressibility and usability.
The first approach is to simply add the unit to the predicate or the object description as shown in the RDF graph in figure 6.4.

![Figure 6.4 – RDF Graph Product](image)
RDF Graph of a solution to represent both measurement and value.

The problem with these two approaches are that the semantics are incorporated into the literals. It is hard to dig out the information without expert knowledge and string manipulation.
A second approach is to combine the data into what it really is, a measurement with a value and a unit. This approach means that the product length has to represented as an object linked to the actual product and the length object has two properties for the value and the unit. The RDF graph in figure 6.5 below shows the approach.

![Figure 6.5 – RDF Graph, Product](image)
RDF Graph of a solution to represent both measurement and value using an indirection of the “ProductLength”.

Semantically this looks very nice. The measurement is now represented for what it is with a value and a unit. The value can be expressed using the rdf:value and the unit can be pointed to an external ontology for science and engineering used by NASA\(^\text{[19]}\). However this comes with a price in practical terms. It will introduce a lot of new triples and it will eliminate the ability to utilize OWL semantics with any of these kinds of properties, because of the level of indirection\(^\text{[4]}\).
A third approach is to simply add another property to the product that describes the unit which the length it is measured with. The RDF graph in figure 6.6 shows this approach.

![Figure 6.6](image)
RDF Graph of a solution to represent both measurement and value using an extra property on the “Product”.

This approach meets somewhere in the middle of the two others, which leaves a solution where some
of the semantics are kept but without the problem of embedding it into the literals. It also means that extra triples has to added explicit for the products. Below is an example of the unit representation for a Newtec Product in Turtle.

```turtle
@prefix nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#>.
@prefix ops: <http://www.openphacts.org/units#>.
@prefix qudt: <http://qudt.org/schema/qudt#>.

nds:ProductLengthUnit rdf:type rdfs:property.
nds:ProductDiameterUnit rdf:type rdfs:property.
nds:ProductVolumeUnit rdf:type rdfs:property.

#New triples introduced
nds:Product1 nds:ProductLengthUnit ops:Millimeter.
nds:Product1 nds:ProductDiameterUnit ops:Millimeter.
nds:Product1 nds:ProductVolumeUnit qudt:Gram.
```

### 6.6.2 Pixel Units

An ontology to describe the pixels was not found in the wild, therefore a new rdfs:class is modeled to represent it. This approach for the Newtec data representing the black and green spots in pixels from table 5.1 will is shown in Turtle below with an example of a product at the end.

```turtle
@prefix nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

#New class to represent the unit pixel
nds:Pixel rdf:type rdfs:class.
nds:Symbol rdf:type rdfs:property.
nds:Symbol rdfs:domain nds:pixel.

nds:ProductGreenSpotUnit rdf:type rdfs:property.
nds:ProductBlackSpotUnit rdf:type rdfs:property.

#Example of a products units for pixel
nds:Product1 nds:ProductGreenSpotUnit nds:Pixel.
```

### 6.7 OWL InverseOf

The ownership of the machines are currently represented with “HasOwner”, see chapter 5. This relationship comes from the machine (subject) to the customer (object) or is simply viewed from the machine side. Semantically it would be nice to show the opposite, where the ownership is viewed from the customer side. This is easy to describe with owl:inverseOf, The Turtle representation of the machine and customer relationship viewed from the customer is shown below.

```turtle
@prefix nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.

nds:HasOwner rdf:type owl:objectProperty.
nds:Owns rdf:type owl:objectProperty.
nds:Owns owl:inverseOf nds:HasOwner.
```
6.8 The Newtec Ontology
The complete Newtec Ontology can be seen appendix F in Turtle format. A snippet is displayed below.

@prefix nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
nds:Customer rdf:type rdfs:class .
nds:ProductBlackSpotUnit rdf:type rdfs:property .
7 Inference

Inference on the Semantic Web can be characterized by discovering new relationships. The Semantic Web data is modeled as a set of (named) relationships between resources. The Newtec data model was constructed in chapter 5 from the data provided by Newtec. Inference means that automatic procedures can generate new relationships based on the data and based on some additional information in the form of a vocabulary, e.g., a set of rules. These rules are expressed in chapter 6 as the Newtec ontology. The inference that comes from the Newtec data and the ontology will be described in this section and will be accessible in the web pages generated in chapter 10.

7.1 Inference in the Newtec data

The Newtec ontology in chapter 6 described the relationship between various pieces of data. This is helpful for users of the data when data is extracted. Looking at it in broader terms the data is living on the Web and a major concern for making data more useful is to have it behave in a consistent way when combined with data from multiple sources. Data from multiple sources can already be seen in the Newtec ontology because some of the Newtec data is referring to external sources such as the FOAF framework[16].

An inference based system for describing the meaning of Semantic Web constructs is elegant, but it should make our data more useful. The inference happens in the application with a new capability that responds to data extraction not only on the original Newtec data (asserted triples) but also to the data constructed from the inference (inferred triples). Figure 7.1 depicts this new capability as the “Inference and Query Engine”[14].

![Figure 7.1 – Semantic Web Architecture](image)

Semantic Web architecture with the inference and query engine on top of the RDF store serving the application interface.

The level of inference depends on the inference engine’ capabilities. The three main levels is none, RDFS and OWL. If no level of inference is supported only the original triples will be available. If RDFS supported a small set of inference is performed as defined in the RDFS standard[17]. If the inference engine supports OWL the large set of inference is possible in accordance to the OWL standard[20].

The level of inference leads to asserted triples versus inferred triples. Asserted triples is the triples that were asserted when the RDF store was populated by one or more sources. The Newtec data expressed
as triples in chapter 5 are the asserted triples. The inferred triples are the additional triples that inferred by one of the inference rules described in the Newtec ontology described in chapter 6. There is of course no logical distinction between inferred and asserted triples, the query and inference engine from figure 7.1 will draw exactly the same conclusions from an inferred triple as it would have from an inferred triple.

The RDFS and OWL standards define what inferences are valid, given certain patterns of triples. When the inference happens is outside the range of the definition of RDFS and OWL. It is clearly important for the implementation that conforms to these standards. Inference can be viewed from two extremes, in one extreme inference happens when data extraction is required. This would mean that the inferred triples are never stored in the persistent RDF store. This gives flexibility if data triples are added, changed or deleted or the ontology rules change but requires that all the inferred triples has to be reconstructed for every data extraction. This can be referred to as “just in time inference”. This fits with figure 7.1 where the queries and inference exist as one unit.

The other extreme is that all inferred triples are added to the RDF store when triples are added, changed and deleted or the ontology changes. This gives an overhead when ever any change to the store happens. This can be referred to as “caching inference”. Figure 7.1 shows the inference happening along with the queries but this not the case, since all inference has been performed prior to the query and the inferred triples are already persistent in the RDF store.

Which of the two strategies to take depends on the implementation. The Newtec data does seldom changes but triples are often added. The number of queries could potentially be large depending on the usage of the application. The strategy for the Newtec solution will be “caching inference” where inference happens when the data is entered into the RDF store.

7.2 Inferred Triples From the Newtec Data Model

The Newtec data model described in chapter 5 and the Newtec ontology constructed in chapter 6 leads to a number of inferred triples. The following section will explain the consequence of the inference between the Newtec data triples and Newtec ontology. It is now time to put these things together and using the data provided by Newtec, which can be viewed in full in appendix A and the Newtec ontology, which can be viewed in full in appendix F. The implementation of the application to hold the Newtec data is in Java. Apache Jena is a framework for Semantic Web data in Java and will be used for the implementation, see chapter 11 for further details on use of the Apache Jena framework. The Java source code to show the different levels of inference are kept minimal and only a snippet is shown to make it easier to understand. Packages and import statements are omitted as well as most of the error checking and validation code. The output of all resources from Jena is too extensive to show and only “Product1” and “Warnez” will be examined.

```java
public class RDFInferenceTest {
    public static void main(String[] args) {
        StringBuilder result = new StringBuilder();
        Model schema = FileManager.get().loadModel("/home/henrik/workspace/JenaTutorial/src/newtec_owl.ttl");
        Model data = FileManager.get().loadModel("/home/henrik/workspace/JenaTutorial/src/newtec_rdf.ttl");

        //No Inference
        Model model = ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM);
        model.read(data, schema, "Turtle");
        printIndividuals(model, result);
    }
}
```
//RDFS Inference
model = ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM_RDFS_INF);
model.read(data, schema, "Turtle");
printIndividuals(model, result);

//OWL Inference
Reasoner reasoner = ReasonerRegistry.getOWLReasoner();
reasoner = reasoner.bindSchema(schema);
InfModel infmodel = ModelFactory.createInfModel(reasoner, data);
model = ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM, infmodel);
printIndividuals(model, result);

public static void printIndividuals(Model model, StringBuilder result) {
    ExtendedIterator iIndividuals = model.listIndividuals();
    while (iIndividuals.hasNext()) {
        Individual i = (Individual)iIndividuals.next();
        result.append("Individual: " + i.getLocalName());

        StmtIterator iProperties = i.listProperties();
        while (iProperties.hasNext()) {
            Statement s = (Statement)iProperties.next();
            result.append("  " + s.getPredicate().getLocalName() + " : " + s.getObject().toString() + "\n");
        }
        iProperties.close();
        result.append("\n");
    }
}

7.2.1 No Inference
Using Apache Jena without any inference will produce the Newtec data triples.

Individual: Product1  ID : 1
BatchName : http://www.newtec.dk/webservice/NewtecDatabaseScheme#Folva13
ProductTime : 6124951413698769478
ProductBlackSpot : 0
ProductGreenSpot : 0
ProductVolume : 70
ProductDiameter : 44
ProductLength : 65

Individual: Warnez  ContactPerson : Mr. Warnez

7.2.2 RDFS Inference
Using Apache Jena with RDFS inference and the rules from the ontology will produce some new triples
highlighted. The rdfs:product is now visible along with the new units for the measurements.

Individual: Product1  type : http://www.newtec.dk/webservice/NewtecDatabaseScheme#Product
BatchName : http://www.newtec.dk/webservice/NewtecDatabaseScheme#Folva13
ProductTime : 6124951413698769478
ProductBlackSpot : 0
ProductBlackSpotUnit : http://www.newtec.dk/webservice/NewtecDatabaseScheme#Pixel
ProductGreenSpot : 0
ProductGreenSpotUnit : http://www.newtec.dk/webservice/NewtecDatabaseScheme#Pixel
ProductVolume : 70
ProductUnit : http://qudt.org/schema/qudt#Gram
ProductDiameter : 44
ProductLengthUnit : http://www.openphacts.org/units#Millimeter
ProductLength : 65
ProductLengthUnit : http://www.openphacts.org/units#Millimeter
ID : 1
7.2.3 OWL Inference

Using Apache Jena with OWL inference will produce new triples highlighted. The type owl:thing, sameAs and type rdf-schema#resource are all logical deductions from owl\textsuperscript{[4]}. The inference rule to have the ownership of the machine and customer be represented in bidirectional can be seen in the Customer as Owns. The rule is explained in chapter 6.
8 Data Extraction and SPARQL

The semantic web is not very useful unless the data can be extracted and used. The data can of course be parsed by well known technologies like Xquery for XML. The Xquery language has a number of problems in regards to RDF. One of the problems are the lack of order of the triples in RDF documents which is needed by the Xquery. The RDF document syntax can be either verbose or abbreviated, which again makes it hard for Xquery to have a precise syntax. None of these variations deviates from the RDF standard[21], nor from the recommended RDF modeling practice[22], but it means that well known XML technologies are hard to use[4].

There are several query languages for the semantic web such as RQS, RDQL and SPARQL. The report will use the SPARQL because it is W3C recommendation[23]. SPARQL is short for SPARQL Protocol and RDF Query Language and was specified by W3C. SPARQL is not a completely new query language, it borrows a lot of it constructs from SQL. This is not by chance since it is meant to be easy for SQL developers to utilize[14].

8.1 Newtec Data and the Batch Report

The batch report selected by Newtec see chapter 4.7 has to be selectable among the possible customers and machines. The web interface that allows the customer to select the right batch report has a number of options to limit the search for the right report. The web interface could of course offer all possible batch reports but this will quickly become unmanageable in a real solution with thousands of batch reports. The selection of a batch report can therefore be limited choosing a particular customer and/or a particular machine ID. This means that there will be a list of customers and a list of machines in accordance to the selected customers. The list of possible batch report is limited by the selection of the customer and/or machine, See table 8.1 below for the batch possibilities when all customers and all machines are selected.

<table>
<thead>
<tr>
<th>Customer</th>
<th>Machine</th>
<th>Batch Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>All</td>
<td>Folva 13</td>
</tr>
<tr>
<td>Warnz</td>
<td>A501</td>
<td>Folva 14</td>
</tr>
<tr>
<td>Lamb Weston</td>
<td>A502</td>
<td>Innovator 1A</td>
</tr>
<tr>
<td></td>
<td>A503</td>
<td></td>
</tr>
</tbody>
</table>

The default selection is all customers and all machines which results in all the batch reports. If a specific customer is selected the number of possible machines and batch reports are minimized. If a machine is selected the number of batch reports are again minimized. The possibilities in each list can be query in the RDF graph using SPARQL. This means that three queries are needed to populate the three lists “Customer”, “Machine” and “Batch Report” in table 8.1 described in section 8.3.1 – 8.3.3.

8.2 SPARQL Select Query

The SPARQL select and where query is one of the fundamental queries and is used for all three queries. The SELECT query has two parts, a set of question words and a question pattern. The WHERE keyword indicates the selection pattern. Below is a select query to find all the machines owned by the
customer Warnez.

`PREFIX nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#>
SELECT ?machine WHERE {nds:Warnez nds:owns ?machine}

<table>
<thead>
<tr>
<th>?machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>A501</td>
</tr>
<tr>
<td>A502</td>
</tr>
</tbody>
</table>

The query returns with “A501” and “A502” which is quite obvious when using the Newtec data from Appendix A. The deeper understanding of how this works it can be seen that “?machine” is the question that needs to be answered. The name of the question in this case machine is irrelevant to the SPARQL but it is advised to name the questions so it is human readable. Using more human readable questions also makes it easier to understand. This will also help if the query has to be visualized. The query can be displayed as a triple as shown in figure 8.1 below.

Figure 8.1 – RDF Graph, SPARQL Query
Visualization of a query. This query is a select-where query to find the machines owned by the customer Warnez.

8.2.1 Customer Select SPARQL Query
To populate the customer list a query that can retrieve all possible customer names are needed. Figure 8.2 shows the RDF graph representation of the Newtec data snippet and a SPARQL query can be constructed that extracts all the customers with the following query.

`PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#>
SELECT ?customerNames WHERE {
  ?customer rdf:type Customer .
}

<table>
<thead>
<tr>
<th>?customerNames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warnez</td>
</tr>
<tr>
<td>Lamb Weston</td>
</tr>
</tbody>
</table>

Figure 8.2 – RDF Graph, Customer SPARQL Select Query
Query to select the customers to populate table 8.1.
8.2.2 Machine Select SPARQL Query
To populate the machine list a query that depends on the selection of the customers are needed. The customer selection splits the query into two possibilities either all customers are selected or a particular customer is selected.

All customers:

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#>

SELECT ?machineID WHERE {
}

<table>
<thead>
<tr>
<th>?machineID</th>
</tr>
</thead>
<tbody>
<tr>
<td>A501</td>
</tr>
<tr>
<td>A502</td>
</tr>
<tr>
<td>A503</td>
</tr>
</tbody>
</table>

Particular customer (customerName=Warnez):

PREFIX nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#>

SELECT ?machineID WHERE {
  ?customer nds:customerName customerName .
}

<table>
<thead>
<tr>
<th>?machineID</th>
</tr>
</thead>
<tbody>
<tr>
<td>A501</td>
</tr>
<tr>
<td>A502</td>
</tr>
</tbody>
</table>
8.2.3 Batch Report Select SPARQL Query

The population of the machine list a query that depends on the selection of the machines are needed. The customer selection splits the query into two possibilities either all machines are selected or a particular machine is selected.

All machines:

```sparql
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#>

SELECT ?batchNames WHERE {
  ?batch rdf:type Batch .
}
```

<table>
<thead>
<tr>
<th>?batchNames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Folva 13</td>
</tr>
<tr>
<td>Folva 14</td>
</tr>
<tr>
<td>Innovator 1A</td>
</tr>
</tbody>
</table>
9 Web 2.0 Solution
The web solution is split into two parts, first part is the development of a Web 2.0 application that in the second part can be enriched with semantics. The first part is explained in this chapter containing the Web 2.0 solution. To host the web solutions a web server is needed. A Tomcat Web Server has been chosen for this purpose. The Tomcat Web Server is easy to launch and use. The server-part of the Apache Tomcat is written in Java. A Java model is therefore developed to hold all the information and the logic to service the Web client. The client side is written in HTML, JavaScript and AJAX calls. The following sections will describe each part of the web application.

9.1 Tomcat Installation
The tomcat web server 8.0 is chosen to host the web solution. The software is downloaded from Apaches web site and installed on a Linux Ubuntu 13.01. See appendix B for the installation and launch of the Tomcat server.

9.2 Newtec Web Application
The web application deployed by Tomcat consist of a number of files. The Tomcat files and installation are described in appendix B. The files specific to this web application is described below.

9.2.1 web.xml
The web.xml file is the Web Application Deployment Descriptor of the application. This file is an XML document that defines everything about the application that a server needs to know. The web.xml file for this project has 3 entries a listener, a servlet and a welcome page. The listener is represented in a class BootEarlyContextListener.java which the name indicates has the task of instantiating the server side, see section below. The servlet is the client sides bridge to the model and is represented in a class NewtecServlet.java, see section below. The last entry in the file is a welcome file and as the entry hints this is the first page handed out when the web server is called. This entry is done with the following lines.

```xml
<welcome-file-list>
  <welcome-file>MainMenuPanel.html</welcome-file>
</welcome-file-list>
```

9.2.1.1 BootEarlyContextListener
The Tomcat Web Server deploys the web application. A context listener is added to instantiate the Java model upon the web server startup. The web.xml file contains an entry for the listener with the following lines.

```xml
<!-- On webappplication start this listener boots the model -->
<listener>
  <listener-class>BootEarlyContextListener</listener-class>
</listener>
```

The contextInitialized method in the BootEarlyContextListener.java is responsible for the instantiation of the Java model with the following line of code.
public void contextInitialized(ServletContextEvent sce) {
    NewtecSemanticWeb.getNewtecSemanticWeb();
}

### 9.3.1.2 NewtecServlet

The web application server side needs to communicate with the client side that is handled with a servlet. The `NewtecServlet.Java` handles all data communication between the clients and the server. The web.xml file has an entry for the servlet with the following lines of code.

```xml
<servlet>
    <servlet-name>servlet_newtec</servlet-name>
    <servlet-class>NewtecServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>servlet_newtec</servlet-name>
    <url-pattern>/execute</url-pattern>
</servlet-mapping>
```

The first part identifies the a servlet and gives it a name in this case `servlet_newtec` and a class that represents the servlet in this case the `NewtecServlet.java`. The second part maps the servlet to the web application. First is an identifier and second the URL `/execute` to use when the client side want to communicate with the server side.

### 9.3 Java Model – Web 2.0

The server side of the web application is written in Java. The model contains the Newtec Data which is saved in four files in accordance to the small sample data described in section 4.4. The server side can be split into two parts, the Java model which contains the data and the logic to access it and the web application server side which services the client requests. The following components are used to construct a basic Java model. See figure 9.1 below.
A few classes represents the Java model. The placeholder of the model is the Newtec semantic web class and is named `NewtecSemanticWeb.java`. This class is responsible for loading the information from the files and create the number of customers, machines, batches and products classes as needed, these classes are named `Customer.java`, `Machine.java`, `Batch.java` and `Product.java`. The classes are instantiated when the Newtec data is read from the files containing the JSON format of the Newtec data. The last class is the `QueryEngine.java` which has the task of retrieving information needed by the NewtecServlet.

### 9.4 Web Application Client Side

The client side is written in HTML, JavaScript and AJAX calls to retrieve data from the server. The idea is to keep the client side as simple as possible without using the variety of web technologies available. The chosen technologies are all W3C standards and are the cornerstones for web development\(^{24}\). Since 4. quarter of 2014 the W3C recommends to use HTML5 for web development and the client side will be developed using HTML5.

One type of report has been chosen to represent the Newtec Data from appendix A. The batch report chosen by Newtec has its own separate page, which leads to a design with a main menu page and a batch report page. The generation of the actual reports happens when the correct customer or time has been chosen. The sketch for the design can be seen below in figure 9.2.

---

**Figure 9.1 – Overview Diagram of Basic Java Model**

The basic Newtec components for the web server interface.

---

**Figure 9.2 – Sketch Of the Web Application**

Hand sketch of web pages needed to develop the Newtec Semantic Web application.
Each page or panel is divided into 2 areas top and middle. The top area works as a header that is consistent for all panels. The top area will have the Newtec logo, the panel description and the version. The middle area is the actual context of the panel. Below is HTML outline code for the 3 panels.

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <link type="text/css" rel="stylesheet" href="newtec.css">
  <title>NEWTEC SEMANTIC WEB</title>
</head>
<body>
<! TOP P ANEL CODE>
<header id="topPanel">
  <div id="topPanelLeft" class="InsideContent">
    <img id="topPanelLeftImage" src="images/NewtecSemanticWeb.png" alt="NewtecSemanticWEB">
  </div>
  <div id="topPanelMiddle" class="InsideContent">
    <h1>Main Menu Panel</h1>
  </div>
  <div id="topPanelRight" class="InsideContent">
    <h1>Version 15.04.11</h1>
  </div>
</header>

<! MIDDLE P ANEL CODE>
<section id="middlePanel">
  <div id="middlePanelLeft" class="InsideContent">
    <!-- INSERT P ANEL SPECIFIC CODE HERE -->
  </div>
  <div id="middlePanelMiddle" class="InsideContent">
    <!-- INSERT JAVASCRIPT HERE -->
  </div>
</section>
<script type="text/javascript">
 <!-- INSERT JAVASCRIPT HERE -->
</script>
</body>
</html>
```

The HTML code above works as a skeleton for the two panels. A css style sheet has been added to allow styling. The style sheet is named newtec.css and the reference can be seen in the head section of the HTML code above.

### 9.4.1 Main Menu Panel

The main menu panel works as navigation to the batch report panel. It therefore has a button in the middle panel, see figure 9.3 below for a screen shot of the main menu panel. The button links to the batch panel.

![Main Menu Panel](image)

**Figure 9.3 – Main Menu Panel**
The Main Menu Panel of the Newtec Web 2.0 solution.
9.4.2 Batch Report Panel

The batch report panel shows batch reports generated from the products and batch information loaded upon startup of the Java model, see appendix A for the complete list of products and batch information. The panel has 3 selection lists. The first list is a selection of possible customers or all customers. If a customer is selected it will narrow down the possibilities of machines and batch reports in the second and third selection lists. The second list is a selection of possible machines or all machines and it will also narrow the possible batch reports in the third list. The third and last list is the possible batch reports that can be shown. They are represented by their name and the number of selectable batch reports depend on the selection of the first and second selection lists. Figure 9.4 shows a batch report panel screen shot.

![Batch Report Panel](image)

The button at the bottom links back to the main menu panel. When a batch name is clicked in the third selection list the batch report appears in front of the selection lists and the background is faded. The effect is done using a number of “div” elements that can be positioned in front of each other, see section 9.4.2.1 for further details. The effect can be seen in figure 9.5 below. The batch report consist of the element specified by Newtec see chapter 4.7 for more details on the reports.
9.4.2.1 Glass and Pop Up Deck

The HTML code to create the background fade and the batch report appearance, which in this application is referred to as the “glass panel” and the “pop-up panel”. The table element is tagged with an id called “batchNameListID” and a JavaScript function is added when a click event occurs. The popup deck has a table of batch report elements and a button to return to the batch report panel. The button calls a JavaScript function called `showGlassDeckAndPopupDeck()`. See HTML and JavaScript code below.

```html
<div id="mainDeck">
  <!-- CONTENT OF PANEL -->
  <tr>
    <td>
      <ul id="batchNameListID" class="customerMachineBatchList">
      </ul>
    </td>
  </tr>
</div>
<div id="glassDeck"></div>
<div id="popupDeck">
  <table id="batchReportTable">
    <!-- TABLE OF BATCH REPORT ITEMS -->
  </table>
  <button id="batchReportDoneButton" type="button" onclick="showGlassDeckAndPopupDeck(false)">Done</button>
</div>
```

JavaScript is added to create the effect. The first JavaScript code adds a click handler to the list elements in the batch selection list. When an item from the list is clicked the data is fetched from the server using a AJAX call. When the AJAX call has returned successfully `showGlassDeckAndPopupDeck()` function is called. This function sets the display element of the class deck and the popup deck to “inline”. The CSS style positions them on top of the main deck. When the batch report “Done” button is clicked the display element is set back to “none”. See JavaScript code below.

```javascript
var batchNameListIDLList = document.getElementById('batchNameListID');
```

Figure 9.5 – Batch Report

The batch report of the Newtec Web 2.0 solution.
The elements need a bit of styling to allow the behavior of a pop up panel with a glass pane.

9.4.2.2 Batch Report Panel Ready State
When the batch report panel is started a JavaScript ready state function is called. It uses a AJAX call to retrieve data from the server to populate the panel. The data needed upon startup is customer, machine and batch names. When the call succeeds a number of table items are generated. The request from the client is handled in the Web server. The request is data extraction of the information searching through the instances of the classes “Customer”, “Machine” and “Batch”. There is no central storage like a database where queries can be performed. See figure 9.6 below for a sequence diagram of the “batch name event”
The JavaScript code below.

```javascript
document.onreadystatechange = function()
{
    var xmlhttp;
    var url = "/newtecSemanticWeb/execute?requestCommand=batchReportTableData"
    if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
        var customerNameList = document.getElementById("customerListID");
        var machineIDList = document.getElementById("machineIDListID");
        var batchNameList = document.getElementById("batchNameListID");
        data = JSON.parse(xmlhttp.responseText);
        for(var i in data.batchReportTableData) {
            for(var j in data.batchReportTableData[i].customers) {
                var li = document.createElement("li");
                var liText = document.createTextNode(data.batchReportTableData[i].customers[j]);
                li.appendChild(liText);
                customerNameList.appendChild(li);
            }
            for(var k in data.batchReportTableData[i].machineID) {
                var li = document.createElement("li");
                var liText = document.createTextNode(data.batchReportTableData[i].machineID[k]);
                li.appendChild(liText);
                machineIDList.appendChild(li);
            }
            for(var l in data.batchReportTableData[i].batchNames) {
                var li = document.createElement("li");
                var liText = document.createTextNode(data.batchReportTableData[i].batchNames[l]);
                li.appendChild(liText);
                batchNameListID.appendChild(li);
            }
        }
    }
}
```

9.5 HTML Validation Service

The HTML pages are run through the W3C HTML validation service to ensure correct HTML markup. The HTML validation service should ensure that the HTML produced is conforming to W3C recommendation[^25]. The two HTML pages were both checked and validated using the Web page “https://validator.w3.org/”.

9.6 Web 2.0 - File Organization

The Newtec Web files are organized inside the Apache Tomcat folder. Appendix G have a complete file structure of the Web application.

[^25]: W3C HTML Validation Service
10 Semantic Web Solution

The Newtec semantic data is now ready to be put to use. Chapter 5 transformed the raw Newtec data into RDF triples. Chapter 6 and 7 enriched the RDF triples with an ontology and inference. Chapter 8 constructed the data extraction with SPARQL. The following chapters will put all this knowledge into use with the Apache Jena framework and enrich the existing web client developed in chapter 9. Exposing the Newtec semantic data on a web page will allow for anyone to extract and use the information. Web crawlers and Web scrapers can extract this information from the Web page and make use of it by semantically understanding the data.

10.1 Newtec Semantic Web - Server Side

The first part of the chapter will cover the changes on the server side. The Java model and the query engine will need changes to accommodate the semantic web solution.

10.1.1 Modeling Semantic Web in Java with Apache Jena

Converting the Newtec RDF into Java a software tool for the purpose is chosen. Apache Jena[26] is one of those software tools written in Java and looks like a very used library in the Java community. It is also the framework used in various text books[4]. First step is to download apache Jena and include it into the Java project. Apache Jena comes with a number of jars ready to use. See appendix C for downloading and integrating it into Eclipse. Table 10.1 below shows the tools and versions used for Java, Eclipse and Apache Jena.

<table>
<thead>
<tr>
<th>Software</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse</td>
<td>4.3 - Kepler Service Release 2</td>
</tr>
<tr>
<td>Java</td>
<td>Sun JRE 1.7</td>
</tr>
<tr>
<td>Apache Jena</td>
<td>2.13.0</td>
</tr>
</tbody>
</table>

10.1.2 Apache Jena and the Newtec Data

The Newtec data described in chapter 4 are available in appendix A is already in the web solution from chapter 9. The Newtec data is read from files containing JSON objects. The Web 2.0 solutions Java model was populated with new instances of the classes “Customer”, “Machine”, “Batch” and “Product” when the JSON objects were deserialized. These classes are no longer needed since the data will flow into a RDF store that the Apache Jena framework offers. A new class “RDFStore.java” is therefore introduced that will hold the Newtec data and will be queried from the “QueryEngine” class. Figure 10.1 below shows the new software component diagram.
The Apache Jena terminology\cite{27} is a bit different than the one used by W3C RDF specification\cite{7}. The RDF specification has subject, predicate and object to represent triples, see chapter 5 for further detail. The Apache Jena framework uses resources and properties. The resources represent subject and objects. The properties represent predicates. The new RDF store will make use of the Apache Jena framework and the code snippet below shows the creation of a new RDF model and the insertion of the “Folva 13” batch with a few properties from the Newtec data. All imports and validation code is omitted.

```java
Model RDFModel;
Resource batch;
String ndsURI = "http://www.newtec.dk/webservice/NewtecDatabaseScheme#"; //Newtec URI

RDFModel = ModelFactory.createDefaultModel();
batch = RDFModel.createResource(ndsURI + "Folva13"); //Create a new batch
resourceBatch.addProperty(RDF.type, "Batch"); //Adding rdf:type
resourceBatch.addProperty(ndsURI + "BatchName", "Folva 13"); //Adding batch name
resourceBatch.addProperty(ndsURI + "StartTime", "2008-09-21T22:27:00+1.00"); //Adding start time
resourceBatch.addProperty(ndsURI + "EndTime", "2008-09-21T22:27:01+1.00"); //Adding end time

The Newtec data from appendix A can all be added to model in a similar way when read a parsed by the JSON interface. The extra triples identified in chapter 6 are also inserted, eg. batchStarted.

10.1.3 Apache Jena and the Newtec Ontology

The inference from the Newtec ontology is performed using the Newtec ontology in Turtle format created in chapter 6 and the full ontology can be seen in appendix F. Inference is described in chapter 7 and below is how the inference is introduced to the web application.

```java
Model schema;
OntModel model;
Model RDFModel;
Reasoner reasoner;
InfModel infmodel;

RDFModel = ModelFactory.createDefaultModel(); //See section 10.2
schema = FileManager.get().loadModel("newtec_owl.ttl"); //Load Newtec ontology
reasoner = ReasonerRegistry.getOWLReasoner(); //Create reasoner
reasoner = reasoner.bindSchema(schema); //Bind ontology with reasoner
infmodel = ModelFactory.createInfModel(reasoner, data); //Perform inference
model = ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM, infmodel); //Create new model
```

49
The above code will create all the triples that will be available to the query engine.

### 10.1.4 Apache Jena and the Newtec Semantic Web SPARQL Queries

The Web 2.0 solution has a number of function calls to retrieve the data from the query engine doing a requests from the Web clients. The Web 2.0 solution was looking into the class instances to find the information needed, see chapter 9.4.2 for the batch report panel event. The SPARQL queries identified in chapter 8 can be used with Apache Jena which also supports SPARQL. The queries identified in chapter 8 can be used to extract the information needed to service the client requests. The customer select SPARQL query from chapter 8 is shown below implemented in Java using the Apache Jena framework.

```java
Model RDFModel;
String queryString;
QuerySolution row;
RDFNode rdfNode;
ArrayList<String> result;
result = new ArrayList<String>(); //Create array list for the result
RDFModel = ModelFactory.createDefaultModel(); //See section 10.2
queryString = "prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>"+
"prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"+"+
"prefix nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#>"+"
"select ?customerNames "+
"where {"+
"?customer rdf:type "Customer" ."+
"?customer <http://www.newtec.dk/webservice/NewtecDatabaseScheme#customerName> ?customerNames ."+
"} }";
query = QueryFactory.create(queryString); //Create query
queryExecution = QueryExecutionFactory.create(query, rdfStoreModel); //Execute query
resultSet = queryExecution.execSelect(); //Get the result set
while(resultSet.hasNext()) {
    row = resultSet.next(); //Loop through the result set
    rdfNode = row.get("customerNames"); //Get next object
    result.add(rdfNode.toString()); //Add to result list
}
queryExecution.close(); //Close query
```

### 10.2 RDF and HTML

The Apache Jena framework made it easy to transform the data from the Newtec grading machines into triples and introduce inference. It also allowed for queries using the SPARQL interface. This is however only means to the end. One of the goals are to enrich the HTML page with machine readable content using RDF. This section covers how to integrate RDF into the HTML page and how the Newtec specific RDF triples are inserted into the HTML.

#### 10.2.1 Embedding RDF Data into HTML Pages

There are number of ways of sharing the information of the RDF triples in HTML pages. The most widely described are microformats[28], eRDF[4] and RDFa[4][29].

50
10.2.1.1 Microformats
Microformats are really just XML tags incorporated into XHTML web pages and supported the declarative expression of semantics. The fundamental components are shown in figure 10.2.

![Figure 10.2 - Microformat](image)

The idea behind microformats are to enrich existing web pages with attributes such as “class” or “rel” which helps both humans and computer agents to determine semantics that otherwise aren't apparent. An easy example from the Newtec data is the markup of machine information. The highlighted text is the extra information added for semantics.

```html
<h1>Machine list</h1>
<div class="machine">
  <div class="machineID">A501</div>
  <span class="customer">Warnez</span>
</div>
```

The structured information in the above example is enabled by a standard vocabulary that is developed for commonly used items. This is usually achieved by embedding it in HTML as special tag attributes that has no impact on the how the browser displays a page. The first uses of the microformat was for business cards with a defined vocabulary that included names, positions, location, start time etc. The defined vocabulary was in vCard specification RFC 2426. One limitation of microformats is the need to specify a controlled vocabulary and provide a parser that can process that vocabulary. It seems to be better to only have one single syntax for marking up HTML pages with RDF. Then there would only be a single processing script for all microformats.

Microformats lower the barrier to publishing data on the Web. This is entirely in line with the high level goals of the Semantic Web but one of the microformats principles is humans first, machines second. Which of course comes from the design goal to enrich existing web pages. Another problem is the lack of URI which makes it hard to uniquely identify resources.

10.2.1.2 eRDF
eRDF is short for embedded RDF and is a subset of RDF standard that works well in XHTML and HTML. The subset means that not all the RDF recommendation is supported. The unsupported items include blank nodes, containers Bag, Seq and Alt, implicit typing such as RDFS sub classing, typed literals and arbitrary statements. The basics of eRDF can be divided into 3 steps. The first step is declare that the web page contains eRDF content using the profile attribute in the head element.

```html
<html>
  <head profile="http://purl.org/NET/erdf/profile">
    ...
  </head>
</html>
```
The profile attribute is an explicit declaration that the web page has some amount of data described in eRDF which also allows for executable transformations like XSLT to be applied to the web page. The second step is to denote namespaces using the link element and the relationship attribute.

```html
<html>
  <head profile="http://purl.org/NET/erdf/profile">
    <link rel="schema.rdf" href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <link rel="schema.rdf" href="http://xmlns.com/foaf/0.1#">
  </head>
</html>
```

The third step is the creation of the statements. Below is an example.

```html
<div id="matt" class="-foaf-Person">Matt used to work with
  <span class="-foaf-person foaf-knows" id="john">John</span> on site
</div>
```

eRDF had its main development activities in 2005 and 2006. There is no type of auto generator available online but there are services available to extract RDF from eRDF documents.

### 10.2.1.3 RDFa
RDFa is a W3C recommendation[29] that permits embedded semantics in existing HTML pages. The idea is enrich web pages much like microformats using the HTML elements like hyperlinks and layout tags such as “div” and “span”. Looking at the timeline RDFa evolved after microformats and it is an attempt to use the best of microformats such as existing web page enhancement and removing the need of heavy semantic web knowledge via easy syntax. Yet it tries to overcome some microformats visible issues such as unscoped vocabularies as well as required community standardization stovepipes. RDFa is inter-operable with the RDF W3C recommendations[7], supporting XML namespaces. It also supports distributes knowledge management which is in perfect line with the semantic web. This leads to developers creating their own vocabularies, taxonomies and ontologies which RDFa enables these to be published without restriction. The basics of the RDFa is described quite well in the RDFa recommendation[30].

In RDFa, a subject IRI reference is generally indicated using @about and predicates are represented using one of @property, @rel, or @rev. Objects which are IRI references are represented using @resource, @src, or @href, whilst objects that are literals are represented either with @content or the content of the element in question (with an optional datatype expressed using @datatype, and an optional language expressed using a Host Language-defined mechanism such as @xml:lang).

### 10.2.2 Embedding Newtec RDF into HTML
From the brief description of the possible frameworks to use above. It is clear that the RDFa is the one for this project. It covers all the features of the RDF as recommended by W3C and it is also a recommendation to use for embedding RDF into HTML by W3C. The support for namespaces and ontology makes it an easy choice. This section will embed the RDF triples into the HTML documents constructed in chapter 5. the RDFa specification[29] is quite extensive and only a subset is needed to cover the need for implementing the Newtec RDF triples into the Web page. W3C has also published a
RDFa primer\cite{31} to get started with the RDFa, which is helpful to embed the syntax of the RDFa into the Web page. The RDFa primer will be used for implementation of the RDF triples. Figure 10.3 shows the “Batch Report Panel” and the source code from the highlighted list element will be used for further examination.

The raw HTML code cannot be viewed directly in the Chrome browser. The Chrome browser does have a source code viewer. Most of the Web page is made up from JavaScrupts and it can not be viewed. An application is used for this called “Quick Source Viewer” see Appendix H for installation and usage of this application for the Chrome browser. Below is the encircled list element from figure 10.3 from the Newtec Semantic Web “Batch Report Panel” using the “Quick Source Viewer”.

\[
\begin{align*}
\text{<li vocab="http://www.newtec.dk/webservice/NewtecDatabaseScheme/" resource="LambWeston">} \\
\text{<span property="customerName">Lamb Weston</span>} \\
\text{<span property="rdf:typeof" content="Customer"></span>} \\
\text{<span property="Owns" content="A503"></span>} \\
\end{align*}
\]

The snippet shows how RDF triples are embedded into the HTML. The snippet is from the list of customers that the user can choose from, see figure 10.3. RDFa specification uses resources for subjects and the first line has the customer resource “LambWeston” which is a customer resource in the RDF store, see chapter 5. The first line also has “vocab” attribute that is used as a prefix. This means that each resource does not have to prefixed explicitly. The resource is identified with its URI “LambWeston” + the prefix from the “vocab” attribute. The content between HTML tags are the object of the triple and is a value. The second line have the content “Lamb Weston” which is the visible part to the human. A “property“ attribute is used to add the semantics that the content is customer name with the URI "CustomerName" + the prefix from the “vocab” attribute declared in the first line. The third line also has a HTML “span” element but with no content between the HTML tags. This is how extra semantics can be embedded without blurring the human presentation. The third line uses the property attribute for rdf:type which is the predicate and the content attribute for “Customer” which is the object. This corresponds to the semantics identified in chapter 7. The forth line is also a HTML “span” element again with no contents between the HTML tags. This line embeds the fact that this customer “Owns” machine “A503”. This semantics corresponds with the owl rule defined in chapter 7.

\text{Figure 10.3}

Snapshot of the batch report panel. The encircled customer name represents a list element in HTML. The element to a human looks normal but contains semantics hidden in the HTML code.
The semantics incorporated can be viewed using a RDF destiller and parser. There are a number of destillers and parsers available on the WWW. The report will make use of two, one from W3C which can transform the HTML into triples using the Turtle format. The second is one http://rdfa.info which can transform the HTML into a graph and RDF/XML. The reason to use two is the ability to show both the RDF graph and the Turtle format which is human readable friendly. See Appendix I for the use of the RDF destiller and parser. Below is the output from the two RDF destiller and parsers in graph in figure 10.4 and Turtle format.

```@prefix ns1: <http://www.newtec.dk/webservice/NewtecDatabaseScheme/> .
@prefix ns2: <http://www.w3.org/ns/rdfa#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
<> ns2:usesV ocabulary ns1: .
<LambWeston> ns1:Owns "A503";
    ns1:customerName "Lamb Weston";
    rdf:typeof "Customer" .```

10.3 RDF information not present in the RDF store
The Newtec ontology and RDF data helps enrich the Web page but there are still semantics that can be added. This section covers a small portion of the suggested semantics that the RDFa primer suggests.

10.3.1 RDFa and Dublin Core
The Dublin Core Metadata Initiative is a project whose goal is to develop metadata standards for a wide range of applications. The project maintains a registry of metadata terms that can be browsed or searched. The terms are primarily oriented toward simple and common descriptive metadata, include title, type, description, authorship and time stamp information. The two term that will be introduced are “http://purl.org/dc/terms/title” and http://purl.org/dc/terms/created. These two terms will be inserted at the top of the HTML “body” element like the HTML code snippet below shows.

```<body >
    <span property="http://purl.org/dc/terms/created" content="2015-05-25"/>
</body>```
10.4 Semantics from the The Newtec Semantic Web Solution
The semantics that can be extracted from the Batch Report Panel can be seen figure 10.5 below.

10.5 Semantic Web – File Organization
The Newtec Web files are organized inside the Apache Tomcat folder. See file structure in Appendix G.
11 Semantic Web and Direct RDF Store Access

Section 10 described how to implement the RDF triples into the existing web page. There could however also be a need to query the RDF store more directly. Queries that is not predefined by the developer of the web page but queries that can answer any questions. The web page uses queries to retrieve information from the RDF store but these are through the servlet using AJAX calls a more generic approach is wishful. There is a possibility to setup a query interface in the web page. This however limits the use for machine interaction and is cumbersome for implementation and the likelihood of reinventing a protocol to transfer the queries. Direct RDF store queries have already been addressed by W3C. W3C has two protocols Linked Data Protocol\[^{33}\] (LDF) and SPARQL Graph Store HTTP Protocol\[^{34}\] (GSP). The two protocols of interacting with the RDF triple store are not so different but originally served 2 purposes. In order to find the better choice for the Newtec implementation a short analysis of the 2 protocols are performed. Both protocols sit on top of the REST architecture so before any analysis of the two frameworks a short introduction to the REST API and the Newtec solution. The last part of the chapter is the actual implementation into the Newtec Semantic Web REST solution.

11.1 REST software architecture

Exposing the NEWTEC RDF store through the REST architecture\[^{35}\] has a number of benefits. The client and server side are separated. The server side is therefore not concerned with the user interface and can therefore often be simpler and more scalable. The client is also independent of the data storage which leads to better portability.

The perhaps best feature is the statelessness. The client and server communication is constrained by no client context being stored on the server between requests. Each request from any client contains all the information necessary to service the request, and session state is held in the client. The session state can be transferred by the server to another service such as a database to maintain a persistent state for a period and allow authentication. The uniform interface constraints are fundamental to the design of any REST service. The uniform interface simplifies and decouples the architecture, which enables each part to evolve independently which comes down to 4 constrains. Individual resources are identified in requests. The resources themselves are conceptually separate from the representations that are returned to the client which means that a response can be represented using HTML, XML or JSON, none of which the server needs to use as internal representation of the data. When a client holds a representation of a resource, it has enough information to modify or delete the resource. Each message includes enough information to describe how to process the message. Clients make state transitions only through actions that are dynamically identified within hypermedia by the server\[^{35}\]. The REST architecture is widely used on the WWW and is therefore a well known and proven technology. The many benefits listed above leads to a very simple implementation into the Newtec solution. A connectionless interface that requires stateless and complete information upon requests. It is build on top the already used HTTP protocol and fits directly into the Newtec solution with the use of a servlet. There is no need to be a aware of the client implementation which is of course not possible for Newtec to dictate or control.
11.2.1 SPARQL Graph Store HTTP Protocol (GSP)
The SPARQL Graph Store HTTP Protocol\textsuperscript{[35]}, or in short GSP is a W3C specification that was designed to handle RDF graphs. It is a protocol for data queering over HTTP in the RDF triple store. The protocol covers distributed updating and fetching of RDF graph content in a Graph Store through the mechanics of the HTTP and REST. The protocol makes use of underlying HTTP and RDF technology with following core constraints.

- Identification of resources through IRI.
- Manipulation of resources through representations of the RDF graph.
- Self-describing messages which is inherent characteristics of RDF as the framework for a Self-Describing Semantic Web\textsuperscript{[35]}.

The protocol specifies the semantics of HTTP operations for managing a Graph Store (RDF triple store). In particular, it provides operations for removing, creating, and replacing RDF graph content as well as for adding RDF statements to existing RDF graph content. The interface defined here uses IRIs to direct native HTTP operations to an implementation of this protocol which responds by making appropriate modifications to the underlying Graph Store. The protocol supports the HTTP methods GET, POST, PUT, DELETE, HEAD and PATCH. The protocol also covers the return codes of the requests in different situations\textsuperscript{[35]}. The protocol uses SPARQL to communicate with the server and the HTTP accept or content-type field indicates the format of the request and the preferred response format. There is no need to change anything on the Newtec web solution to make this protocol work.

11.2.2 Linked Data Platform (LDP)
The idea behind Linked Data Platform\textsuperscript{[33]} is to combine two Web related concepts REST and RDF to help solve some of the long standing challenges involved in building and combining software into a platform for information interaction. The combination of RDF and the REST API seems natural, with RDF providing a standard way to serialize information about things identified by URIs and REST providing a way to obtain and alter the state of those things. The basic technique is to expose RDF on the Web, allowing authorized clients to see and modify object state using HTTP operations with an RDF data format. The platform works with linked data platform resources and the actual linked data platform server is needed to respond to the requests. There are two kinds of resources which is divided into RDF resources and everything else such as images, HTML pages etc. LDP has a number of HTTP methods that it supports GET, POST, PUT, DELETE, HEAD, PATCH and OPTIONS. The approach to interact with the information relies on the principal that web pages have organized the content into containers. Many HTTP applications and sites have organizing concepts that partition the overall space of resources into smaller containers. Blog posts are grouped into blogs, wiki pages are grouped into wikis, and products are grouped into catalogs. Each resource created in the application or site is created within an instance of one of these container-like entities, and users can list the existing artifacts within one. The Newtec solution is also grouped into containers using HTML5 tags and the RDF adds another layer to this. The organization is not the only need. There is also an overhead with handling the resource requests.
11.2.3 Linked data Platform vs SPARQL Grapgh Store HTTP Protocol
The fact that W3C seems to recommend two different approaches a comparison is carried out[36]. The two protocols were developed at different times. The GSP was released before LDP and the two protocols also seems to address 2 different goals. GSP is targeted directly at the RDF resources while LDP targets all resource available on the web solution. There are therefore also two views of the resources. GSP has the advantage of only dealing with RDF resources and it naturally uses SPARQL as the backbone for queries which is also defined as triples. LDP needs a more elaborate frame work to access all available resources. The two protocols are however not so different when it comes accessing the information through HTTP methods. The biggest difference is clearly the view of resources and how they are accessed. In relation to the Newtec Semantic Web solution the GSP specification seems to be more the right way. The goal is to give access to the triple store. GSP also has the advantage of using SPARQL which is in line with the rest of the implementation. LDP needs a more elaborate server part to access the resources and GSP use SPARQL which is already part of the implementation using the JENA framework, see section 10.

11.3 Newtec Semantic Web GSP Implementation
The RDF triples available in the RDF store is exposed using the GPS protocol. The platform holds a lot of possibilities which will only be partly meet, see method implementation[33]. There are a number of HTTP methods using the GSP protocol[37]. The HTTP methods are described below for completeness of the specification and to explain the response codes from the servlet.

11.3.1 HTTP PUT
A request that uses the HTTP PUT method must store the enclosed RDF payload as RDF graph content. The initial HTTP request must be understood to have the same effect as the sequence of SPARQL Update operations. No implementation for this method and a request to the servlet will responded with a the status code 405 Method Not Allowed.

11.3.2 HTTP DELETE
A request that uses the HTTP DELETE method should delete the RDF graph content identified by either the request or encoded IRI. This method may be overridden by human intervention on the origin server. If there is no such RDF graph content in the Graph Store, the server must respond with a 404 Not Found response code. No implementation for this method and a request to the servlet will responded with a the status code 405 Method Not Allowed.

11.3.3 HTTP POST
A request that uses the HTTP POST method and a request IRI that identifies RDF graph content must be understood as a request that the origin server perform an RDF merge of the enclosed RDF payload enclosed into the RDF graph content identified by the request or encoded IRI. No implementation for this method and a request to the servlet will responded with a the status code 405 Method Not Allowed.

11.3.4 HTTP GET
The HTTP GET method should be used to retrieve a graph representation of the networked RDF knowledge identified by the Request-URI. This is equivalent to the following SPARQL query.
The semantics of the GET method change to a "conditional GET" depending on the existence of headers indicating constraints on the request. A conditional GET requests that the graph representation be transferred only under the circumstances described by the conditional header field(s). The conditional GET method is intended to reduce unnecessary network usage by allowing cached entities to be refreshed without requiring multiple requests or transferring data already held by the client. This is the only supported method and is explained with an example in section 8.

11.3.5 HTTP HEAD
The HTTP HEAD method is identical to GET except that the server must not return a message-body in the response. It is meant to be used for testing dereference able IRIs for validity, accessibility, and recent modification. No implementation for this method and a request to servlet will responded with a the status code 405 Method Not Allowed.

11.3.6 HTTP PATCH
The method can be used to request that a set of changes described in the request entity be applied to the named graph associated with the graph IRI of the RDF graph content resource identified by the request IRI. No implementation for this method and a request to servlet will responded with a the status code 405 Method Not Allowed.

11.3 Newtec Semantic Web GSP API
The implementation of GSP in the Newtec solution is carried out extending the Newtec web server. Only the HTTP method GET will implemented as part of the solution. The remaining of the HTTP methods will be answered with a HTTP error code 405 (Method Not Allowed) which is not in accordance with the GSP protocol.

11.3.1 GSP Servlet
A new servlet is created to serve as interface to the Newtec Semantic Web GSP API. The new servlet is added to the web.xml file with the following lines.

```xml
<url-pattern>/execute</url-pattern>
</servlet-mapping>

<url-pattern>/restapi</url-pattern>
</servlet-mapping>
```

The new servlet is created in the Java server application using the file name NewtecServletRESTAPI.java. The new class is responsible for the requests made by clients. The SPARQL queries are passed to the query engine class which handles the actual query.
11.3.1.1 Newtec Customer Query

The query that is used to test the Newtec GSP API is a query to extract customer names. The query can be seen below.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#>

SELECT ?customerNames WHERE {
}
```

See section 8 for further detail on the query and how the turtle format is constructed. The Apache Jena framework uses the query in turtle format to execute the query. See section 10 for further detail on the Apache Jena framework.

The cumbersome task of expressing the SPARQL query in HTML URL encoding, which uses the ASCII encoding, requires a bit of encoding knowledge. The ASCII encoding needs to replace unsafe characters with “%” followed by two hexadecimal digits. Below in table 11.1 is a list of the special characters needed to transform the SPARQL query into HTML URL encoding that can be used in the Advanced REST Client Application.

<table>
<thead>
<tr>
<th>Character</th>
<th>UTF-8</th>
<th>Character</th>
<th>UTF-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
<td>%3A</td>
<td>space</td>
<td>%20</td>
</tr>
<tr>
<td>&lt;</td>
<td>%3C</td>
<td>?</td>
<td>%3F</td>
</tr>
<tr>
<td>/</td>
<td>%2F</td>
<td>{</td>
<td>%7B</td>
</tr>
<tr>
<td>&gt;</td>
<td>%3E</td>
<td>}</td>
<td>%7D</td>
</tr>
<tr>
<td>newline</td>
<td>%0D</td>
<td>#</td>
<td>%23</td>
</tr>
<tr>
<td>carriage return</td>
<td>%0A</td>
<td>“</td>
<td>%22</td>
</tr>
</tbody>
</table>

The special character encoding can now be used to make the actual transformation of the SPARQL query in turtle format. Table 11.2 shows each line transformed.

<table>
<thead>
<tr>
<th>SPARQL – Turtle Format</th>
<th>SPARQL – URL Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFIX rdf: <a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a></td>
<td>PREFIX+rdf:+%3Chttp%3A%2F%2Fwww.w3.org/1999/02/22-rdf-syntax-ns%23%3E%0D%0A</td>
</tr>
<tr>
<td>PREFIX rdfs: <a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a></td>
<td>PREFIX+rdfs:+%3Chttp%3A%2F%2Fwww.w3.org/2000/01/rdf-schema%23%3E%0D%0A</td>
</tr>
<tr>
<td>PREFIX nds: <a href="http://www.newtec.dk/webservice/NewtecDatabaseScheme#">http://www.newtec.dk/webservice/NewtecDatabaseScheme#</a></td>
<td>PREFIX+nds%3A+%3Chttp%3A%2F%2Fwww.newtec.dk/webservice/NewtecDatabaseScheme%23%3E%0D%0A</td>
</tr>
<tr>
<td>SELECT ?customerNames WHERE {</td>
<td>SELECT+%3FcustomerNames+WHERE+%7B%0D%0A</td>
</tr>
<tr>
<td>?customer rdf:type &quot;Customer&quot; .</td>
<td>%3Fcustomer+rdf%3Atype+%22Customer%22+%0D%0A</td>
</tr>
</tbody>
</table>
11.3.1.2 Newtec GSP API Testing
An agent that can interact with the Newtec GSP REST API is needed to perform tests. Google Chrome however has a number of good application that can be used making a custom REST agent implementation irrelevant. The Google Chrome app “Advanced REST Client” is used for this purpose see appendix D for more information on download and installation of the application. The application will be used for testing and snapshots of the interface is used to show results of various HTTP requests. The SPARQL turtle query from section 10.3.1.1 can be copied directly into the Advanced REST client application and used to query the Newtec GSP API.

11.3.1.3 GSP Accept Field
The HTTP get request should specify the preferred format in the accept field of the HTTP header. If the HTTP header does not specify the preferred format in the accept field the response will be formatted to rdf/xml. The protocol specifies that in the absence of the accept header field the implementation can try to guess the content of the request.

The implementation however will discard requests without the accept field with HTTP status code 406 (Not Acceptable). This HTTP status code will be accompanied by an accept field that shows the allowed field. In this case the response header will have an accept field with “rdf/xml”. See the HTTP get request below in figure 11.1.

![Figure 11.1 – Advantage REST Client, HTTP Code 406](image_url)

HTTP get request to the Newtec GSP API with an unsupported accept field. The HTTP response status code is 406 (Not Acceptable). The HTTP response header shows the allowed accept header format, rdf/xml.
11.3.1.4 GSP Malformed Query
If the HTTP get request has a malformed query or uses unsupported HTTP verb the response is a HTTP status code 405 (Method Not Allowed). See figure 11.2.

![Figure 11.2 – Advantage REST Client, HTTP Code 405](image)
HTTP get request to the Newtec GSP API with a malformed query or unsupported HTTP verb. The HTTP response status code is 405 (Method Not Allowed).

11.3.1.5 GSP Correct Query
If the HTTP get request is constructed correctly the HTTP response will contain a HTTP header with the content/type set to text/xml and the HTTP status code 200 (OK) to indicate the request was successful. See figure 11.3 below.

![Figure 11.3 – Advantage REST Client, HTTP Code 200](image)
HTTP get request to the Newtec GSP API with a correct content. The HTTP response status code is 200 (OK). The HTTP header content/type is set to text/xml. The response is a list of customer names in accordance with the query.

The above query entered into the Advanced REST Client application shows how queries can be tested to retrieve information from the RDF triple store in Apache Jena rdf/xml format.
12 Conclusion

The Newtec semantic web interface allows both humans and computer agents to retrieve information. The analysis of the semantic web lead to an understanding of the data and how it could be connected using RDF triples. RDF triples which is one of the cornerstones in the semantic web has the potential to give the data element semantics but also to connect it semantically to other data elements. The idea that all things in the world can be uniquely identified is the key to semantically give a better understanding of the content of the information available. The ontologies allows users to understand what the information is about. The use of ontologies from the wild allows for standard ways of defining properties such as a units or human properties.

The transformation from a Newtec Web 2.0 solution to a Newtec Semantic Web solution was straightforward at the client side since the only change was new HTML elements that was embedded with the existing HTML elements. The new HTML elements are not visible at the presentation layer in the browser for the human so there was no need for new styling. The server side had to be transformed from a class storage approach to a RDF triple store. The replacement of the class representation of the Newtec data to a RDF store was achieved with Apache Jena framework. The Apache Jena framework also enabled queries using the SPARQL interface which allowed a standard interface of querying data, instead of a self made querying interface. The embedded RDF triples can be retrieved by web scrapers and web crawlers to allow better semantic understanding of the content of the web pages. The analysis of the Semantic Web and embedding RDF into the web pages also revealed new data that was not present as raw data. The Newtec ontology meant new inferred RDF triples and the W3C RDF primer recommended RDF tagging of each page using resources such as the Dublin Core to allow better semantic of each web page. The raw data which was hand picked by Newtec was also enriched with units which lead to a semantic understanding of the numbers represented.

The Newtec REST interface that allows access to the triple store directly using the SPARQL Graph Store HTTP Protocol is a good solution for more specialized queries than the web pages present. The Web page is limited in the sense there is no more data than the Web developer make available. The Newtec reports are constructed from far more Newtec data than the key numbers that the Web page gives access to.

The Newtec Semantic Web solution needs a database that allowed far more data to be stored. The data that flows to the Newtec servers every day is vast, considering that more than a 1000 customers with more than 20000 machines have been sold. There are a number of databases available and the interesting databases are not relational database but graph databases suited for applications such as the Newtec Semantic Web. It would also be worth to have a look at the reports being represented since a lot of the reports are averages of the numbers form the products. New RDF triples could be constructed holding this kind of information for faster data retrieval also known from relational database design. It would allow for optimization of data stored or policies of how long data needs be stored. The Newtec ontology has been started and far more work could be put into this part. A better ontology would be the foundation of how data should be retrieved and how to understand the presented data. The ontologies in the wild are also managed by a number of frameworks that allow for description of each rule and also representation of multiple language.

The next obvious step is to launch the Newtec Semantic Web on the WWW. There is no technical
reason not to launch this approach and the embedded RDF enrichment can happen at the pace that the development allows for. The transformation from Web 2.0 to Semantic Web solution can consist of many small steps each bringing it closer to Tim Berner-Lee' dream.
13 References

[7] [http://www.w3.org/RDF]
[8] [http://en.wikipedia.org/wiki/Subject#Linguistics
[9] [http://en.wikipedia.org/wiki/Predicate_%28grammar%29
[10] [http://www.w3.org/TR/turtle
[11] [http://www.w3.org/TR/REC-rdf-syntax
[18] http://www.w3.org/TR/owl-time/
[22] http://www.w3.org/standards/techs/rdfbp#w3c_all
[23] http://www.w3.org/TR/rdf-sparql-query
[25] https://validator.w3.org/about.html
[26] https://jena.apache.org
[27] https://jena.apache.org/tutorials/rdf_api.html#ch-Introduction
[29] http://www.w3.org/TR/html-rdfa/
[31] http://www.w3.org/TR/xhtml-rdfa-primer/
[33] http://www.w3.org/TR/ldp/
[34] http://www.w3.org/TR/sparql11-http-rdf-update/
[38] http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.htm
[40] https://en.wikipedia.org/wiki/Ontology_(information_science)
Appendix A – Customer, Machine, Batch and Product JSON Packets

The 3 samples collected from the machines are given below in the raw format the machines deliver them to the web server. Every batch has an info packet and there after there is a series of packets of the products. The customer and machine information is also presented below.

A.1 Product Data

```json
{
    "packetType": "productListPacket",
    "machineID": "A501",
    "NoOfProducts": 2,
    "length": [65, 58],
    "diameter": [44, 35],
    "volume": [70, 40],
    "filter0": [0, 1635],
    "filter1": [0, 1635],
}
```

```json
{
    "packetType": "productListPacket",
    "machineID": "A502",
    "volume": [29],
    "NoOfProducts": 1,
    "diameter": [30],
    "length": [59],
    "filter0": [1822],
    "filter1": [1822],
    "time": ["2015-02-26 10:25:13"]
}
```

```json
{
    "packetType": "productListPacket",
    "machineID": "A503",
    "volume": [96],
    "NoOfProducts": 1,
    "diameter": [53],
    "length": [62],
    "filter0": [0],
    "filter1": [0],
    "time": ["2015-02-25 14:07:01"]
}
```

A.2 Batch Data

```json
{
    "packetType": "batchInfoPacket",
    "batchName": "Folva 14",
    "machineID": "A501",
}
```

```json
{
    "packetType": "batchInfoPacket",
    "batchName": "Folva 13",
    "machineID": "A501",
}
```

```json
{
    "packetType": "batchInfoPacket",
    "batchName": "Innovator 1A",
    "machineID": "A503",
}
```
A.3 Machine data

```
{
    "packetType": "machineInfoPacket",
    "machineID": "A501",
    "CustomerName": "Warnez"
}
```

```
{
    "packetType": "machineInfoPacket",
    "machineID": "A502",
    "CustomerName": "Warnez"
}
```

```
{
    "packetType": "machineInfoPacket",
    "machineID": "A503",
    "CustomerName": "Lamb Weston"
}
```

A.4 Customer Data

```
{
    "packetType": "customerInfoPacket",
    "CustomerName": "Warnez",
    "ContactPerson": "Mr Warnez"
}
```

```
{
    "packetType": "customerInfoPacket",
    "CustomerName": "Lamb Weston",
    "ContactPerson": "Mr Lamb Weston"
}
```
Appendix B – Setup of Tomcat In Linux Ubuntu 12.04
This appendix is how Apache Tomcat is installed. The last section is a how the Apache Tomcat is relaunched and the Java classes are compiled.

B.1 Version
Linux: Ubuntu 12.04 i386
Apache Tomcat: 6.0.18
JDK: 1.8.0_20

B.2 Prerequisites
Download Oracle Java JDK

Extract Java JDK
# mkdir ~/applications
# cd ~/applications/
# tar -xzvf (download folder)/jdk-8u40-linux-i586.tar.gz

Download Tomcat 8.20.tar.gz
http://apache.mizakim.net/tomcat/tomcat-8/v8.0.20/bin/apache-tomcat-8.0.20.tar.gz

B.3 Installation
Installing Tomcat from a binary release (tar file) requires manual creation of the Tomcat user account. For security reasons I created a user account with no login shell for running the Tomcat server.

# sudo groupadd tomcat
# sudo useradd -g tomcat -s /usr/sbin/nologin -m -d /home/tomcat tomcat

Extract the tar file to /var/lib and changed the ownership of all files and directories to tomcat

# cd (tomcat install folder)
# sudo tar zxvf (download folder)/apache-tomcat-8.0.20.tar.gz
# sudo chown -R tomcat.tomcat (tomcat install folder)/apache-tomcat-8.0.20

Check the Tomcat version

# /var/lib/apache-tomcat-6.0.18/bin/version.sh
Using CATALINA_BASE: /var/lib/apache-tomcat-8.0.20
Using CATALINA_HOME: /var/lib/apache-tomcat-8.0.20
Using CATALINA_TMPDIR: /var/lib/apache-tomcat-8.0.20/temp
Using JRE_HOME: /home/henrik/applications/jdk1.8.0_20
Using CLASSPATH: /var/lib/apache-tomcat-8.0.20/bin/bootstrap.jar:/var/lib/apache-tomcat-8.0.20/bin/tomcat-juli.jar
Server version: Apache Tomcat/8.0.20
Server built: Feb 15 2015 18:10:42 UTC
Server number: 8.0.20.0
OS Name: Linux
OS Version: 3.13.0-35-generic
Architecture: i386
For security reasons the Tomcat server does not run as user root but as tomcat which was created with no login shell. Therefore, to run Tomcat use the su command with the -p option to preserves all the environment variables when switching to tomcat. Since the tomcat account has no login shell, it needs to be specified with the -s option. Create tomcat start script file.

### B.4 Create Tomcat instance

Create a new tomcat instance. It helps porting the webapp around without deploying directly into the default tomcat folder. Creating a new base directory for a new instance requires the creation and copying of various directories and configuration files.

```
# mkdir -p $(install folder)/tomcat-instance/semantic.web.com
# cd $(install folder)/tomcat-instance/semantic.web.com
# cp -a $(tomcat install folder)/apache-tomcat-8.0.20/conf .
# mkdir common logs temp server shared webapps work
# chown -R tomcat.tomcat $(install folder)/tomcat-instance
```

Create an environment file for the new Tomcat instance. This will be useful for easily setting the environment variables when starting/stopping the new Tomcat instance.

```
# cat > $(install folder)/startTomcat.sh << EOF
export JAVA_HOME=/usr/java/jdk1.8.0_40
export PATH=\$JAVA_HOME/bin:\$PATH
export CATALINA_HOME=/$(tomcat install folder)/apache-tomcat-8.0.20
export CATALINA_BASE=/$(install folder)/semantic.web.com
\$CATALINA_HOME/bin/startup.sh
EOF

# cat > $(install folder)/stopTomcat.sh << EOF
\$CATALINA_HOME/bin/shutdown.sh
EOF
```

CATALINA_HOME is the base directory of Tomcat that contains all the libraries, scripts etc. for Tomcat. This is the parent directory of the extracted Tomcat tar file.

CATALINA_BASE is the base directory of the new Tomcat instance, which in this example points to /$(install folder)/tomcat-instance/sales.example.com.

### B.5 Setup web application for tomcat instance

In Tomcat web application root directories are created under $CATALINA_BASE/webapps by default. Create the webapp layout.

```
# mkdir $CATALINA_BASE/webapps/semantic_web
```

To configure Tomcat to recognize the new web application under $CATALINA_BASE/webapps/semantic_web
($CATALINA_BASE/semantic.web.com/webapps/sales), the $CATALINA_BASE/conf/server.xml file needs to be edited. The conf file sets the URL to reach the web application in this case "http://localhost/semanticWeb"

# nano $CATALINA_BASE/conf/server.xml

Find the following section

```xml
<Host name="localhost" appBase="webapps"
    unpackWARs="true" autoDeploy="true"
    xmlValidation="false" xmlNamespaceAware="false">
```

Add

```xml
    <Context docBase="semantic_web" path="/semanticWeb"/>
```

B.6 Home page and Java servlet application

The basics of the web server instance is done. An index and Java servlet needs to be added to get started. First an index HTML file is created.

# cat > $CATALINA_BASE/webapps/semantic_web/index.html << EOF
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><META http-equiv=Content-Type content="text/html"></head>
<body>
<H3>Apache Tomcat Index page</H3>
<a href="/semanticWeb/execute">Press to reach Java Servlet</a>
</body>
</html>
EOF

Create Java Servlet in the classes folder in the WEB-INF folder

# cat > $CATALINA_BASE/webapps/semantic_web/WEB-INF/classes/Sales.java << EOF
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Sales extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Java Servlet reached</title>");
        out.println("<head>");
        out.println("<h1>Executing semantic web</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
EOF
To compile the new Java servlet, the servlet-api.jar JAR file is needed which can be specified with either the -classpath option or the CLASSPATH environment variable. The -classpath option for SDK tools is preferred over the CLASSPATH environment variable since it can be set individually for each application without affecting others. In the following example I specify the path of the class directory with the basename "*" (if you are unfamiliar with basename, see 'man basename'). This is equivalent to specifying all files with the extensions .jar or .JAR files in the directory and therefore individual JAR files like servlet-api.jar don't need to be specified.

The following command should now compile the Java servlet.

```
# cd $CATALINA_BASE/webapps/semantic_web/WEB-INF/classes
# javac -classpath "$CATALINA_HOME/lib/*" Sales.java
```

To configure servlets and other components for an application, an XML file called web.xml needs to be configured.

```
# $CATALINA_BASE/conf/web.xml
# $CATALINA_BASE/webapps/{your-appname}/WEB-INF/web.xml
```

The first one is the default web.xml file which is the base for all web applications in a Tomcat JVM instance, and the latter one is for the web application where WEB-INF resides for overwriting application specific settings. Create a new web.xml file.

```
# cat > $CATALINA_BASE/webapps/semantic_web/WEB-INF/web.xml << EOF
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
version="2.5">
  <servlet>
    <servlet-name>servlet_semantic</servlet-name>
    <servlet-class>Semantic</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>servlet_semantic</servlet-name>
    <url-pattern>/execute</url-pattern>
  </servlet-mapping>
</web-app>
EOF
```

For each servlet there is a <servlet> element. It identifies the servlet name (<servlet-name>) and the Java class name (<servlet-class>). The servlet mapping (<servlet-mapping>) maps a URI to the servlet name (<servlet-name>). In the above example "/execute" in "http://localhost:8080/semanticWeb/execute" maps to "servlet_semantic" which points to the "Semantic" servlet class. Note that the order of these elements is important. So when you open the URL "http://localhost:8080/semanticWeb/execute", the "Semantic" Java servlet will be executed.

Start and test tomcat with the new instance. The following should come up when entering the following...
url into a browser.

http://localhost:8080/semanticWeb

B.7 Compiling and Apache Tomcat Relaunch Script

```
#!/bin/bash
export CATALINA_HOME=~/.master_project/tomcat/apache-tomcat-8.0.20
export CATALINA_BASE=~/.master_project/tomcat/tomcat_test/tomcat-instance/newtec.semanticweb.dk
javac -classpath "$CATALINA_HOME/lib/*" *.java
$CATALINA_HOME/bin/shutdown.sh
$CATALINA_HOME/bin/startup.sh
```
Appendix C – Jena and Eclipse Tutorial

The following section describes how Jena was added to Eclipse for quicker development purposes. The appendix shows how to create a “Hello World” example.

C.1 Version
Linux: Ubuntu 12.04 i386
JDK: 1.8.0_20
Apache Jena 2.13.0

C.2 Prerequisites

C.3 Create a Java Project
Eclipse organizes files into projects, so we need a project for this tutorial. Depending on the plugins installed, Eclipse may show a large number of different types of project. A plain Java project is fine for a basic Jena application. In the file menu choose new then Java Project. When the “New Java Project” dialog comes up name the project “JenaTutorial” and press the button “Finish”, see figure below.

Create a New Java Class File
Add a new Java class File to the Project. Unfold the new project and right click on the “src” folder. Under the “new” item choose the “class” item. A new dialog appears, enter the name “HelloRDFWorld” into the “Name” area and click the “Public static void main(...)” click box. Press the button “Finish”. See figure below.
C.4 Eclipse User Library

Before any code is entered it is necessary to add the libraries from the Apache Jena package. There are actually a few different ways of setting the build path in Eclipse. One way to do this is to create a lib directory in the project top-level folder, then copy the Jena “.jar” files there, and then link that directory to the project's build path. That works, but there's a better way: defining a user library. A user library is a declaration of a library (collection of supporting code) that can be referenced from any project. Once set up it can use the same library definition in multiple different projects. Moreover, if subsequently Jena is updated to a new release, then once the library is updated every project in Eclipse workspace will see the new version. With the copy-files-to-the-lib-folder method, it is necessary to re-copy to every project that uses Jena.

From the top bar choose “Window” then “preferences”. See figure below.
This will open a new dialog. Open “Java” in the select list and then open “Build Path” in the new select
list. Click the “User Libraries” which will open a new dialog. Enter “JenaLibs” as the name of the user
libs. See Figure below.

![New User Library Dialog](image1.png)

Select the new library “JenaLibs” and click the button “Add External JARs” A new dialog appears and
where the Apache Jena libs needs to located. When located select all and press the button “OK”. See
figure below.

![Add External JARs Dialog](image2.png)

The new user lib has been created, press the button “OK” to finish. Next the new user library needs to
be added to the Java project. Right click on the project “JenaTutorial” and select “Properties” a new
dialog appears. Select the “Java Build Path” in the select list. Press the button “Add Library”. In the
new dialog select the option “User Library” and select “JenaLibs”. See figure below. The “User Library” has now been added to the project.

C.5 Hello World RDF Java Code
All the prerequisites have been done. It is now time to add some code and see the effect. Enter the code as shown in figure below.
Run the code in Eclipse by clicking the Button “Run”. See figure below for result.

The same can be done from a console. The Apache Jena library has been copied next to the source code for convenience. See figure below.
Appendix D - Advanced REST Client

Testing a GSP REST API can be cumbersome with a custom REST client implementation and luckily a number of developers need to test REST APIs. It is therefore relative easy to find an application for Google Chrome that have these capabilities. One such application is called Advanced REST Client. The appendix runs through the procedure to install the application.

D.1 Version
Advanced Rest Client: 1.0.8
Google Chrome Version: 37.0.2062.120

D.2 Download and Installation of the Application
First step is to download the application by entering the following URL in the Google Chrome browser.

https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfcellkdfbfbjeloo

This will bring up the page below in figure D1 and by clicking the green button in the top right corner the application will be installed.

![Figure D1](image)

*Figure D1*
*Installation of the Google Chrome application Advanced REST Client.*

D.3 Launching the Application
To launch the application simply open the Google Chrome browser in an empty tab and click the app icon the top left corner to see the applications installed in Google Chrome. Next press the Advanced REST Client application to launch.
D.4 Using the Application
The usage of the application is simple in figure D2 a HTTP get request to the Newtec GPS API is shown.

![HTTP get request](image)

**Figure D2**
Launching the Google Chrome application Advanced REST Client.

**Figure D3**
Usage of the Google Chrome application Advanced REST Client. Example of a HTTP get request to the Newtec GSP API.
Appendix E – Newtec Data Transformation into Triples

Conversion of tabular Newtec data into triples. Table E.1 holds the name space and the qname of the name space. Table E.2-E.5 contains the customer, machine, batch and product data sample in tabular form. Only the ID of table E.2 is used the rest is converted using the semantics described in chapter 5.

<table>
<thead>
<tr>
<th>Table E.1 URI and qname for the Newtec database</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
</tr>
<tr>
<td><a href="http://www.newtec.dk/webservice/NewtecDatabaseScheme#">http://www.newtec.dk/webservice/NewtecDatabaseScheme#</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table E.2 Tabular Data Of Newtec Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>Products</td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table E.3 Tabular Data Of Newtec Machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machines</td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table E.4 Tabular Data Of Newtec Batches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batches</td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table E.5 Tabular Data Of Newtec Customers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customers</td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table E.6 Concatenation of Newtec name space and tabular data of products</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name space</td>
</tr>
<tr>
<td>nds (Table E.1)</td>
</tr>
<tr>
<td>nds (Table E.1)</td>
</tr>
<tr>
<td>nds (Table E.1)</td>
</tr>
<tr>
<td>nds (Table E.1)</td>
</tr>
<tr>
<td>nds (Table E.1)</td>
</tr>
<tr>
<td>nds (Table E.1)</td>
</tr>
<tr>
<td>nds (Table E.1)</td>
</tr>
<tr>
<td>nds (Table E.1)</td>
</tr>
</tbody>
</table>
## Table E.7 Triples representing the data from table E.2 – E.5 using the semantics from chapter 5

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>nds:Product1</td>
<td>nds:ProductID</td>
<td>1</td>
<td>nds:Product2</td>
<td>nds:BatchName</td>
<td>Folva 13</td>
</tr>
<tr>
<td>nds:Product1</td>
<td>nds:ProductTime</td>
<td>61249..</td>
<td>nds:Folva 13</td>
<td>nds:StartTime</td>
<td>6124..</td>
</tr>
<tr>
<td>nds:Product1</td>
<td>nds:ProductLength</td>
<td>65</td>
<td>nds:Folva 13</td>
<td>nds:EndTime</td>
<td>6124..</td>
</tr>
<tr>
<td>nds:Product1</td>
<td>nds:ProductDiameter</td>
<td>44</td>
<td>nds:Folva 14</td>
<td>nds:StartTime</td>
<td>6124..</td>
</tr>
<tr>
<td>nds:Product1</td>
<td>nds:ProductVolume</td>
<td>70</td>
<td>nds:Folva 14</td>
<td>nds:EndTime</td>
<td>6124..</td>
</tr>
<tr>
<td>nds:Product1</td>
<td>nds:ProductGreenSpots</td>
<td>0</td>
<td>nds:Innovator 1A</td>
<td>nds:StartTime</td>
<td>6124..</td>
</tr>
<tr>
<td>nds:Product1</td>
<td>nds:ProductBlackSpots</td>
<td>0</td>
<td>nds:A501</td>
<td>nds:ranBatch</td>
<td>Folva 13</td>
</tr>
<tr>
<td>nds:Product2</td>
<td>nds:BatchName</td>
<td>Folva 13</td>
<td>nds:A502</td>
<td>nds:ranBatch</td>
<td>Folva 14</td>
</tr>
<tr>
<td>nds:Product2</td>
<td>nds:ProductID</td>
<td>2</td>
<td>nds:A503</td>
<td>nds:ranBatch</td>
<td>Innovator 1A</td>
</tr>
<tr>
<td>nds:Product2</td>
<td>nds:ProductVolume</td>
<td>40</td>
<td>nds:Warnez</td>
<td>nds:CustomerContactPerson</td>
<td>Mr. Warnez</td>
</tr>
<tr>
<td>nds:Product2</td>
<td>nds:ProductGreenSpots</td>
<td>1635</td>
<td>nds:Warnez</td>
<td>nds:CustomerContactPerson</td>
<td>Mr. Lamb Weston</td>
</tr>
<tr>
<td>nds:Product2</td>
<td>nds:ProductBlacsSpots</td>
<td>1635</td>
<td>nds:Lamb Weston</td>
<td>nds:CustomerContactPerson</td>
<td>Mr. Lamb Weston</td>
</tr>
</tbody>
</table>
Appendix F - Newtec Ontology in Turtle Format

PREFIX nds: <http://www.newtec.dk/webservice/NewtecDatabaseScheme#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>.
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX ops: <http://www.openphacts.org/units#>
PREFIX qudt: <http://qudt.org/schema/qudt#>

nds:Customer rdf:type rdfs:class.
nds:Machine rdf:type rdfs:class.
nds:Batch rdf:type rdfs:class.
nds:StartedBatch rdf:type rdfs:class.
nds:EndedBatch rdf:type rdfs:class.
nds:ContactPerson rdf:type rdfs:class.
nds:StartedBatch rdfs:subclassof nds:Batch.
nds:ContactPerson rdfs:subclassof foaf:Person.
nds:ProductLength rdf:type rdfs:property.
nds:ProductDiameter rdf:type rdfs:property.
nds:ProductGreenSpot rdf:type rdfs:property.
nds:ProductLength rdf:s:range xsd:integer.
nds:ProductTime a time:Instant.
nds:Batch a time:Instant.
nds:End a time:Instant.
nds:Pixel rdf:type rdf:class.
nds:Symbol rdf:type rdfs:property.
nds:Symbol rdfs:domain nds:pixel.
nds:ProductLengthUnit rdf:type rdfs:property.
nds:ProductDiameterUnit rdf:type rdfs:property.
nds:ProductVolumeUnit rdf:type rdfs:property.
nds:ProductGreenSpotUnit rdf:type rdfs:property.
nds:ProductBlackSpotUnit rdf:type rdfs:property.
Appendix G – Web Application File Structure

The Web application file structure has its root at the “newtecSemanticWeb” folder which used by Apache Tomcat see Appendix A. The file structure is split into two parts – one for Web 2.0 solution and one for the Semantic Web solution

G.1 Web 2.0

newtecSemanticWeb
  mainMenuPanel.html
  batchReportPanel.html
  newtec.css
  images
    background.png
    button-medium.png
    NewtecSemanticWeb.png
  json_data
    A501_json_2products.txt
    A502_json_1products.txt
    A503_json_1products.txt
    batch_json.txt
    machine_json.txt
    customer_json.txt
  WEB_INF
    web.xml
    classes
      Batch.java
      BatchReportDataDTO.java
      BootEarlyContextListener.java
      compile_and_restart.sh
      Customer.java
      Machine.java
      NewtecSemanticWeb.java
      NewtecServlet.java
      Product.java
      QueryEngine.java
      RDFStore.java
  lib
    json-20140107.jar
    servlet-api.jar

G.2 Semantic Web

newtecSemanticWeb
  mainMenuPanel.html
  batchReportPanel.html
  newtec.css
  images
    background.png
    button-medium.png
    NewtecSemanticWeb.png
json_data
A501_json_2products.txt
A502_json_1products.txt
A503_json_1products.txt
batch_json.txt
machine_json.txt
customer_json.txt
WEB-INF
web.xml
classes
  BatchReportDataDTO.java
  BootEarlyContextListener.java
  compile_and_restart.sh
  NewtecSemanticWeb.java
  NewtecServlet.java
  NewtecServletRESTAPI.java
  QueryEngine.java
  RDFStore.java
lib
  json-20140107.jar
  servlet-api.jar
  jena-lib

#FOLDER
#DATA, ASCII Format of the product data
#DATA, ASCII Format of the product data
#DATA, ASCII Format of the product data
#DATA, ASCII Format of the batch data
#DATA, ASCII Format of the machine data
#DATA, ASCII Format of the customer data
#FOLDER
#WEB file for web server, see chapter 9
#FOLDER
#JAVA CLASS, for internal use
#JAVA CLASS, See chapter 9
#SCRIPT, to compile and launch Apache Tomcat
#JAVA CLASS, See chapter 9
#JAVA CLASS, See chapter 9
#JAVA CLASS, See chapter 11
#JAVA CLASS, See chapter 9 and 11
#JAVA CLASS, See chapter 9 and 11
#FOLDER
#JAR file, for json interface
#JAR file, servlet api for Apache Tomcat
#FOLDER, jar files to run Apache Jena, see appendix C
Appendix H – Quick Source Viewer

The Google Chrome Browser does not directly have the ability to show the source code for rich Web clients using Java Script to populate the Web page. The application “Quick Source Viewer” can be installed in the Chrome Browser to allow such capabilities.

**H.1 Version**

Quick Source Viewer: Version 1.0.8  
Google Chrome Version: Version 37.0.2062.120

**H.2 Download and Installation of the Application**

First step is to download the application by entering the following URL in the Google Chrome browser.

https://chrome.google.com/webstore/detail/quick-source-viewer/cfmcghennfbpmhemnnfjhdmnbidpanb

This will bring up the page below in figure D1 and by clicking the green button in the top right corner the application will be installed.

![Figure H1](Image)

*Figure H1*

*Installation of the Google Chrome application Quick Source Viewer.*

**H.3 Using the Application**

Getting the HTML source code is now quite simple go to the Web page of interest and click the sprocket in the top right hand corner to view the source code. See figure H2 below.

![Figure H2](Image)

*Figure H2*

*The usage of the application is done opening the Web page and clicking the sprocket in the top right hand corner.*
The application opens a new tab that contains the source code on the right hand side see figure H3 below.

**Figure H3**
The Quick Source Viewer opens a new tab with the source code from the selected Web page on the right hand side.
Appendix I – RDF Parser and Destiller

Extracting the RDF information from web pages can be quite cumbersome. There are number of RDF destiller and parsers available on the WWW. W3C has a destiller and parser that can extract and transform the RDF from a web page into Turtle format. Another tool from www.RDFa.info/tool can convert the embedded RDF into a RDF graph.

I.1 W3C RDFA 1.1 Distiller and Parser

The tool is available online at

http://www.w3.org/2012/pyRdfa/#distill_by_input

Below in figure I1 is a screenshot of the online destiller and parser tool from W3C.

The “Go!” button at the bottom will transform the RDF from the HTML code into Turtle format see figure I2 below.

Figure I1

W3C online RDF destiller and parser screenshot. The html code is copied and pasted in form the Newtec Semantic Web solution. The “Go!” button at the bottom starts the transformation.

Figure I2

Output from the W3C RDF destiller and parser. The format is Turtle. The input is from the Newtec Semantic Web Solution.
I.2 RDFa.info RDF Destiller and Parser

The RDF can be transformed into a RDF graph. The online tool RDFa.tool can be used for this. The tool is available at:

http://rdfa.info/play/

Below in figure I3 is a screenshot of the online destiller and parser tool from RDFa.info.

![RDFa.info destiller and parser using the Newtec semantic web as input in the left hand window. The right hand window displays the HTML output. The bottom window shows the RDF graph of the RDF that was embedded inside the HTML code.](image)

Figure I3 above shows how the Newtec semantic web solution can be destilled and parsed into a RDF graph.