Integrering av semantiska webben i ett existerande system

Sakarias Stenbäck

Examensarbete för ingenjör (YH)-examen

Utbildningsprogrammet för informationsteknik

Vasa 2016

EXAMENSARBETE

Författare:	Sakarias Stenbäck
Utbildningsprogram och ort:	Informationsteknik, Vasa
Handledare:	Kaj Wikman

Titel: *Integrering av semantiska webben i ett existerande system*

Datum: 10.4.2016 Sidantal: 36

Abstrakt

Arbetet gjordes åt InPlace Solutions och gick ut på att implementera semantiska webben i ett av deras existerande system. Detta gjordes genom att skapa en applikation som konverterar tabelldata till RDF-data samt genom att byta ut datalagringen till en RDF-databas. Informationen som behandlades var observationer av länders säkerhetssituation. Dessa observationer skull kunna filtreras och sändas till en annan applikation där de kunde visualiseras. Genom att integrera semantiska webben går det att länka ihop data om dessa observationer till offentliga RDF-grafer och på så sätt få mera information om t.ex. länderna.

Applikationen kodades i C# som en Windows Forms-applikation, och för att hantera RDF-data användes kodbiblioteket dotNetRDF. Filtreringen görs av användaren via det grafiska gränssnittet. Resultatet presenteras i en lista och kan sändas till företagets applikation där det visualiseras på en världskarta.

Resultatet blev en applikation som konverterar tabelldata till RDF-data och där användaren kan filtrera observationerna. Genom att visualisera dessa observationer får användaren en bra överblick över säkerhetssituationen i världen. I och med detta arbete har företaget en bra grund för vidareutveckling inom området.

Språk: svenska Nyckelord: semantiska webben, RDF, ontologi, SPARQL

BACHELOR'S THESIS

Author:	Sakarias Stenbäck
Degree Programme:	Information Technology, Vaasa
Supervisors:	Kaj Wikman

Title: *Integration of The Semantic Web in an existing system*

Date: 10.4.2016 Number of pages: 36

Summary

The thesis was done for InPlace Solutions and consisted of integrating The Semantic Web into an existing system. This was done by creating an application where table-data was converted to RDF-data and by swapping the data storage to a RDF-database. The information that was treated consisted of observations done on countries security situation. These observations should be able to be filtered and sent to another application where they could be visualized. By integrating The Semantic Web, it is possible to link data about the observations to public RDF-graphs and thus get more information about e.g. the countries.

The application was programmed in C# as a Windows Forms-application, and to handle RDF-data the dotNetRDF-library was used. The filtration is done by the user in the graphical interface. The result is presented in a list and can be sent to the company's application where it is visualized on a world map.

The result became an application that converts table data to RDF-data and where the user can filter the observations. By visualizing these observations, the user gets a good overview of the security situation in the world. As a result of this thesis the company has a good foundation for further development regarding this area of technology.

Language: Swedish Key words: The Semantic Web, RDF, ontology, SPARQL

Innehållsförteckning

1 Uppdragsgivare och uppgift 1

1.1 Uppdragsgivare 1

1.2 Bakgrund 1

1.3 Uppgift 2

2 Semantiska webben 3

2.1 Koncept 3

2.2 Semantik 5

2.3 Länkad Data 5

3 Tekniker 7

3.1 RDF 7

3.2 RDF-triplett 8

3.3 Vokabulär 10

3.4 RDF Schema 10

3.5 Ontologi 12

3.6 OWL 12

3.7 SPARQL 15

4 Verktyg 17

4.1 Protégé 17

4.2 Microsoft Visual Studio 18

4.3 dotNetRDF 19

4.4 OpenLink Virtuoso 20

5 Utförande 21

5.1 Planering 21

5.2 Applikationen 22

5.3 Datamodell 23

5.4 Konvertering av tabelldata till RDF-data 25

5.5 Filtrering av RDF-data 28

5.6 Sändning av data för visualisering 31

6 Resultat och diskussion 33

6.1 Resultat 33
6.2 Diskussion 33
7 Källförteckning 35

1 Uppdragsgivare och uppgift

I detta kapitel presenteras uppgiften för examensarbetet samt företaget som fungerade som uppdragsgivare.

1.1 Uppdragsgivare

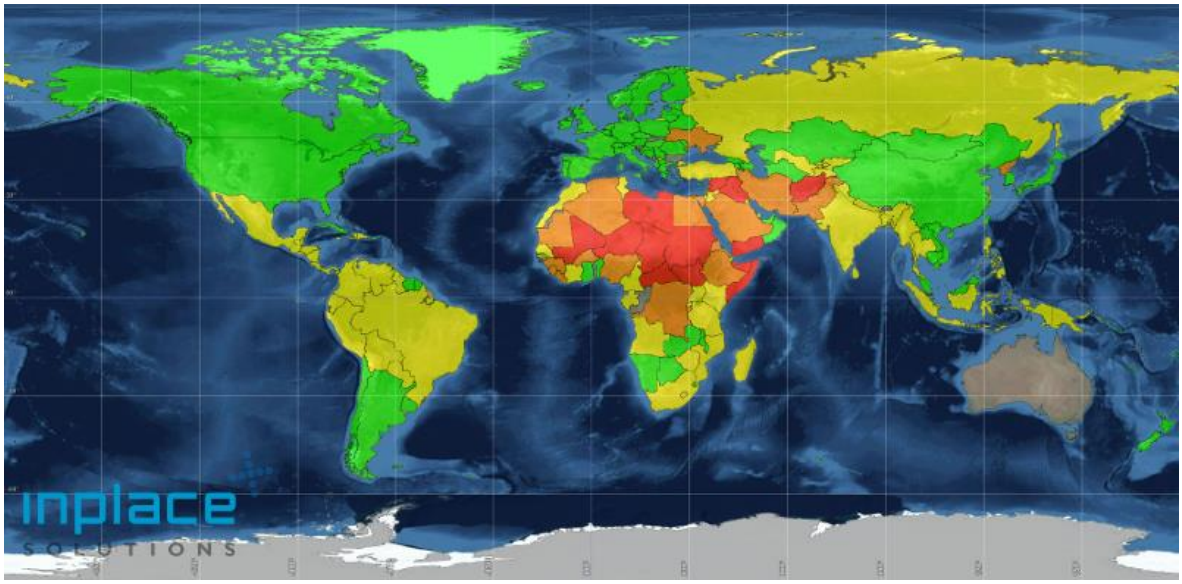
InPlace Solutions är ett företag som jobbar med lägesdata, vilket innefattar riskhantering, behandling av gps-data och producering av kartor. Företaget är lokaliserat i Kronoby och grundades 1994 av Thomas Nynäs som fungerar som chef.

Figur 1. InPlace Solutions logo.

InPlace Solutions har i dagsläget åtta anställda och deras huvudsakliga kunder är vatten- och elkraftverk, försvarsmakten och olika sjukhusdistrikt. En bra beskrivning av företagets produkter är att de tar data och gör datan till information, som i sin tur blir till kunskap hos användarna. Produkterna erbjuder bland annat administration av kartor, situationsbilder i realtid samt hantering och analys av lägesbundna data. En stor del av tjänsterna de erbjuder är konsultation angående implementation av GIS-lösningar. [1]

1.2 Bakgrund

InPlace Solutions samlar in data som beskriver säkerhetssituationen i många av världens länder. Dessa observationer innehåller information om vilket land det är frågan om, när observationen har gjorts och detaljer kring säkerhetssituationen. Beroende på om det är fred, krig eller om det har skett någon naturkatastrof så klassas länderna med siffrorna ett till fyra enligt hur säkert det är att vistas i och resa till länderna. Denna information importeras sedan till en av företagets produkter där informationen visualiseras på en världskarta genom att färglägga länderna.



Figur 2. Visualiseringen som görs av applikationen. [1]

Systemet fungerar genom att importera Excel-filer till en SQL-databas, varifrån observationerna sedan laddas in i produkten där den visualiseras. Detta system skulle bytas ut till ett som integrerar semantiska webben. Genom att göra det skapas möjligheten att länka den samlade informationen till publika RDF-grafer och på så sätt få mera informationen om t.ex. länderna.

1.3 Uppgift

Uppgiften som utfördes gick ut på att integrera semantiska webben i en av företagets produkter. Genom att under arbetets gång dokumentera de olika stegen skulle företagets anställda ha möjlighet att bekanta sig med teknikerna som ingår i semantiska webben. Konceptet skulle också demonstreras genom att skapa en applikation som använde sig av dessa tekniker och på så sätt fungera som grund för vidareutveckling.

Uppgiften var att byta ut det befintliga systemet till ett med stöd för RDF-data. Applikationen som gjordes skulle konvertera observationerna som fanns lagrade i tabellform i Excel-filer till RDF-data och sedan spara detta till en kompatibel databas. Dessutom skulle applikationen kunna hämta och presentera observationerna samt ge användaren möjlighet att filtrera dem. De filtrerade observationerna skulle sedan skickas till företagets produkt där den kunde visualiseras. Alla komponenter som användes i applikationen skulle bestå av öppen källkod.

2 Semantiska webben

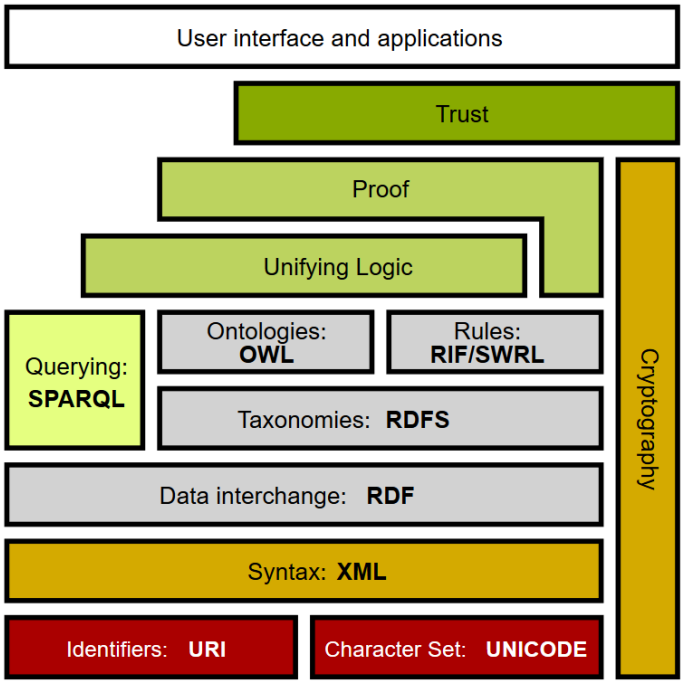
I detta kapitel förklaras semantiska webben som koncept och vilka komponenter som ingår i uppbyggnaden av den.

2.1 Koncept

Semantiska webben (eng. The Semantic Web) är olika standarder som är framtagna för att göra webben smartare. Visionen är att ha all publicerad data i samma format så att det lätt går att länka ihop och upptäcka ny data. Detta görs med hjälp av olika tekniker som är öppna standarder för att så många som möjligt skall börja anpassa sig till semantiska webben. [16]

Mannen bakom konceptet är Tim Berners-Lee. Han är också grundaren av webben, World Wide Web, som vi använder oss av idag med HTML- och HTTP-standarderna. Det fanns mera avancerade hypertext system när WWW blev till, men eftersom Berners-Lees system byggde på enkla specifikationer som han publicerade som öppna standarder blev det så populärt. På samma sätt är standarderna för semantiska webben publicerade som W3C standarder så att alla kan ta del av och implementera dem. Värt att nämna är att semantiska webben inte är ett nytt internet utan bygger på den befintliga infrastrukturen. Bob DuCharme definierar semantiska webben som ”*a set of standards and best practices for sharing data and the semantics of that data over the Web for use by applications”* [4]. [3]

Ett av målen med semantiska webben är att internet skall bli smart. Med smart menas inte direkt smarta applikationer eftersom det redan finns gott om sådana, utan smart med tanke på att rätt och uppdaterad information alltid är tillgänglig. Hur välprogrammerad och smart än en webbapplikation är så är den bara så användbar som dess data är korrekt. En användare som besöker en webbsida förväntar sig att informationen på sidan är uppdaterad och korrekt. I annat fall upplever användaren webben som ”dum” om en sida har uppdaterad information medan en annan inte har det. För att förhindra detta behövs inte en smart infrastruktur i sig utan en infrastruktur som länkar data till smarta applikationer så att informationen är uppdaterad och användningen upplevs som smart. [5]



Figur 3. Semantiska webbens uppbyggnad. [6]

Med figur 3 som grund gås det igenom de delar av semantiska webben som framkom under arbetes gång. En mera grundlig genomgång av dessa tekniker tas upp i kapitel 2.

Längst ner i figuren finns Unicode och URI. Unicode är en standard för att formatera internationella tecken för att möjliggöra att alla språk kan användas på internet. URI, som står för Uniform Resource Identifier, är en standardiserad global resurs-id som används för att särskilja resurser från varandra. [7]

Extensible Markup Language (XML) är ett märkspråk (eng. markup languange) som används för att strukturera information och ger en syntax för metadata. Detta används i semantiska webben i form av XML namnrymder (namespaces) och XML scheman. [8]

Resource Description Framwork, RDF (se kap. 3.1), är en datamodell för hur man lagrar samt representerar data om data, dvs. metadata, i så kallade RDF-grafer. RDF Schema (se kap. 3.2) är en extension till RDF som ger möjlighet att definiera klasser och egenskaper. OWL som står för Web Ontology Language (se kap. 3.4) lägger ytterligare till möjligheten att skapa avancerade klasstrukturer och egenskaper för att med hjälp av dessa få mer detaljerade modeller som kallas ontologier (se kap 3.3). [7]

SPARQL står för Protocol and RDF Query Language (se kap. 3.5) och är ett förfrågningsspråk samt ett protokoll med vilket man gör förfrågningar mot en databas och hämtar RDF-data. [9]

2.2 Semantik

Ordet semantik syftar på ord och uttrycks betydelse. Med hjälp av teknikerna som används för att bygga upp semantiska webben förklaras ord och koncept så att applikationer kan förstå vad t.ex. ett dokument handlar om. Vi människor förstår innehållet i ett dokument och kan dra slutsatser av det vi läser, men för en maskin är det bara olika tecken radade efter varandra. [16]

Genom att beskriva och länka ihop innehållet med hjälp av relationer bygger man upp ett nätverk av ord och uttryck vilket gör att applikationer kan programmeras att förstå ett koncept. Det vill säga *förstå* i den meningen att applikationen med hjälp av kopplingarna mellan data kan presentera information åt användaren som denne kan tänkas vara intresserad av baserat på innehållet i ett dokument. [10]

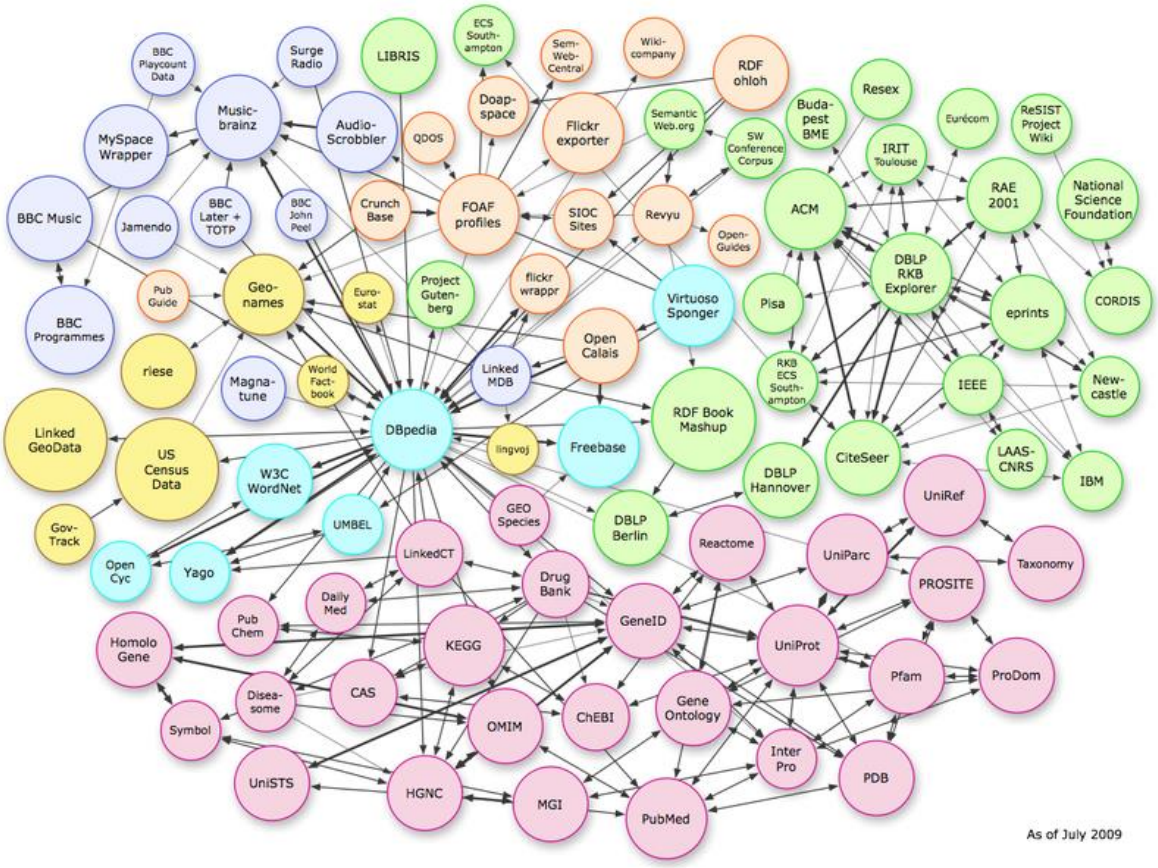
Som exempel kan tas ett dokument som handlar om fotboll. Läsaren förstår att det handlar om en sport, och beroende på läsarens kunskaper om fotboll och sport överlag förknippar läsaren olika saker med fotboll. Tanken med semantiska webben i detta fall är att man har taggat ordet fotboll och gjort kopplingar till information på webben som hör till fotboll, t.ex. historia och regler. Därmed kan förslag ges på andra dokument om fotboll eller en annan sport som läsaren kanske är intresserad av.

2.3 Länkad Data

Länkad data (eng. Linked data) är termen som beskriver offentlig data och relationerna mellan detta som bygger upp semantiska webben. Tanken med länkad data är att webben skall vara som en stor databas där all data är i samma format och länkas ihop för att skapa ett stort nätverk av information. Det är Tim Berners-Lee som förutom semantiska webben också har definierat länkad data och han har lagt fyra regler som grund för konceptet:

1. Varje resurs skall bestå av en URI
2. HTTP URIs skall användas så att det går att få mera information om en resurs
3. Standarder som RDF och SPARQL skall användas för att erhålla mera information om en resurs
4. Länkar till andra URIs skall också presenteras så att nya resurser kan upptäckas.

Berners-Lee berättar att dessa punkter är mera riktlinjer än regler, men att den fulla potentialen av länkad data inte uppnås om man avviker från dem. [11]



Figur 4. Illustration av Länkad data. [12]

Man kan tänka sig länkad data som en enda stor databas på webben där all data är ihopkopplad med hjälp av relationer. Traditionella relationsdatabaser är och kommer troligen att vara det mest använda sättet att lagra data på, men det finns områden där länkad data har fördelar. Ett av dessa områden är delning av data. I relationsdatabaser använder man sig av primär- och sekundärnycklar för att koppla ihop data mellan olika tabeller inom samma databas enligt en egen modell. Detta gör att om man vill länka ihop data mellan olika databaser med helt olika modeller och nycklar krävs det en omstrukturering av databaserna för att göra en gemensam modell med gemensamma nycklar. Länkad data löser detta genom att bara ha en modell för hur data lagras, RDF (se kap. 3.1), och att använda sig av globala nyckar i form av URIs, vilket gör att all data som publiceras enligt denna modell lätt kan länkas ihop. [13]

Som exempel kan tas två oberoende webbsidor där den ena listar alla filmer som vunnit en Oscar samt skådespelarna som medverkat i dessa filmer, och den andra webbsidan har en stor databas på skådespelare och mer detaljerad information om dessa. En användare som besöker en av sidorna skulle uppenbart dra nytta av att dessa sidors data skulle vara länkade så, att användaren skulle kunna söka upp en film och få mer information om en skådespelare som medverkat i den. Med två olika relationsdatabaser är det inte möjligt att koppla ihop tabeller på detta sätt. För att förverkliga detta borde de båda parterna, som uppehåller

sidorna, samarbeta och göra en gemensam modell på databaserna så att t.ex. båda använder samma id på alla skådespelare för att på så sätt kunna länka ihop data.

Om båda webbsidorna skulle använda sig av RDF som dataformat och ha samma vokabulär för att beskriva data skulle inga ändringar i strukturen vara nödvändiga för att kunna länka ihop data. Detta är en fördel med länkad data jämfört med relationsdatabaser, men samma sak kan också ses som en nackdel jämfört med relationsdatabaser med tanke på t.ex. prestanda. Beroende på vad man har för behov, är den ena lösningen mera fördelaktig än den andra.

3 Tekniker

Detta kapitel kommer att ta upp de tekniker som användes för att utföra uppgiften. Varje del kommer att ge en grundläggande förståelse för hur tekniken fungerar och vilken roll den spelar i uppbyggnaden av semantiska webben.

3.1 RDF

RDF – Resource Description Framework – ligger som grund för semantiska webben och är en datamodell för hur data skall sparas. RDF är en W3C rekommendation vilket motsvarar en webb standard och är utvecklat för att representera information på webben så att mjukvara kan ”förstå” informationen och så att informationen lätt kan delas mellan oberoende system. Strukturen på datalagringen i RDF-format är så kallade tripletter (eng. triples) som består av tre delar: subjekt, predikat och objekt. [14]

Tabell 1. Exempel på tripletter.

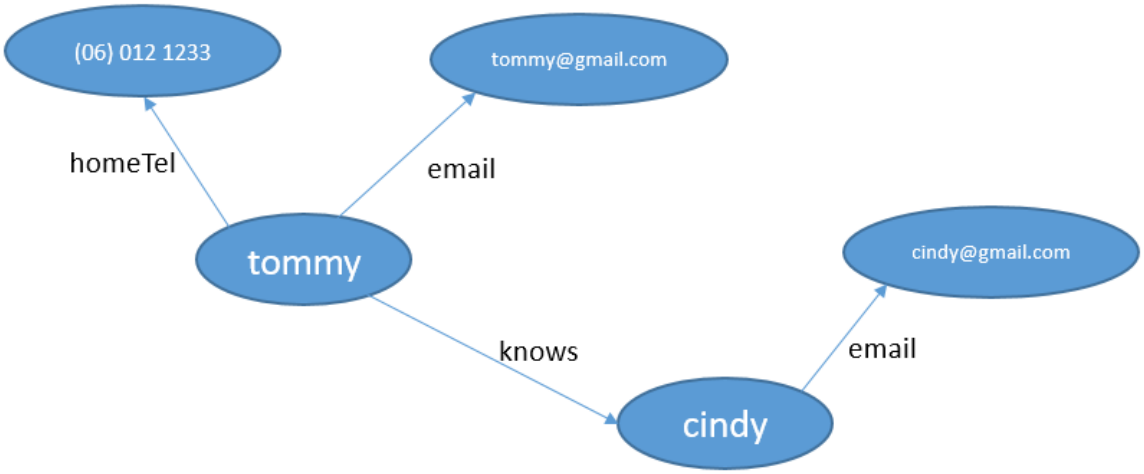
Subjekt	Predikat	Objekt
tommy	homeTel	(06) 012 1233
tommy	knows	cindy
cindy	email	cindy@gmail.com

Tabell 1 visar exempel på konceptet med tripletter, och skall tolkas som att det finns två olika resurser, `tommy` och `cindy`, där `tommy` har ett hemtelefonnummer och `cindy` har en e-postadress. Tanken med detta format är att man lätt skall kunna lägga till relationer till subjekten utan att behöva ändra på modellen. Om man t.ex. vill lägga till en e-postadress åt `tommy` så läggs en triplett till där subjektet är `tommy` och predikatet är email och objektet hans e-postadress. [9]

Tabell 2. Ny triplett tillsatt till ett befintligt subjekt.

Subjekt	Predikat	Objekt
tommy	homeTel	(06) 012 1233
tommy	email	tommy@gmail.com
tommy	knows	cindy
cindy	email	cindy@gmail.com

Som syns i tabell 2 ändrades inte strukturen på tabellen, det tillkom endast en ny rad. Detta till skillnad från tabelldata där hela strukturen på tabellen ändras om en ny egenskap läggs till. Detta gör att sammanslagning av RDF-data går väldigt enkelt i och med att all data är representerad i samma form.



Figur 5. RDF-graf visualiserad.

Flera tripletter i samma fil som har kopplingar till varandra bildar en så kallad RDF-graf. När en RDF-graf visualiseras representeras subjekten och objekten som noder och predikaten som länkar mellan noderna. En grafisk representation av tabell 2 kan ses i figur 5.

3.2 RDF-triplett

Som nämndes ovan sparas RDF-data i form av tripletter som består av subjekt, predikat och objekt. Första kolumnen i tripletten är subjektet som fungerar som id för en resurs. Andra kolumnen är predikatet som är relationen eller egenskapen mellan subjektet och objektet. Objektet är antingen en annan resurs, dvs en annan tripletts subjekt, eller en textsträng. Eftersom dessa resurser skall vara unika så används en unik URI som id och därför är alltid subjektet i en triplett en URI. Också predikatet måste vara en URI eftersom varje relation skall vara unik. Objektet kan däremot vara antingen en URI om det är frågan om en annan resurs eller en textsträng om det är frågan om ett värde. [15]

Tabell 3. RDF-triplettens struktur.

Subjekt	Predikat	Objekt
URI	URI	URI/textsträng

Till skillnad från URL (Uniform Resource Locator) som representerar en unik fysisk plats på internet så kan en URI (Uniform Resource Identifier) representera vilken typ av resurs som helst och det behöver inte finnas en webbsida bakom den [17]. Detta gör att en resurs kan representera precis vad som helst: t.ex. ett föremål, en person eller en världsreligion. En URI kan också vara en URL men behöver inte vara det. Ett exempel på en RDF-graf i textformat syns i figur 6.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .
@prefix ex : <http://example.com/> .

ex:person01 rdf:type  foaf:Person ;
            foaf:name "Eva" .

ex:person02 rdf:type           foaf:Person ;
            foaf:name          "Tommy" ;
            vcard:hasEmail     "tommy@gmail.com" ;
            vcard:hasTelephone "(06) 012 1233" ;
            foaf:knows         ex:person03 .

ex:person03 rdf:type       foaf:Person ;
            foaf:name      "Cindy ;
            vcard:hasEmail "cindy@gmail.com" ;
            ex:hasParent   ex:person01 .
```

Figur 6. RDF-graf i text.

I figuren syns att tripletterna har URIs som subjekt och predikat, och objekten består antingen av en URI eller av en textsträng. Tripletterna i grafen är skrivna med Turtle syntaxen som är en W3C rekommendation och är framtagen för att göra dem mera lättläsliga för människor jämfört med t.ex. den äldre syntaxen RDF/XML. I Turtle syntaxen avslutas en triplett med en punkt, men liksom i figur 6 så kan man använda sig av semikolon för att inte behöva upprepa subjektet för varenda triplett, dvs. alla predikat och objekt med semikolon emellan ända fram till en punkt hör till samma subjekt. [9]

3.3 Vokabulär

För att effektivt kunna använda sig av semantiska webben behövs det överenskommelse över gemensamma vokabulärer. En vokabulär (eng. vocabulary) eller en namnrymd (eng. namespace) är en samling predikat och klasser som är framtagna för att förklara ett visst koncept. Tanken är att alla inom semantiska webben skall använda sig av samma vokabulär för samma koncept. I figur 6 som använder Turtle syntax är vokabulären definierade i början av filen med taggningen `@prefix` som följs av en förkortning och själva URI'n. När själva tripletterna definieras används förkortningarna för att göra filen mer lättläst. [26]

FOAF som är en förkortning av Friend of a Friend kan tas som exempel. Denna vokabulär är framtagen för att beskriva online profiler av personer och deras relationer. I figur 6 syns att FOAF används för att visa att instanserna `ex:person01`, `ex:person02` och `ex:person03` är av typen `foaf:Person`, vilket berättar att det är frågan om fysiska personer. I och med detta kan någon som är intresserad av att få information om alla personer i RDF-grafen göra en förfrågning mot den som returnerar alla instanser som är av typen foaf:Person. [18]

Man kan skapa egna vokabulärer med hjälp av RDF Schema och OWL (se kapitel 3.2 och 3.3), men troligtvis finns det redan en passande vokabulär på webben som någon annan har gjort och som redan används i flera RDF-grafer. Detta kräver att man lägger tid på att hitta vokabulärer som används men bidrar till en smidigare upplevelse för alla som vill ta del av data man publicerar. Det finns inga regler för vilka vokabulär man skall använda sig av men det visar sig snabbt vilka som är populär och välanvända på så sätt blir dessa inofficiella standarder. [9]

3.4 RDF Schema

RDF Schema är en W3C rekommendation vars fulla titel är ”RDF Vocabulary Description Language: RDF Schema”. RDF Schema är en extension till RDF och ger möjligheten att skapa vokabulär (se kapitel 3.1.2) med hjälp av egenskaper som `rdfs:domain`, `rdfs:range` och `rdfs:class`. Styrkan med RDF Schema är möjligheten att skapa vokabulär och ontologier som möjliggör härledning (inference) samt att beskriva egenskaperna med `rdfs:comment` och `rdfs:label`. Följande RDF-graf demonstrerar detta. [19]

```
@prefix ab:   <http://learningsparql.com/ns/addressbook#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

ab:Musician
      rdf:type rdfs:Class ;
      rdfs:label "Musician" ;
      rdfs:comment "Someone who plays a musical instrument" .

ab:MusicalInstrument
      a rdfs:Class ;
      rdfs:label "musical instrument" .

ab:playsInstrument
      rdf:type rdf:Property ;
      rdfs:comment "Identifies the instrument that someone plays" ;
      rdfs:label "plays instrument" ;
      rdfs:domain ab:Musician ;
      rdfs:range ab:MusicalInstrument .

ab:i0432 ab:firstName     "Richard" ;
         ab:lastName       "Mutt" ;
         ab:email          "richard49@hotmail.com" ;
         ab:playsInstrument ab:piano .
```

Figur 7 RDF-graf.

I figur 7 ser man att i RDF-grafen deklareras två klasser, en egenskap och en person. Instansen `ab:playsInstrument` är subjekt i flera tripletter som skiljs åt av semikolon (se kap. 3.1.1). Domain och range tripletterna skrivs om för att lättare kunna tolka dem.

```
ab:playsInstrument rdfs:domain ab:Musician .
ab:playsInstrument rdfs:range ab:MusicalInstrument .
```

Figur 8. Omskrivna tripletter.

Domain-egenskapen i figur 8 gör att objektet i en triplett som använder sig av `ab:playsInstrument` tillhör klassen `ab:Musician`. På samma sätt deklarerar range-egenskapen i figur 8 att subjektet i en triplett som använder `ab:playsInstrument` är av typen `ab:MusicalInstrument`. I RDF-grafen i figur 7 finns tripletten `ab:i0432 ab:playsInstrument ab:piano` vilket gör att `ab:i0432` nu tillhör klassen ab:Musician och ab:piano tillhör klassen ab:MusicalInstrument. [20]

Detta skiljer sig från objekt orienterad programmering där man först definierar en klass med dess attribut och sedan skapar objekt av den klassen som har alla attribut som klassen har. Med RDF kan samma instans höra till flera olika klasser då det är attributen eller egenskaper som avgör till vilka klasser man hör. Enligt dessa regler kommer `ab:i0432` att returneras om man frågar efter alla instanser av typen `ab:Musician` av grafen i figur 7 även om det inte direkt står att `ab:i0432` är av den typen. [21]

3.5 Ontologi

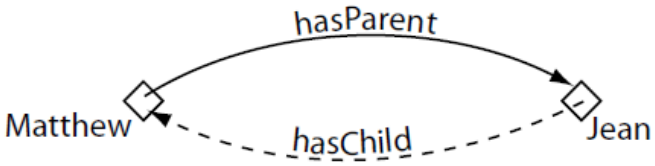
En ontologi är en datamodell med vilken man förklarar ett koncept och relationerna mellan olika ting som konceptet består av. Detta för att kunna representera kunskap om verkliga ting så att applikationer kan använda sig av det. En ontologi förser system och användare med egenskaper och klasser samt regler för hur dessa skall användas. Ontologier skapas i form av RDF-tripletter och sparas som RDF-grafer. Data som skall användas med ontologin kan antingen finnas i samma graf eller i en separat graf. Tanken är att man lätt skall kunna länka data att använda sig av en viss ontologi.[26]

Vokabulär kan anses vara ontologier, men vanligtvis menar man mera komplexa modeller när man talar om ontologier. Språket som används för att skapa ontologier är OWL, Web Ontology Language, som är ytterligare ett tillägg till RDF och RDF Schema. OWL tas upp i nästa kapitel. [22]

3.6 OWL

Web Ontology Languange (OWL) är en W3C-rekommendation som bygger vidare på RDF Schema. Precis som RDF Schema så medför OWL flera klasser, egenskaper och regler vilket gör att man kan skapa komplexa datamodeller som kallas ontologier (se kap. 3.3). Med hjälp av OWL kan man möjliggöra kraftfull mjukvaru-slutledning (eng. reasoning) vilket lägger grund för smartare applikationer. [23]

Reasoning betyder att mjukvara med hjälp av relationerna i en ontologi kan härleda nya korrekta relationer som inte direkt är definierade. Detta görs med hjälp av egenskaper som finns i OWL-vokabulären. Bland de viktigaste egenskaperna finns `owl:DataProperty` och `owl:ObjektProperty`. Dessa används för att skapa nya egenskaper i en ontologi och dessa egenskaper kan vara av olika typer vilket bestämmer hur de får användas. Exempel på sådana egenskaper är t.ex. inversa egenskaper, där exempelvis *hasParent* och *hasChild* naturligt kunde vara varandras inverser. [24]



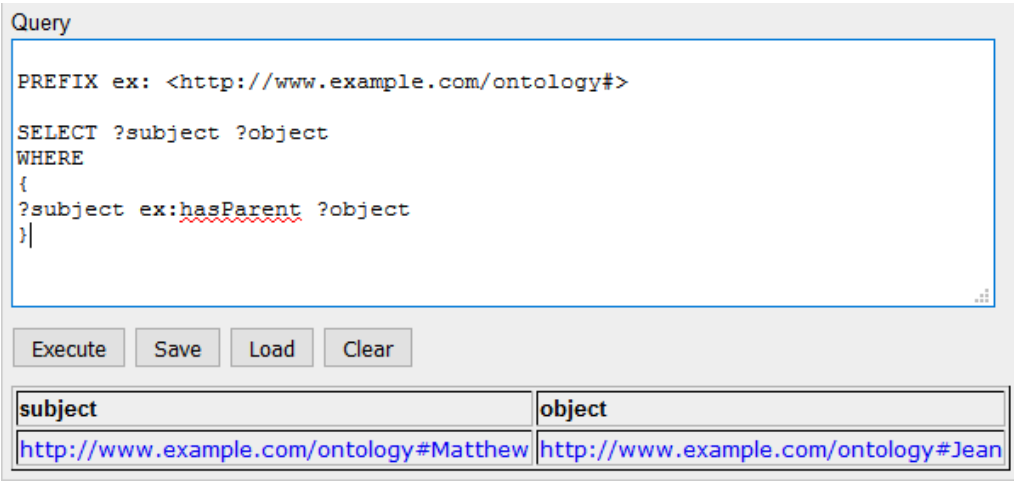
Figur 9. Invers egenskap. [24]

I figur 9 är det bara utskrivet att Matthew har en förälder Jean, men eftersom *hasParent*'s inversa egenskap är *hasChild* kan reasonern härleda sig till att Matthew är Jeans barn. Följande exempel visar hur det fungerar i praktiken, grafen ses i figur 10.

```
23
24      <!--
25      ///////////////////////////////////////////////////////,
26      // Object Properties
27      ///////////////////////////////////////////////////////,
28       -->
29      <owl:ObjectProperty rdf:about="&ontology;hasChild"/>
30
31      <owl:ObjectProperty rdf:about="&ontology;hasParent">
32          <owl:inverseOf rdf:resource="&ontology;hasChild"/>
33      </owl:ObjectProperty>
34
35      <!--
36      ///////////////////////////////////////////////////////,
37      // Classes
38      ///////////////////////////////////////////////////////,
39       -->
40      <owl:Class rdf:about="&ontology;Person"/>
41
42      <!--
43      ///////////////////////////////////////////////////////,
44      // Individuals
45      ///////////////////////////////////////////////////////,
46       -->
47      <owl:NamedIndividual rdf:about="&ontology;Jean">
48          <rdf:type rdf:resource="&ontology;Person"/>
49      </owl:NamedIndividual>
50
51      <owl:NamedIndividual rdf:about="&ontology;Matthew">
52          <rdf:type rdf:resource="&ontology;Person"/>
53          <hasParent rdf:resource="&ontology;Jean"/>
54      </owl:NamedIndividual>
55  </rdf:RDF>
```

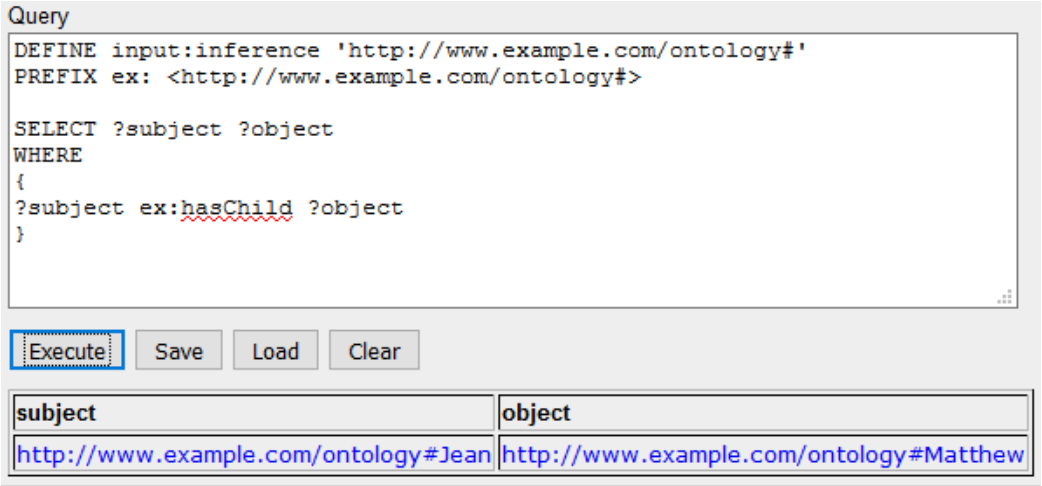
Figur 10. RDF-graf med inversa egenskaper.

Under Individuals-rubriken i figur 10 syns att endast Matthew har en relation till Jean genom egenskapen *hasParent*. Detta kan kontrolleras genom att ladda upp grafen till en databas med en SPARQL-motor så att det går att göra förfrågningar mot grafen.



Figur 11. SPARQL-förfrågning mot grafen.

Som man kan se i figur 11 så returneras tripletten som beskriver att Matthew har Jean som förälder, resultatet syns längst ner i figuren. SPARQL returnerar de tripletter passar modellen man konstruerar i WHERE-satsen, i detta fall alla tripletter som har predikatet `ex:hasParent` med vilka subjekt och objekt som helst, vilket bara fanns en av i grafen och dess subjekt och objekt returnerades eftersom det var specificerat i SELECT-satsen. Om man nu gör förfrågning där alla tripletter med `ex:hasChild` skall returneras borde inte några tripletter returneras eftersom det inte finns sådana i grafen. Men eftersom databasen som används har en inference-motor så klarar den av att läsa att de två egenskaper är varandras inverser och på så sätt tolka att Matthew måste vara Jeans barn eftersom Jean är förälder till Matthew.



Figur 12. Härledd triplett enligt inversa egenskaper.

I figur 12 kan man se hur SPARQL-förfrågningen returnerar tripletten `ex:Jean ex:hasChild ex:Matthew` (`ex:` används som förkortning för ontologins URI) som inte direkt finns utskriven i grafen (figur 10). För att inference-motorn skall fungera är man tvungen att definiera vilken graf som skall användas för slutledningen, vilket i detta fall var

samma graf som all data fanns i. Detta görs med DEFINE-kommandot till först i SPARQL-förfrågningen. [42]

3.7 SPARQL

SPARQL Protocol and RDF Query Language (SPARQL) är liksom SQL ett frågespråk men också ett protokoll för att kunna skicka och hämta RDF-data. SPARQL är en W3C rekommendation och den senaste versionen är 1.1 sedan 21.3.2013. [25]

För att få tag på önskad data gör man SPARQL-förfrågningar mot triplettdatabasen genom att konstruera triplettmönster som data skall passa in i. Man konstruerar alltså tripletter så att data man vill komma åt passar in på variablernas plats i triplettmönstren. [9] [25]

```
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix vcard:  <http://www.w3.org/2006/vcard/ns#> .
@prefix ex :    <http://example.com/> .

ex:person01 rdf:type  foaf:Person ;
            foaf:name "Eva" .

ex:person02 rdf:type            foaf:Person ;
            foaf:name           "Tommy" ;
            vcard:hasEmail      "tommy@gmail.com" ;
            vcard:hasTelephone "(06) 012 1233" ;
            foaf:knows          ex:person03 .

ex:person03 rdf:type        foaf:Person ;
            foaf:name       "Cindy ;
            vcard:hasEmail "cindy@gmail.com" ;
            ex:hasParent    ex:person01 .
```

Figur 13. RDF-graf.

Säg att man vill komma åt Tommys e-postadress som finns lagrad i grafen i figur 13. Detta görs genom att konstruera triplettmönstret i förfrågningen så att subjektet och predikatet passar in enligt grafen samt att det man vill åt, i detta fall e-postadressen, skall vara en variabel. Variabler kännetecknas med ett frågetecken i början med valfritt namn efter. [9]

```
PREFIX ex: <http://www.example.com/>
PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>

SELECT ?emailAddress
WHERE {
ex:person02 vcard:hasEmail ?emailAddress .
}
```

Figur 14. SPARQL-förfrågning.

Vad denna förfrågning gör är att berätta åt SPARQL-processorn hos databasen att returnera alla objekt för alla tripletter som har `ex:person02` som subjekt och `vcard:hasEmail` som predikat. I *SELECT*-satsen berättar man vilka värden man vill ha returnerade, i detta fall `?emailAddress`, och i *WHERE* -satsen konstruerar man triplettmönstren. *PREFIX*-delen definierar precis som i RDF-graferna förkortningar av vokabulär och ontologier för att göra allting mera lättläsligt. [9]

I fallet ovan antas det att man vet vilken id Tommy har, `ex:person02`. Detta är nästan aldrig fallet utan mera naturligt vore att man endast vet namnet på personen. I det fallet använder man sig av variabler inne i *WHERE*-satsen för att få rätt instans, i detta fall `ex:person02`, vilket syns i figur 15.

```
2
3   PREFIX ex: <http://www.example.com/>
4   PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>
5   PREFIX foaf: <http://xmlns.com/foaf/0.1/>
6
7   SELECT ?emailAddress
8   WHERE {
9   ?person foaf:name "Tommy" .
10  ?person vcard:hasEmail ?emailAddress .
11  }
12
```

Figur 15. SPARQL-förfrågning med flera variabler.

Om SPARQL-processorn hittar en triplett med predikatet `foaf:name` och objektet `Tommy` så lagrar den subjektet, i detta fall `ex:person02`, i variabeln `?person` vilken sedan används för att göra samma förfrågan och få samma resultat som i figur 14. Det enda som returneras är dock e-postadressen eftersom endast den finns i *SELECT*-satsen. [9]

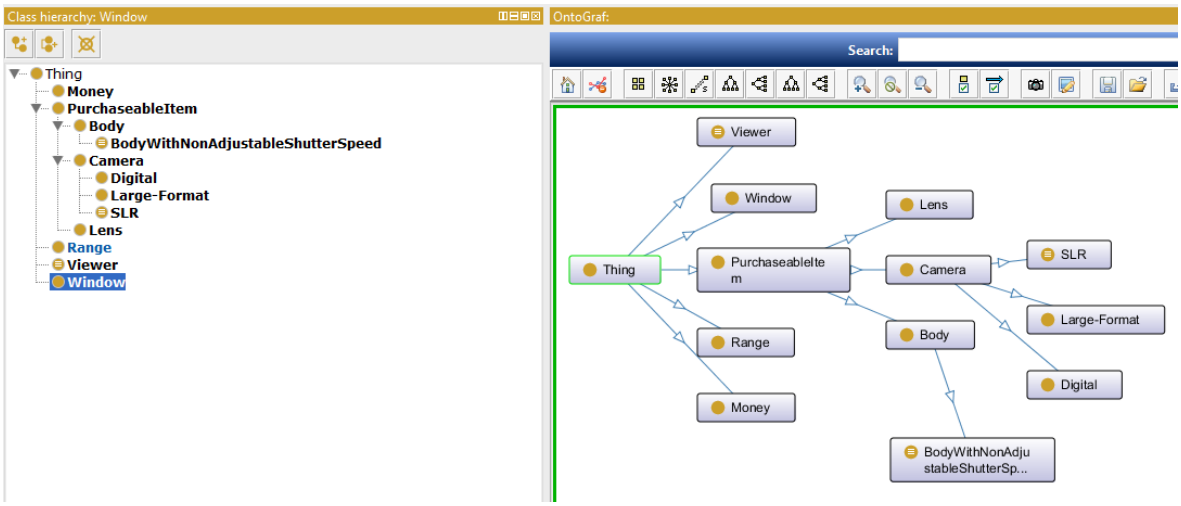
4 Verktyg

I detta kapitel nämns kort om de verktyg och dess funktionaliteter som i huvudsak användes för att utföra detta arbete.

4.1 Protégé

Protégé är en gratis applikation med öppen källkod som det går att skapa ontologier med. Applikationen har ett grafiskt gränssnitt med vilket man kan skapa t.ex. klasser, egenskaper och instanser. Protégé är utvecklat vid Stanford Center for Biomedical Informatics Research (BMIR) och är framtaget för att skapa och upprätthålla ontologier inom den medicinska världen, men det går att skapa vilka typer av ontologier som helst med applikationen. [27]

Applikationen har enkla menyer och verktyg för att bestämma vilka typer av klasser och kopplingar man vill skapa. Eftersom källkoden är öppen finns det plugins som medför olika funktioner som kan förenkla arbetet. En sådan plugin är OntoGraf som ger en grafisk representation av ontologin för att få en överblick över klasserna och kopplingarna.

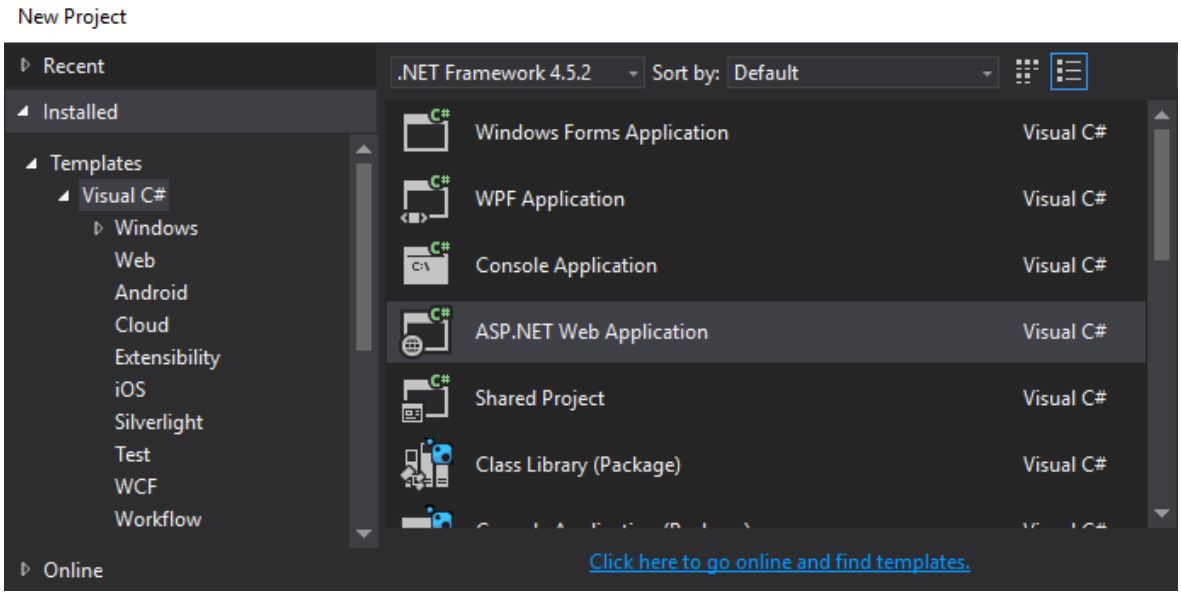


Figur 16. OntoGraf i Protégé.

I figur 16 syns klasserna till vänster i klass hierarkin, och till höger syns en grafisk representation av klasserna och hur de är kopplade till varandra. Andra funktionaliteter som Protégé erbjuder är att grafiskt bestämma om en egenskap t.ex. är invers eller funktionell istället för att vara tvungen att skriva det med ett textbehandlingsprogram.

4.2 Microsoft Visual Studio

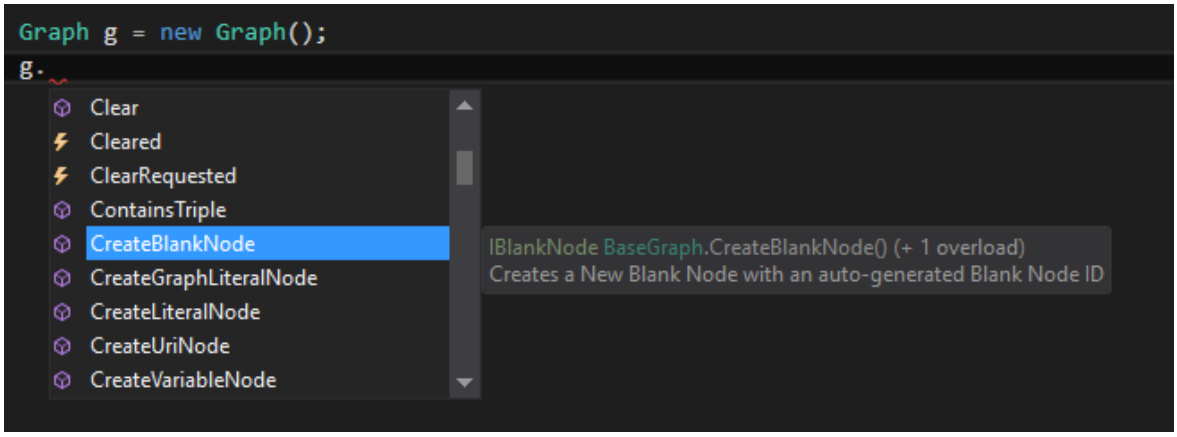
Visual Studio är utvecklat av Microsoft och är en programutvecklingsmiljö. Den har stöd för flera programmeringsspråk som t.ex. C++, C#, J# och F# samt stöd för bland annat debugging, tester och UI design. Det går att skapa många olika typer av projekt i Visual Studio och programmera applikationer för skrivbordet, webben eller mobila enheter. [28]



Figur 17. Olika typer av projekt i Visual Studio.

NuGet Manager kommer med Visual Studio och är en pakethanterar där det går att ladda ner olika kodbibliotek, t.ex. komponenter som hör till Microsofts .NET ramverk. Det går också att skapa egna paket publicera dem till NugGet Gallery som är databasen för alla paket. NuGet Manager användes i detta arbete för att ladda ner och använda dotNetRDF-ramverket (se kap. 4.3) som behövdes för att hantera RDF-data i C#. [29]

En funktion som Visual Studio har är IntelliSense. IntelliSense är ett samlingsnamn för funktioner som hjälper användaren att förstå koden och effektivera kodningen. Detta görs bland annat genom att lista ett objekts funktioner och egenskaper, visa info om parametrar för en metod samt visa förslag när man börjar skriva någonting. [30]



Figur 18. Exempel på listning av metoder med hjälp av IntelliSense.

4.3 dotNetRDF

dotNetRDF är ett kodbibliotek som är baserat på .NET och erbjuder funktionalitet för arbete med RDF, SPARQL och andra tekniker inom semantiska webben. Kodbiblioteket är byggt på öppen källkod och drivs som ett projekt lett av skaparen Rob Vesse. Biblioteket har stöd för flera olika databashanterare, bland andra Apache Jena och OpenLink Virtuoso (se kap. 4.4), varav det senare användes i detta arbete. [31]

De grundläggande funktionerna som dotNetRDF för med sig är att skapa RDF-grafer, tripletter och att ansluta till en databas eller en SPARQL-endpoint. Kärnklasserna bygger alla på tre olika interface: INode, IGraph och ITripleStore. Exempel på hur en triplett skapas och lagras i en graf syns i följande figur. [32]

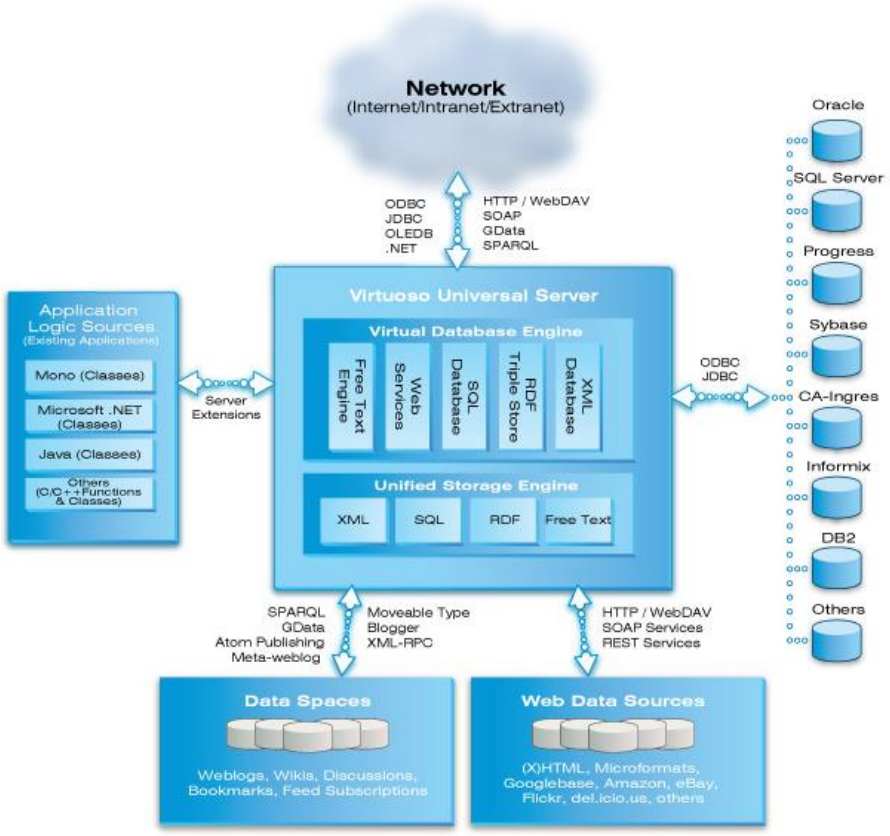
```
//Need a Graph first
IGraph g = new Graph();

//Create some Nodes
IUriNode dotNetRDF = g.CreateUriNode(UriFactory.Create("http://www.dotnetrdf.org"));
IUriNode createdBy = g.CreateUriNode(UriFactory.Create("http://example.org/createdBy"));
ILiteralNode robVesse = g.CreateLiteralNode("Rob Vesse");

//Assert this Triple
Triple t = new Triple(dotNetRDF, createdBy, robVesse);
g.Assert(t);
```

Figur 19. Exempel på hur en triplett skapas med hjälp av dotNetRDF.

I figuren syns till först hur en graf skapas, varefter subjektet, predikatet och objektet för tripletten skapas (se kap. 3.1.1 för mera information om tripletter). I dotNetRDF är en tripletts beståndsdelar någon typ av nod, i detta fall finns det två URI-noder och en nod med textdata. Till sist skapas tripletten av dessa noder och sparas i grafen. [32]

4.4 OpenLink Virtuoso

OpenLink Virtuoso är en så kallad Universal Server och kan fungerar som web-, fil- och databasserver. Den ger möjlighet att administrera SQL, XML och RDF samt att få tillgång till RDF-databasen via bland annat SPARQL, ODBC och lagrade SQL procedurer. Virtuoso har också en OWL-reasoner (se kap. 3.4) som har stöd för subklasser och inversa egenskaper. [33]



Figur 20. Översikt över OpenLink Virtuoso. [34]

OpenLink Virtuoso finns i två olika versioner, en kommersiell och en med öppen källkod. I detta arbete användes versionen med öppen källkod då detta var ett krav från arbetsgivaren. Alternativ till Virtuoso kunde vara Apache Jena som är ett Java-kodbibliotek med öppen källkod för att programmera applikationer för semantiska webben. Apache Jena har också en egen RDF-databas och SPARQL-endpoint. Detta valdes dock inte då jag inte var bekant med Java och de på företaget redan hade bekantat sig med dotNetRDF biblioteket (se kap. 4.3). Därför valdes Virtuoso som RDF-databas och SPARQL-endpoint och dotNetRDF som API för applikationen som kodades i C#. [33] [35]

5 Utförande

I detta kapitel beskrivs planeringsskedet, utförandet och funktionaliteten hos applikationen som skapades.

5.1 Planering

Eftersom semantiska webbens som koncept var väldigt nytt för både företaget och mig själv stod det klart att en stor del av arbetet skulle gå till informationssökning och dokumentering. Arbetets uppgift skulle varar att producera dokument som introducerar företagets anställda till semantiska webben samt en applikation där teknikerna skulle demonstreras och ligga som grund för vidareutveckling. Applikationen skulle göras som en Windows Forms-applikation i C#.

Den beräknade tidsåtgången delades upp i två fyraveckors-perioder. Fyra veckor för att bli bekant med konceptet och teknikerna och fyra veckor för att göra applikationen. Under hela arbetets gång skulle dessutom allt som gjordes dokumenteras. En presentation av hela arbetet skulle avsluta hela projektet.

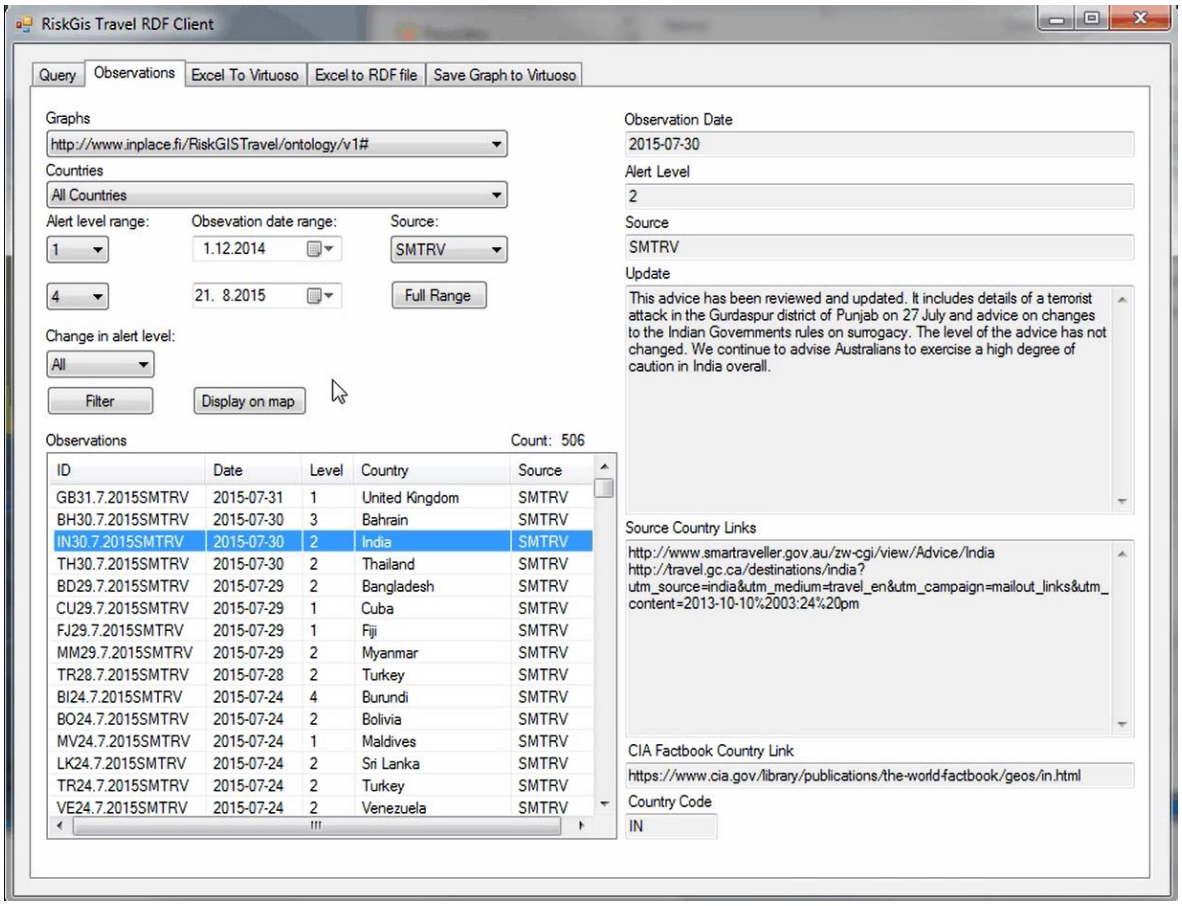
Företaget hade en bok som behandlade ämnet, *Semantic Web For Dummies* skriven 2009 av Jeffery T. Pollock, vilken jag läste som introduktion. Tanken var att jag efter att ha läst boken skulle ha tillräckligt med kunskap för att kunna avgränsa applikationen så att uppgiften inte skulle bli för överväldigande, och på det sättet veta vilka tekniker jag skulle vara tvungen att fördjupa mig i.

Det visade sig dock att boken mera var riktad mot företag som undrar vad semantiska webben kan användas till då större delen av boken gick åt att bygga upp scenarion där semantiska webben var överlägsen andra lösningar. Detta mötte inte mina behov vilka var att få en förståelse för teknikerna och ha exempel på hur man kommer igång med att bygga en applikation som använder sig av semantiska webben. Några praktiska exempel på hur man implementerade teknikerna fanns inte, inte heller några kodexempel eller hur man använder några av de verktyg som fanns med i boken.

Efter att ha läst boken och inte fått den grund jag behövde för att komma igång sökte jag information på nätet. Där hittade jag artiklar, webbsidor och forum med exempel och beskrivningar av de olika teknikerna så att jag kunde komma igång med att testa teknikerna och på samma gång dokumentera.

5.2 Applikationen

Applikationen kodades i C# i Visual Studio 2012 som en Windows Forms-applikation. Applikationens användargränssnitt består av ett enda fönster vari flikar används för att navigera mellan de olika funktionerna.



Figur 21. Användargränssnittet.

Det som applikationen skulle åstadkomma var att låta användaren välja en Excel-fil och en RDF-graf till vilken observationerna i Excel-filen skulle konverteras och sparas. Observationerna skulle presenteras och kunna filtreras enligt säkerhetsnivå, datum, land och källa. Man skulle också kunna lista de observationer vars säkerhetsnivå har ändrat under ett visst tidsintervall. På detta sätt kunde man lätt identifiera länder där det skett antingen en förbättring eller försämring i säkerhetssituation. Observationerna skulle kunna skickas till en annan applikation där de visualiserades.

5.3 Datamodell

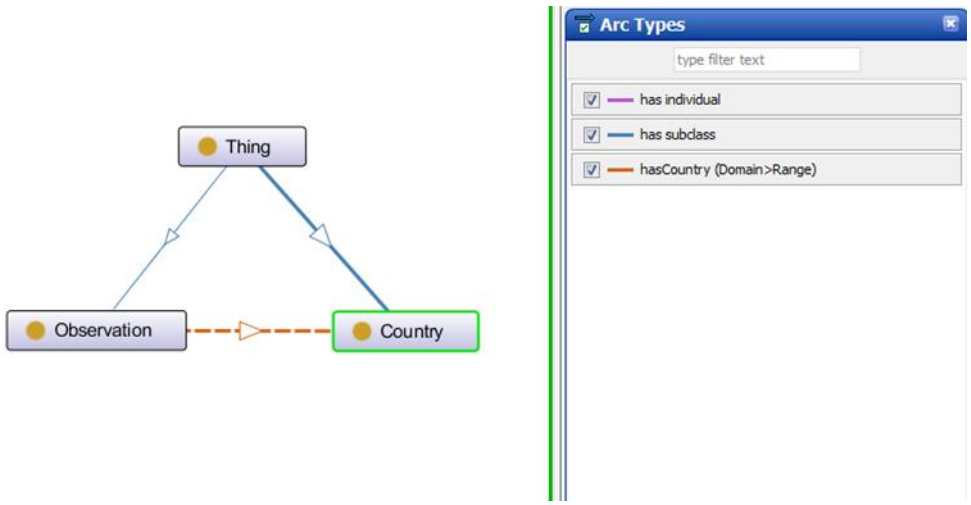
Data som skulle konverteras till RDF-data var i tabellform som skrivits in i Excel. Varje rad bestod av en observation, och kolumnerna bestod av information om observationen. Observationerna representerade säkerheten i ett visst land med tanke på hur säkert det är att resa till och vistas i landet vid tiden observationen gjordes. Kolumnerna finns listade i tabell 4.

Tabell 4. Kolumnerna för en observation .

Kolumn	**Förklaring**
observationDate	Datumet då observationen gjordes
alertLevel	Går från 1– 4, där 1 är säkert att resa till landet och 4 är farligt
source	Källan som publicerat observationen
update	Beskrivning av läget i landet för observationen i fråga
ciaFactbook	Länk till CIA Factbooks information om landet
countryID	Landets ISO-3166 alpha2 kod
countryName	Landets namn
link	Länk till landets senaste observation

Som id för observationerna fungerade en kolumn i Excel med automatisk numrering. Detta fungerar inte bra i detta sammanhang då det lätt kunde förekomma observationer med samma id då observationerna fanns lagrade i flera olika Excel-filer. Därför ändrades observationernas id att vara en kombination av *countryID*, *observationDate* och *source*. Dessa tre kolumner fungerar bra som id då det högst troligt bara kommer en observation om ett visst land från en källa per dag.

Till att börja med skapades ontologin (se kap. 3.3 för ontologi) i Protégé (se kap. 4.1) genom att först ta reda på vilka klasser som behövdes för uppgiften. Eftersom ett krav var att kunna sortera observationerna enligt land var Country en nödvändig klass, likaså Observation. Dessa två klasser gjordes disjunkta så att ingen instans kan vara både av typen Country och Observation på samma gång. Relationen mellan klasserna gjorde jag med objekt-egenskapen *hasCountry* vars domän (eng. domain) gjordes att vara klassen Observation och dess räckvidd (eng. range) att vara klassen Country. Detta betyder att en observation kan vara kopplad till endast ett land, men ett land kan ha kopplingar till flera observationer.



Figur 22. Klasserna och deras relation till varandra.

Till dessa klasser tilldelades egenskaperna som listas i tabell 4 så att *countyID*, *conutryName* och *ciaFactbook* definierar Country klassen och resten definierar klassen Observation.

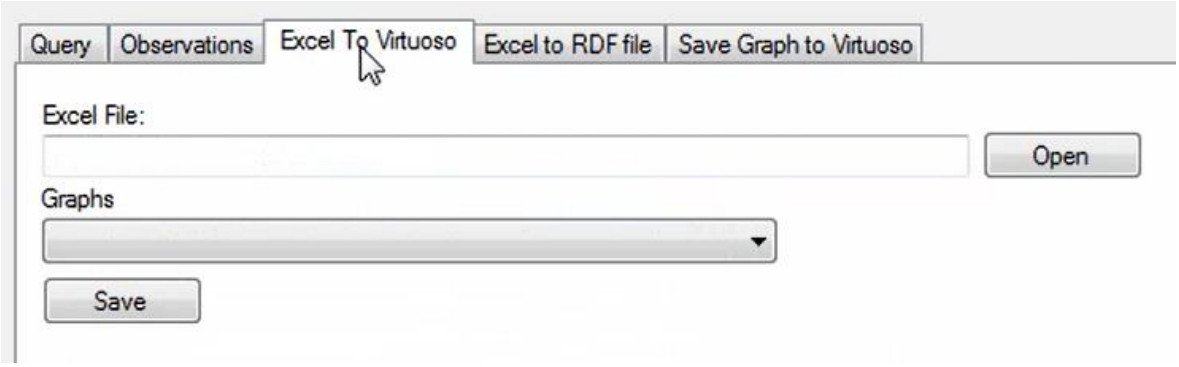
Till skillnad från objekt-orienterade system där det är klassdefinitionen som bestämmer vilka egenskaper varje objekt skall ha, fungerar RDF-baserade system så att det är själva egenskaperna som bestämmer vilken klass en instans tillhör. Detta betyder att oberoende vilka egenskaper en instans har, om *countryID*, *countryName* eller *ciaFactbook* är en av egenskaperna för instansen så antas denna vara av typen Country om inga andra regler förhindrar detta. Eftersom ontologin i detta fall var så liten behövde jag inte ägna desto mer tid till att fundera på regler för klasserna. [37]

Egenskaperna för klasserna kan antingen vara en URI eller en så kallad literalnode som består av en textsträng som representerar data. URI-noder består i detta fall av *ciaFactbook* och *link* som båda är länkar. Egenskapen *observationDate* är en literalnode med en XMLSchema-datum ändelse (eng. tag), vilket kallas en typed literal. Det är genom att sätta till ändelser i literalnodes som man bestämmer vilken data typ det är frågan om. SPARQL har stöd för typed literals vilket gör att det bland annat går att sortera resultat efter datum. [38]

Denna ontologi möjliggör att klassificera alla instanser att antingen höra till klassen Country eller Observation. Detta görs med hjälp av mjukvaruslutledning (se kap. 3.4) baserat på instansernas egenskaper som tidigare nämnts. Applikationen som gjordes i detta arbete definierar dock vilken klass instanserna hör till när data konverteras från tabelldata till RDF-tripletter. Detta val gjordes för att göra tester under arbetets gång enklare eftersom mjukvaruslutledningen då inte behövdes ta i beaktande om någonting uppträdde onormalt. Ontologins viktigaste uppgift är dock att definiera alla klasser, egenskaper och relationer så att användaren vet vad som skall användas i SPARQL-förfrågningar för att kunna få önskat resultat.

5.4 Konvertering av tabelldata till RDF-data

Användaren skulle välja en Excel-fil från vilken observationerna skulle konverteras till RDF-tripletter. Det skulle också väljas till vilken graf tripletterna skulle sparas i. Användargränssnittet för detta steg syns i figur 23.



Figur 23. Användaren kan välja filen för observationerna samt till vilken graf dessa skall konverteras.

För att kunna hantera RDF-tripletter hämtades dotNetRDF-biblioteket (se kap. 4.3) med NuGet Package Manager. Tripletterna skulle skapas från data som fanns i Excel-filen och för att kunna läsa in dessa behövdes en referens till Microsoft.Office.Interop.Excel-paketet med i projektet, detta gjordes också med NuGet Package Manager. Figur 24 visar hur data läses från filen och en lista med predikaten för tripletterna skapas.

```
private Graph ExcelToGraph(string sFilePath, string sGraphUri)
{
    Excel.Application objExcel = new Excel.Application();
    Excel._Workbook objWorkbook;
    Excel._Worksheet objWorksheet;

    objWorkbook = objExcel.Workbooks.Open(sFilePath);
    objWorksheet = (Excel._Worksheet)objExcel.Application.ActiveWorkbook.Sheets[1];
    objWorksheet.Select();

    List<String> lstPredicate = new List<String>();
    List<String> lstSubjects = new List<String>();
    int nColumns = objWorksheet.UsedRange.Columns.Count;
    int nRows = objWorksheet.UsedRange.Rows.Count;


    // Headers/Predicates
    for (int i = 1; i <= nColumns; i++)
    {
        if (objWorksheet.Cells[1, i].Value != null)
        {
            lstPredicate.Add(objWorksheet.Cells[1, i].Value.ToString());
        }
        else
        {
            nColumns = i - 1;
            break;
        }
    }
```

Figur 24. Kod som läser data från Excel.

Tabelldata går relativt enkelt att konvertera till RDF-tripletter vilket görs i applikationen. Varje rad i tabelldata är en unik instans med en id som i detta fall är en kombination av tre kolumner. Denna id-kombination fungerar som subjekt för varje triplett. Predikaten för tripletterna utgörs av kolumnerna i tabellen och objekten är cellens värde för de olika kolumnerna. Konceptet demonstreras till näst utifrån tabell 5.

Tabell 5. Tabelldata som skall konverteras till RDF-tripletter.

ID	observationDate	alertLevel	countryID	source
AF1.12.2014SMTRV	1.12.2014	3	AF	SMTRV

Data i tabell 5 kan konverteras till RDF-tripletter enligt det som nämndes ovan. Antalet tripletter som behövs är samma som antalet kolumner, id-kolumnen borträknad, vilket i detta fall är fyra. Subjektet för de fyra tripletterna är samma eftersom det handlar om samma observation, som i detta fall är *AF1.12.2014SMTRV*. Predikaten består av namnet på kolumnerna förutom id-kolumnen, och objekten består av cellernas värde för varje kolumn. Resultatet av konverteringen kan ses i figur 25.

```
2
3   AF1.12.2014SMTRV  obsertvationDate  1.12.2014 .
4   AF1.12.2014SMTRV  alertLeve         3 .
5   AF1.12.2014SMTRV  countryID         AF .
6   AF1.12.2014SMTRV  source            SMTRV .
7
8
```

Figur 25. Tripletterna som resulterade från tabell 5.

På detta sätt läser applikationen data från en Excel-fil och skapar RDF-tripletter utgående från den. Dessa tripletter sparas sedan i samma RDF-graf som ontologin. Före detta var det dock nödvändigt att göra grafen kompatibel med ontologin, det vill säga göra så att alla egenskaper överensstämmer med de som finns i ontologin. Subjekten måste också bestå av giltiga URIs. Detta löstes genom att konkatenera ontologins URI som finns i variabeln *sGraphUri* med subjekten och egenskaperna i det skede de skapades. I figur 26 syns detta där *nodeSubject* och *nodePredicate* får sina värden.

```
for (int i = 0; i < nActualRows - 1; i++) // rows
{
    IUriNode nodeSubject = g.CreateUriNode();

    nodeSubject = g.CreateUriNode(UriFactory.Create(sGraphUri + lstSubjects[i]));

    for (int a = 0; a < nColumns; a++) // columns
    {
        IUriNode nodePredicate = g.CreateUriNode(UriFactory.Create(sGraphUri + lstPredicate[a]));

        string sObject;
        if (objWorksheet.Cells[i + 2, a + 1].Value != null)
        {
            sObject = objWorksheet.Cells[i + 2, a + 1].Value.ToString();
        }
```

Figur 26. Subjekten och predikaten får sina värden.

Resultatet är en RDF-graf som har all data som fanns i Excel-filen men i form av RDF-tripletter. Varje triplett skapas från en rad i gången. Predikaten eller egenskaperna till en triplett är skapade från Excel-filens kolumn namn, subjekten är en kombination av värden från tre kolumner och objekten är skapade från cellernas värde för den aktuella raden. I denna form är grafen kompatibel med ontologin, men eftersom instanserna inte ännu har delats in i klasser kan ingen skillnad mellan dem göras.

Som redogörs för i kapitel 5.2 så finns det två klasser, Observation och Country, och varje instans i RDF-grafen skulle höra till en av dessa. Detta görs efter att RDF-grafen har skapats med all data från Excel-filen. Eftersom det är egenskaperna hos en instans som avgör vilken klass den hör till görs klassindelningen genom att identifiera de tripletter som beskriver egenskaperna och subjekten för dessa görs att tillhöra en av klasserna. [37]

Identifieringen görs genom SPARQL-förfrågningar mot RDF-grafen med de konverterade tripletterna. Eftersom dotNetRDF ger möjlighet att göra förfrågningar mot grafer i själva koden behöver man inte först ladda upp grafen till en SPARQL-endpoint. I förfrågningarna används kommandot CONSTRUCT som konstruerar tripletter efter det mönster man definierar, vilket gör att man skapa nya tripletter utifrån en befintlig graf. Detta tillåter att man kan modifiera data om det behövs före de nya tripletterna skapas vilket kan ses i figur 27. [39] [25]

```
SparqlParameterizedString queryString = new SparqlParameterizedString();

queryString.Namespaces.AddNamespace("", new Uri(m_sUriFrom));
queryString.Namespaces.AddNamespace("ontology", new Uri(m_sUriTo));

queryString.CommandText = " CONSTRUCT { ?countryId ?factbook ?link }"
                        + " WHERE { "
                        + " ?x :cia_factbook ?link. "
                        + " ?x :rgt_country_id ?country. "
                        + " ?x :rgt_country ?backup . "
                        + " BIND(IF(?country='', ?backup, ?country) as ?countryId) ."
                        + " FILTER(?factbook = ontology:ciaFactbook) "
                        + " } ORDER BY ?country";
```

Figur 27. SPARQL-förfrågning med ett CONSTRUCT-kommando.

Förfrågningen i figur 27 skapar tripletter enligt modellen som definieras i CONSTRUCT-satsen. Variablerna i denna sats får sina värden från WHERE-satsen, där det också kontrolleras om variabeln *?country* saknar värde. Kombinationen av BIND- och IF-satserna gör så att variabeln *?countryId* alltid får ett värde. Om landet saknar ISO-kod som skall sparas i variabeln *?country* så kommer istället landets namn som finns i variabeln *?backup* att sparas i *?countryId*. Detta är nödvändigt att kontrollera eftersom inte alla länder har en ISO-kod och på så sätt kunde flera tripletter bli ofullständiga.

Liknande förfrågningar görs för varje egenskap, vilka för klassen Country är *countryName*, *countryId* och *ciaFactbook* (se kap. 5.2). Resultatet blir en graf med tripletter där subjekten skall tillhöra klassen Country. Samma sak görs för egenskaperna som tillhör klassen Observation. Själva instanserna görs i dotNetRDF med funktionen *CreateIndividual()* som finns hos objekt av klassen OntologyClass, i detta fall *classCountry*. Detta syns i figur 28.

```
//run through the triples and create individuals and assert them to the ontology graph
foreach (Triple t in resultGraph.Triples)
{
    //individual id/name
    string sIndividual = t.Subject.ToString();
    string sObject = t.Object.ToString();

    int nIndex = sIndividual.IndexOf(" ");
    if (nIndex != -1)
    {
        sIndividual = sIndividual.Replace(" ", "_");
    }

    //create individual for specific class
    Individual individualCountry = classCountry.CreateIndividual(new Uri(m_objSparqlQueries.UriTo + sIndividual));
```

Figur 28. Instanser av klassen Country skapas.

Efter allt detta är hela kedjan av konvertering från tabelldata till RDF-data som följer ontologin färdig. Resultatet är en RDF-graf som innehåller både ontologin och alla RDF-tripletter. Grafen laddas upp till databasen genom ett VirtuosoManager-objekt som har fått de nödvändiga parametrarna för att ansluta till databasen.

5.5 Filtrering av RDF-data

Som databashanterare användes OpenLink Virtuoso (se kap. 4.4). I applikationen skapades en anslutning till databasen som RDF-grafen sparades i. För att hämta grafen skapades ett VirtuosoManager-objekt som ingår i dotNetRDF. Detta objekt får sina parametrar i konstruktorn för applikationsfönstret från applikationens konfigurationsfil.

```
//-----------------------------------------------
//  Constructor
//-----------------------------------------------
public frmMain()
{
    InitializeComponent();

    lvwColumnSorter = new ListViewColumnSorter();
    this.lvwObservations.ListViewItemSorter = lvwColumnSorter;

    m_objVirtuoso = new VirtuosoManager(Properties.Settings.Default.VirtuosoAddress,
                                        Properties.Settings.Default.VirtuosoPort,
                                        VirtuosoManager.DefaultDB,
                                        Properties.Settings.Default.VirtuosoUserName,
                                        Properties.Settings.Default.VirtuosoPassword);

    m_gGraph = new Graph();
    m_lstObservations = new List<string>();
    cmbChange.SelectedItem = cmbChange.Items[0];
}
```

Figur 29. Ett objekt för att ansluta till OpenLink Virtuoso skapas.

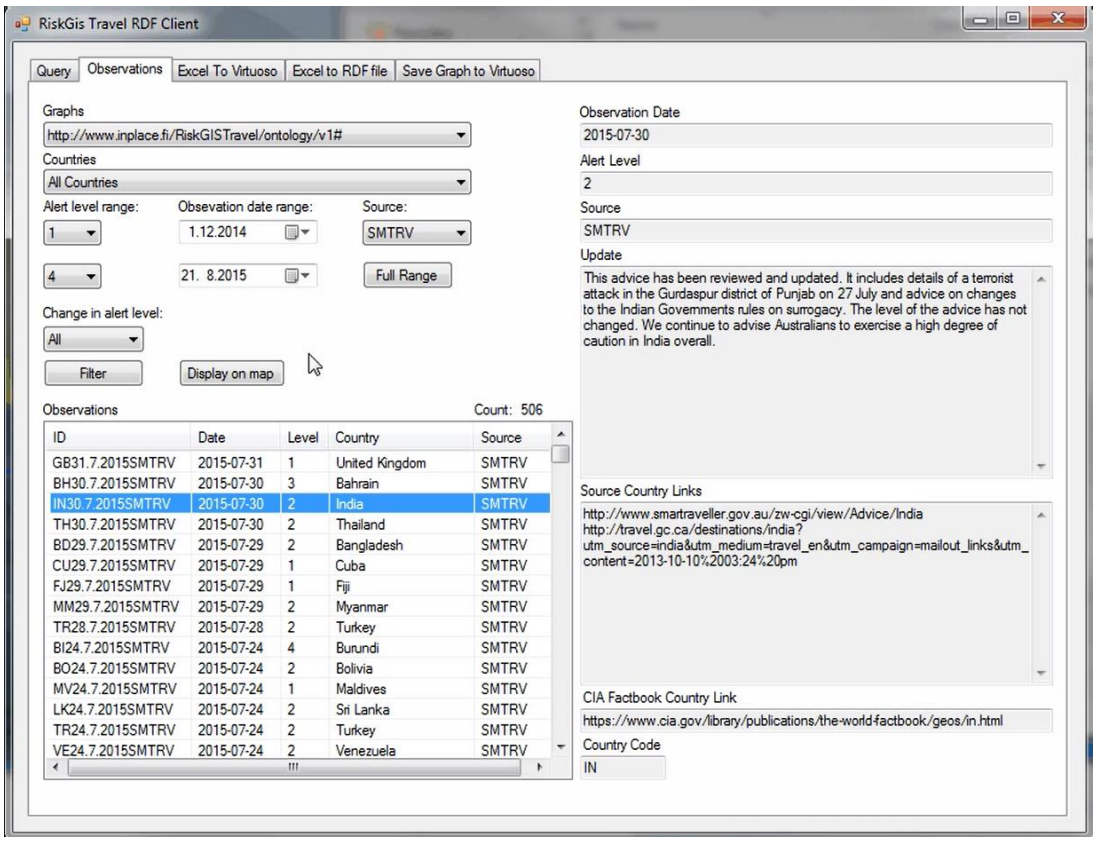
Samma objekt användes allra först till att hämta alla grafer som finns sparade i databasen. Dessa grafer listas i applikationen och användaren kan då välja vilken graf som skall hämtas, och efter att ha valt en graf sparas den i objektet *m_gGraph* som syns i figur 29. Grafen innehåller observationer och länderna som är kopplade till dem.

Alla länder hämtas med hjälp av metoden *GetTriplesWithPredicate()* som finns hos objektet *m_objGraph*. Genom att definiera predikatet *countryName* som parameter i metoden kommer tripletterna som returneras att innehålla ländernas id som subjekt och deras namn som objekt. På det sättet kan ländernas namn visas i en dropdown-lista och deras id sparas i objekten i listan, så att det när man valt ett land går att använda landets id för att hitta alla observationer som är kopplade till det landet. I figur 30 syns det hur listan med tripletterna skapas samt hur dropdown-listan får sina värden från denna lista.

```
IUriNode nodePredicate = m_gGraph.CreateUriNode(":countryName");

ComboboxItem itmCountry;
List<Triple> lstObservations = new List<Triple>(m_gGraph.GetTriplesWithPredicate(nodePredicate));
foreach (Triple t in lstObservations)
{
    itmCountry = new ComboboxItem();
    itmCountry.Name = t.Object.ToString();
    itmCountry.Value = t.Subject.ToString();

    cmbCountries.Items.Add(itmCountry);
}

cmbCountries.SelectedIndex = cmbCountries.Items.IndexOf(itmAllCountries);
```

Figur 30. Alla länder i den valda grafen hämtas och listas.

När länderna har listats har användaren möjlighet filtrera observationerna enligt land, datum, säkerhetsnivå och källa. Dessa parametrar sparas i objekt och används i SPARQL-satser för att hämta alla observationer som möter de vals om användaren har gjort. Efter filtreringen presenteras användaren med observationer som går att få mer information om när de markeras med musen, vilket kan ses i figur 31.



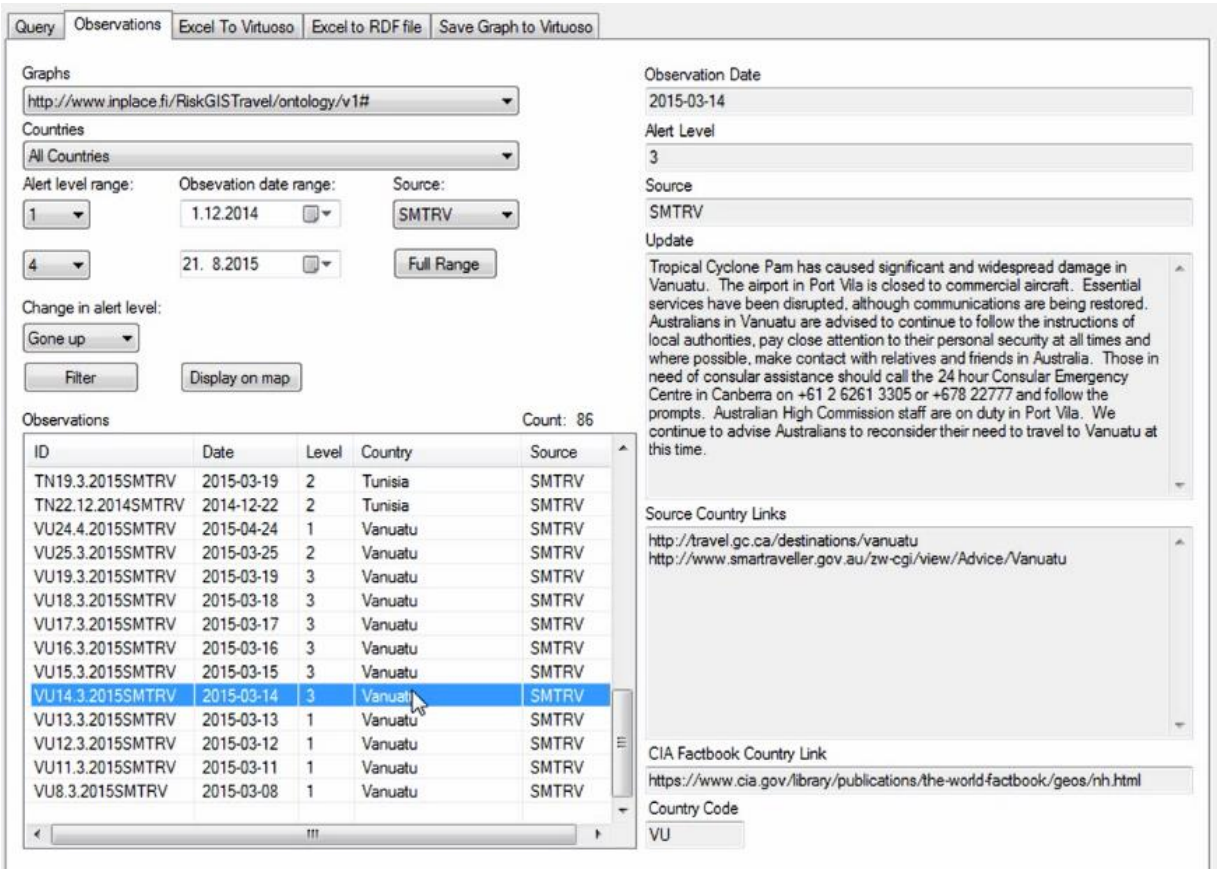
Figur 31. De filtrerade observationerna presenteras.

I kod är resultatet av filtreringen ett SparqlResult-objekt som innehåller alla tripletter som returnerades av SPARQL-förfrågningen. För att lista observationerna som syns i figur 31 skapades *ListViewItems* som fick sina värden från tripletterna. dotNetRDF gör det enkelt att hantera tripletter och hämta deras värden med hjälp av metoder som finns hos objekt av klassen SparqlResult.

En funktion som skulle finnas med i applikationen var att identifiera de länder där säkerhetssituationen har ändrat inom det tidsintervall man har valt. På detta sätt kunde man t.ex. se var i världen under en viss tid det hade blivit farligare att resa.

Detta löstes genom att först filtrera alla observationer enligt det angivna tidsintervallet. Till näst plockades alla länder som hörde till observationerna ut genom att göra SELECT DISTINCT-förfrågan, som hämtar bara ett exemplar av varje land, varefter en lista med dessa länder skapades. Listan gicks igenom sorterat efter datum och den första observationens säkerhetsnivå för varje land sparades och de andra jämfördes mot denna.

Beroende på om någon observations nivå hade ändrats jämfört med den första visste man att nånting hade hänt som påverkat landets säkerhet. De länder vars säkerhetsnivå hade ändrats sparades sedan i en ny lista. Observationerna för dessa länder kunde sedan presenteras och på så sätt fick användaren se bara de länder vars säkerhetssituation hade ändrats. Ett exempel på detta syns i figur 32.



Figur 32. De länder som har blivit farligare att resa till under den angivna tiden listas.

I figuren har observationerna filtrerats så att de länder vars säkerhetsnivå någon gång under den angivna tiden har gått upp. Detta syns i och med flervalslistan med texten ”Gone up”. Som exempel syns att Vanuatu den 14 mars 2015 drabbats av cyklonen Pam vilket gjorde att säkerhetsnivån höjdes från ett till tre.

Med hjälp av denna filtrering går det lätt att identifiera länder där det skett någon katastrof. Detta är värdefull information för t.ex. affärsmän eller folk som planerar att åka på semester. I nästa kapitel beskrivs hur denna information visualiseras så att är ännu lättare att ta till sig informationen.

5.6 Sändning av data för visualisering

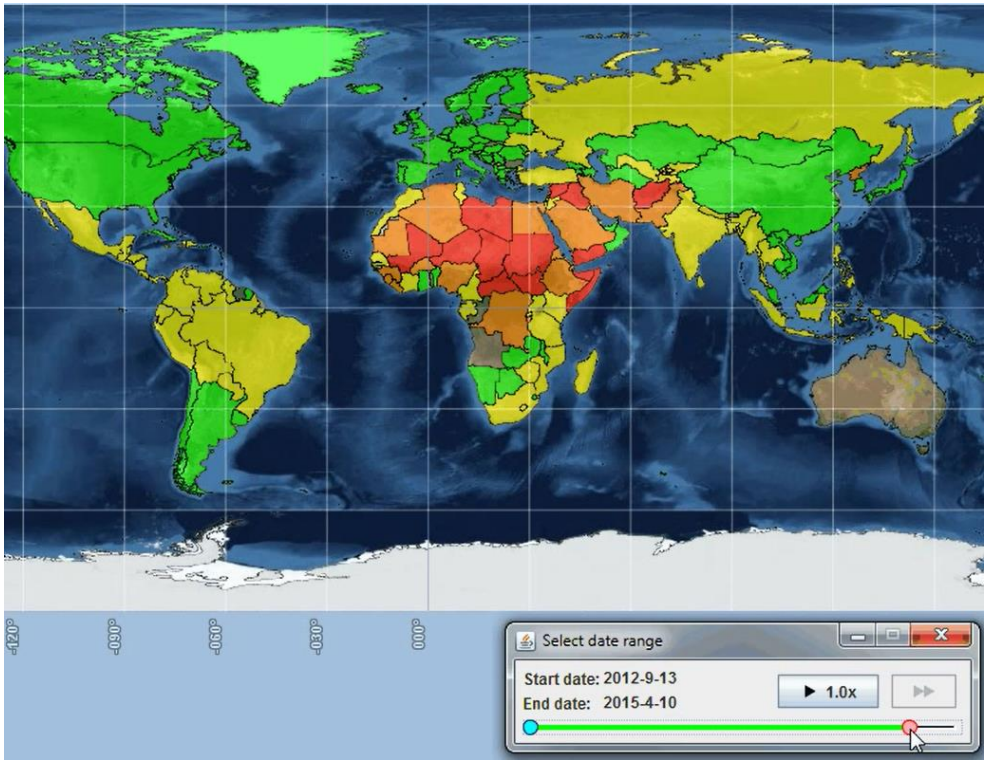
Det sista som krävdes av applikationen var att kunna sända de filtrerade observationerna till en av företagets applikationer där de visualiserades på en världskarta. Detta gjordes genom en TCP-anslutning mellan applikationerna. TCP står för Transmission Control Protocol, och är ett protokoll som definierar hur data skall sändas mellan elektroniska apparater. Företagets applikation fungerade som server och applikationen som skapats i detta arbete fungerade som klient. [41]

Genom att använda .Net-bibliotekets klass TcpClient skapades ett objekt som anslöt till servern. Det som behövs för att ansluta till servern är IP-adressen och porten som servern lyssnar på, detta syns i figur 33. Efter det skapades ett .Net NetworkStream-objekt från anslutningen som användes för att skicka data med. Informationen som serverapplikationen behövde för att visualisera observationerna var landets id, säkerhetsnivån och datumet för observationen.

```
    m_objTcpClient = new TcpClient(
        Properties.Settings.Default.RGTravelTcpIP,
        Properties.Settings.Default.RGTravelTcpPort);

    // Get a client stream for reading and writing.
    m_objNetworkStream = m_objTcpClient.GetStream();

    // Translate the passed message into ASCII and store it as a Byte array.
    Byte[] data = System.Text.Encoding.ASCII.GetBytes(sObservations);

    // Send the message to the connected TcpServer.
    m_objNetworkStream.Write(data, 0, data.Length);
}
```

Figur 33. Observationerna sänds över TCP-anslutningen.

Dessa egenskaper skickades som en enda textsträng med semikolon som skiljetecken för varje observation och kommatecken mellan egenskaperna för observationerna. På detta sätt kunde serverapplikationen skilja på observationerna och deras egenskaper och visualisera dem.



Figur 34. Applikationen där observationerna visualiseras.

I figur 34 ses resultatet av visualiseringen. Tidsaxeln nere till höger ger möjlighet att se utvecklingen för länderna inom den valda tiden som observationerna filtrerades efter. Detta ger en bra överblick över utvecklingen av säkerhetssituationen i världen.

6 Resultat och diskussion

I detta kapitel presenteras resultatet av arbetet samt diskussion om hur arbetet framskred.

6.1 Resultat

Resultatet av arbetet blev en lyckad integrering av tekniker inom semantiska webben i ett existerande system. I och med detta fick företaget en startpunkt för vidareutveckling inom området. Med hjälp av applikationen som skapades gick det att konverterar tabelldata till RDF-data samt hämta och filtrera RDF-data. Applikationen kunde också användas för att sända data som filtrerats till en annan applikation där informationen visualiserades.

I det tidigare systemet fanns det ingen möjlighet att filtrera observationerna. Applikationen laddade automatiskt in alla observationer som fanns i SQL-databasen och visualiserade dessa. I och med applikationen som detta arbete bestod av finns det möjlighet att välja vilka observationer som skall visualiseras. Detta ger mer funktionalitet åt systemet då man nu kan välja inom vilken tidsram man visualiserar observationerna, samt identifiera tidpunkter då ett lands säkerhetssituation har förändrats.

6.2 Diskussion

Arbetet fungerade som ett pilotarbete för att integrera tekniker som används inom semantiska webben. Ingen i företaget hade bekantat sig desto mer med teknikerna och därför fanns ingen att ställa teknikspecifika frågor till. Detta i kombination med att boken jag läste som introduktion inte gav tillräckligt med kunnande för att komma igång med att skapa applikationen försvårade starten avsevärt. Efter att ha läst boken visste jag dock vilka tekniker jag skulle läsa mera om för att komma igång. Detta gjordes genom att söka i nätartiklar och på olika forum.

De största problemen jag stötte på berodde på att jag inte hade en tillräckligt bra helhetsbild av hur tekniken fungerade och vad användningsområdet var. Eftersom konceptet med semantiska webben och länkad data var nytt både för mig och för arbetsgivaren gick det för mycket tid åt att komma underfund med det. Istället för att efter att ha läst boken börjat söka

information på internet tror jag det skulle ha varit bättre att ha lånat en annan bok som gett mera guidning för hur man kommer igång med att programmera applikationen. På detta sätt hade mindre tid gått åt att försöka hitta information på nätet och mera tid hade funnits att sätta på applikationen och dess funktioner.

Trots det så efter att ha fått tillräckligt med kunskap om teknikerna kom arbetet bra igång. Funktionaliteten hos applikationen som skapades diskuterades hela tiden med uppdragsgivaren och projektet avgränsades under arbetets gång för att hållas inom tidsramarna. Från början var det tänkt att också GeoSPARQL skulle ingå i arbetet, dvs. möjligheten att spara, hämta och hantera geografisk information [36]. Men detta skulle ha dragit ut på tiden för mycket och därför valdes det bort. Jag skapade dock dokument som beskrev tekniken vilket kan fungera som grund för vidareutveckling för att implementera tekniken i applikationen.

En av sakerna jag fick lära mig under arbetets gång var vikten av att dokumentera allt som gjordes. Inte bara med tanke på detta examensarbete utan också för företagets del så att de inte behöver lägga all tid på sökning av information som jag har gjort, utan har färdigt material att utgå från. Också skapandet av applikationen betyder att de inte behöver göra alla misstag och lösa de grundläggande problem som jag stötte på utan har en fungerande lösning att utgå från vid vidareutveckling.

Genom detta arbete fick jag erfarenhet av att jobba självständigt med projekt. Jag fick lära mig vikten av att hålla möten regelbundet för att se hur projektet framskrider och att hela tiden bolla idéer med uppdragsgivaren. Jag har sett fördelen med att planera och få en klar helhetsbild över vad som skall göras före man börjar på med arbetet. Framförallt har det varit givande att få ha bidragit med en fungerande applikation som nu används i företaget. Jag är tacksam för detta arbete som InPlace Solutions erbjöd mig och som har fått mig att växa som programutvecklare.

7 Källförteckning

[1] InPlace Solutions [Online]
http://www.inplace.fi/what-we-do/products.aspx [hämtat: 17.3.2016]

[2] RiskGis Travel [Online]
http://www.inplace.fi/media/1119/RGTravel_150401_World.png [hämtat: 17.3.2016]

[3] DuCharme, B., 2013. *Learnin SPARQL 2nd Edition,* O'Reilly Media, s. 38.

[4] DuCharme, B., 2013. *Learnin SPARQL 2nd Edition,* O'Reilly Media, s. 37.

[5] Allemang, D. & Hendler, J., 2008. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*, Morgan Kaufmann, s. 2-4.

[6] Semantic Web Stack [Online]
https://upload.wikimedia.org/wikipedia/commons/f/f7/Semantic_web_stack.svg
[hämtat: 5.11.2015]

[7] Semantic Web Architecture [Online]
http://www.obitko.com/tutorials/ontologies-semantic-web/semantic-web-architecture.html
[hämtat: 5.11.2015]

[8] Daconta, M.C., Obrst, L.J. & Smith, K.T, 2003. *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*, Wiley Publishing, Inc., s. 27-28.

[9] DuCharme, B., 2013. *Learnin SPARQL 2nd Edition,* O'Reilly Media.

[10] Daconta, M.C., Obrst, L.J. & Smith, K.T, 2003. *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*, Wiley Publishing, Inc., s. 195-198.

[11] Linked Data [Online]
https://www.w3.org/DesignIssues/LinkedData.html [hämtat: 11.11.2015]

[12] Illustration of Linked Data [Online]
http://www.bbc.co.uk/staticarchive/182def81ebb7559fba4d64b6e7d77fd34a4dd36b.png
[hämtat: 11.11.2015]

[13] Semantic Modeling [Online]
http://www.linkeddatatools.com/semantic-modeling [hämtat: 11.11.2015]

[14] Segaran, T., Evans, C. & Taylor, J., 2009. *Programming th Semantic Web*, O'Reilly, s. 63-64.

[15] Segaran, T., Evans, C. & Taylor, J., 2009. *Programming th Semantic Web*, O'Reilly, s. 64-68.

[16] Antoniou, G. & van Harmelen, F., 2008. *Semantic Web Primer 2nd Edition*, MIT Press.

[17] Antoniou, G. & van Harmelen, F., 2008. *Semantic Web Primer 2nd Edition*, MIT Press, s. 67-68.

[18] Allemang, D. & Hendler, J., 2008. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*, Morgan Kaufmann, s. 169-177.

[19] DuCharme, B., 2013. *Learnin SPARQL 2nd Edition,* O'Reilly Media, s. 61-69.

[20] Antoniou, G. & van Harmelen, F., 2008. *Semantic Web Primer 2nd Edition*, MIT Press, s 90.

[21] RDF Schema 1.1 [Online]
https://www.w3.org/TR/rdf-schema/ [hämtat: 2.4.2016]

[22] Ontology [Online]
http://semanticweb.org/wiki/Ontology [hämtad 2.4.2016]

[23] OWL Web Ontology Language Reference [Online]
https://www.w3.org/TR/owl-ref/ [hämtad 2.4.2016]

[24] Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition 1.3 [Online]
https://mariaiulianadascalu.files.wordpress.com/2014/02/owl-cs-manchester-ac-uk_-eowltutorialp4_v1_3.pdf [hämtad 3.4.2016]

[25] SPARQL 1.1 Overview [Online]
https://www.w3.org/TR/sparql11-overview/ [hämtat: 12.12.2015]

[26] Antoniou, G. & van Harmelen, F., 2008. *Semantic Web Primer 2nd Edition*, MIT Press, s. 128-129.

[27] Protégé Overview [Online]
http://protege.stanford.edu/about.php [hämtat: 14.12.2015]

[28] Visual Studio [Online]
https://en.wikipedia.org/wiki/Microsoft_Visual_Studio [hämtat: 4.4.2016]

[29] NuGet Overview [Online]
http://docs.nuget.org/consume/overview [hämtat: 4.4.2016]

[30] Using IntelliSense [Online]
https://msdn.microsoft.com/en-us/library/hcw1s69b.aspx [hämtat: 4.4.2016]

[31] dotNetRDF [Online]
http://www.dotnetrdf.org [hämtat: 4.4.2016]

[32] dotNetRDF Library Overview [Online]
https://bitbucket.org/dotnetrdf/dotnetrdf/wiki/UserGuide/Library%20Overview [hämtat: 4.4.2016]

[33] OpenLink Virtuoso [Online]
https://www.w3.org/2001/sw/wiki/OpenLink_Virtuoso [hämtat: 4.4.2016]

[34] OpenLink Virtuoso Overview [Online]
http://virtuoso.openlinksw.com/images/varch625.jpg [hämtat: 4.4.2016]

[35] Apache Jena [Online]
https://jena.apache.org/ [hämtat: 4.4.2016]

[36] GeoSPARQL [Online]
https://www.w3.org/2011/02/GeoSPARQL.pdf [hämtat: 6.4.2016]

[37] Segaran, T., Evans, C. & Taylor, J., 2009. *Programming th Semantic Web,* O'Reilly, s. 129-131.

[38] DuCharme, B., 2013. *Learnin SPARQL 2nd Edition,* O'Reilly Media, s. 53-55.

[39] SPARQL query with dotNetRDF [Online]
https://bitbucket.org/dotnetrdf/dotnetrdf/wiki/UserGuide/Querying%20with%20SPARQL
[hämtat: 8.4.2016]

[40] SPARQL 1.1 Query Language [Online]
https://www.w3.org/TR/sparql11-query/ [hämtat: 8.4.2016]

[41] TCP/IP [Online]
http://www.w3schools.com/website/web_tcpip.asp [hämtat: 8.4.2016]

[42] OpenLink Virtuso Infrernce Rules & Reasoning [Online]
http://docs.openlinksw.com/virtuoso/rdfsparqlrule.html#rdfsparqlrule [hämtat: 10.4.2016]