Description of computer tools

Digital appendix forming part of the data package for my thesis (Eide, 2012).

The data package can be found at: http://www.oeide.no/dg/dp/ (checked 2012-08-29)

Øyvind Eide

August 31, 2012

This appendix consists of additional information connected to chapters 5 and 6 of my thesis (Eide, 2012). Its main purpose is to describe the application and its use in a more technical way than what is found in the thesis, giving a better understanding of how the application was developed and used as part of the research. The overview given here can also be used as an introduction to the application if the purpose is to run it or to use the source code, as a whole or in parts.

Contents

List of Figures

List of Tables

1 Overview of the application

The structure of GeoModelText is based on the general framework of the experiments, as the application was written to make the experiments possible. Thus, the structure follows to a large degree the steps summarised in section 5.1.3. One purpose of this appendix is to explain how the transformations between the stages are implemented.

A primary model of a string of text, e.g. an interview, is the sum of the information imported from the source TEI files and the information added in the application as part of the manual modelling. Doing experiments on this model includes running rules of calculation on the model to see what results come out. One of these rules of calculation is to create a GML document based on the model. The Geography Markup Language is an XML encoding for the transport and storage of geographic information (Portele, 2007). In the context of GML, geographical information includes both the spatial and non-spatial properties of geographic features, but in my work I only use GML to express geographical vector data which can be expressed as a map image using GIS software.

The experiment is performed through the following overall steps:

1. The TEI version of the source text is imported and stored in the Java application *(text → primary model)*.

2. The TEI version of the person and place registers are imported and stored in the Java application *(text → primary model)*.

3. A window in the Java application is used to do the manual modelling. This results in the primary model *(text → primary model)*.

4. Windows in the Java application are used to add extra information, such as co-reference *(text → primary model)*.

5. Windows in the Java application are used to manually formalise the model step by step *(primary model → formalised model)*.

6. Algorithms in the Java application are used for computational step by step formalisation of the model *(primary model → formalised model)*.

7. Windows in the Java application are used to visualise parts of the model as well as the results of partial experiments *(formalised model → visualisation)*.

8. Export facilities are used to export data from the Java application in order to be analysed externally *(formalised model → analysis)*.

9. Export facilities are used to export experiment results from the Java application *(formalised model → results)*.

10. External analysis is done in the Oxygen® XML editor[1] *(formalised model → analysis)*.

11. External visualisation is done in the gruff RDF browser[2]

12. External visualisation and analysis is done in the qGIS software package[3]

Being my own programmer, I am able to do rapid cycles of software adjustments when I see the need for this. The main documentation of such adjustments and corrections is in the data package:[4] in the lab diary, as comments in the source code as well as in the bug tracking and feature request tracking systems in Sourceforge. All this information is available from the data package webpage.

1.1 Data

This is part of the process from text to primary model. The actual source document used in the experiment is a digital version of Schnitler (1962), created as part of the Documentation Project (Eide and Sveum, 1998) and converted to TEI P5 as a preparation for the current project. In addition, material created in an earlier project was included. This latter material was used in the form of a TEI P5 export from the original Common LISP application in which it was created (Eide, 2004).

In the application, a Document Object Model[5] structure is created based on the TEI XML file. This means that the whole XML document is stored as one Java object, and there are additional Java objects representing each of the XML nodes. An XML node is either an element, which are marked

[1] Webpage: http://www.oxygenxml.com/ (checked 2011-11-20)

[2] Webpage: http://www.franz.com/agraph/gruff/ (checked 2011-11-20). *(formalised model → visualisation)*.

[3] Webpage: http://www.qgis.org/ (checked 2011-11-20) *(formalised model → map)*.

[4] Data package URL: http://www.oeide.no/dg/dp/ (checked 2012-08-29)

[5] DOM, webpage: http://www.w3.org/DOM/ (checked 2011-11-28)

with XML tags in the TEI document, or a text leaf nodes, which are the strings in the TEI document.[6]

When we discuss a transfer of information from a TEI document to a Java application, a change in the level of abstraction is implied which makes the concepts used not fully interchangeable.

```
<p xml:id="Schn1_44714">1te Viidne, heeder <name type="person"
xml:id="Schn1_44720">Ole Nilsen</name></p>
```

Figure 1: TEI document fragment. **hi** tags, representing superscript and italics, are removed to improve readability.

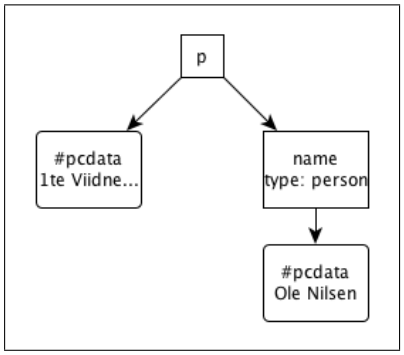


Figure 2: The tree structure of the TEI document fragment.

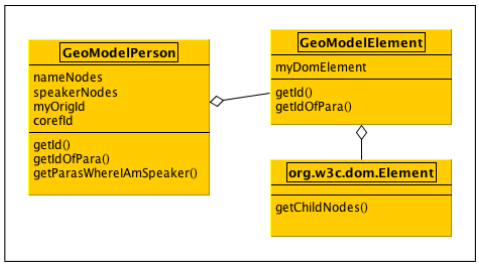


Figure 3: Simplified UML model of a fragment of the object structure in the Java application representing the relationship between the paragraph and the person name from figures 1 and 2.

An example of a TEI document can be found in figure 1. TEI documents are special cases of XML documents. Each unit in the tree structure is an element or a text node, as shown in figure 2. These elements and text nodes each represent a node in the tree structure. When the TEI document is imported into the Java application the elements are still elements, but instead of being represented in a text file by start and end tags they are now represented as objects. Each element is an object, from the outer **?XML** element to the leaf nodes. These objects are of three types:

- Document, DOM type 9 (the root node, that is, the full XML document)

[6]The DOM tree also includes attribute values and some XML structures I do not use.

- Element, DOM type 1[7]
- Text, DOM type 3

Seen as representations of XML elements, such objects can be said to encapsulate other objects, but seen as Java objects they are all at the same level, as parts of the structure shown in figure 3. An important consequence of this is that the structure can be processed not only in an XML like hierarchical mode, but also without taking the XML hierarchy into consideration. This is the freedom of the database model described in section 5.2 of my thesis (Eide, 2012).

1.2 Application

The computer application is developed in Java version 1.6.[8] In addition to the standard libraries, the Jena package is used for storing and accessing RDF triples.[9] A set of tools developed at the Unit for Digital Documentation[10] by Jon Holmen and me was included in the project as a separate library and further developed. The application is called GeoModelText and is maintained in Sourceforge.[11]

The core unit of the application is the paragraph level element. The element types included as core unit elements include paragraphs (TEI: **p** elements) and tables (TEI: **table** elements). In figure 4 the contents of one such element is shown in the text box on the left hand side. All names and dates marked up in the text are included in the application as a starting point for the model building. Through the modelling process, other entities are added to the imported ones. The entities are divided into the following groups:

TEI elements These elements were there from the import of data into the application from the TEI files. They are not mutable, nor is the core information connected to them, with the exception of the co-reference information. If anything else is incorrect, it must be changed in the

[7]The attributes are connected to the Element objects.

[8]Webpage for API specification: http://docs.oracle.com/javase/6/docs/api/ (checked 2012-02-07).

[9]Webpage: http://jena.sourceforge.net/ (checked 2011-11-28)

[10]Webpage: http://www.edd.uio.no/ (checked 2011-11-20)

[11]Webpage: http://sourceforge.net/projects/geomodel/develop (checked 2011-11-20)

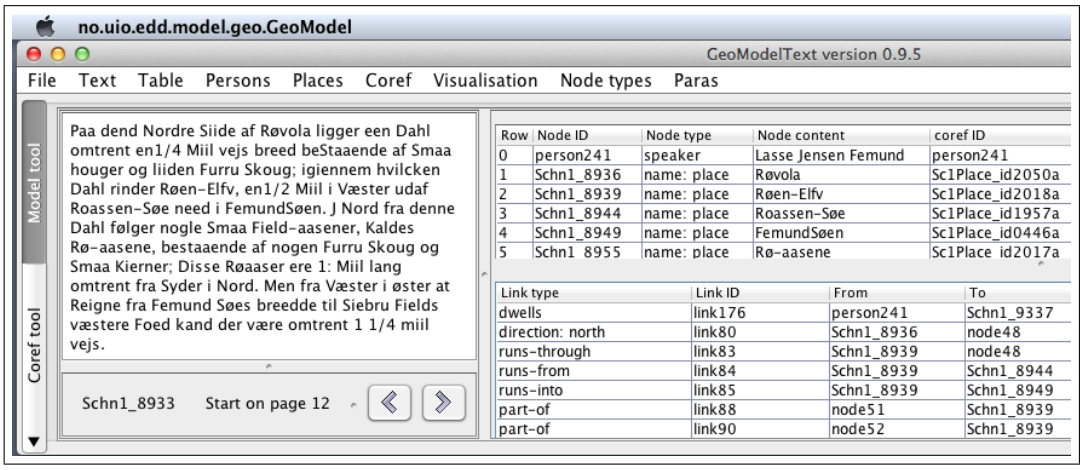


Figure 4: Screenshot from the modelling tool.

underlying TEI documents before the application is restarted, which leads to a re-reading of the TEI files. The following types exist:

- person names
- place names
- dates

speaker If there is a person responsible for the oral statement upon which the written text is based, then it is shown in the first line. The person objects representing the speaker of each paragraph level element are read from the person register. They form part of the core data and are not mutable.

added nodes These are nodes added by the user, having full control over them. They are used to store strings referring to places, persons, events, or any other types of entities the user choose to record. They are stored in an external file called **rdfVersion.xml** between sessions.

On the bottom right of figure 4 the properties, that is, the links between the entities, are found. The properties are all entered manually by the user. Each of them establish a triple representing a relationship between two other objects, the domain and the range. Domains and ranges can be places,

persons, or other properties. When domains or ranges are other properties, this means that the whole triple around the property is the domain or range. An example of this is shown in sentence 6.4 in Eide (2012, 177).

Each property has a link type detailing what the property means. The information in this link type is used in the formalisation of the relationships later in the process. Properties as well as mutable entities can be created and removed by menu choices. All menu choices used frequently are also accessible as keyboard shortcuts.

The properties discussed here represent explicit links. There is also another type of links between entities: co-references.

1.3 Co-reference

There are two ways to create co-references in the computer application. One is local to each entity and can be seen on the screenshot in figure 4. An entity can be co-referred to another one by entering the ID value of another entity in the **Coref ID** column of the table line for the entity. This will result in connections to all other entities co-referred with either the co-referer or the co-referred as well. In the same window there is also a menu choice giving the user some basic information about the selected entity, including a full list of all direct and indirect co-references.

The second way to record co-references was developed in reply to my need for an efficient way of entering large quantities of co-reference links into the system. The goal was to make a tool where all detectable co-references pairs should be entered as quickly as possible. The implementation was based on the fact that a good place name register was established when the printed edition was published in 1962.

A window was created where all place names in the original TEI document is included on the top and each record from the place name register was put on the bottom, as seen in the screenshot in figure 5.[12]

A paragraph of context around any place name occurrence may be accessed, presenting the text tagged or un-tagged at user discretion. This window was used to mass-connect most of the place names in the text to register entries, as described in Eide (2012, 124–130). For practical reasons,

[12]The place register had to be used with some care because in the creation of it, attempts were made to correct erroneous information in Schnitler's text, whereas my aim is to use what was said in the text, correct or not.

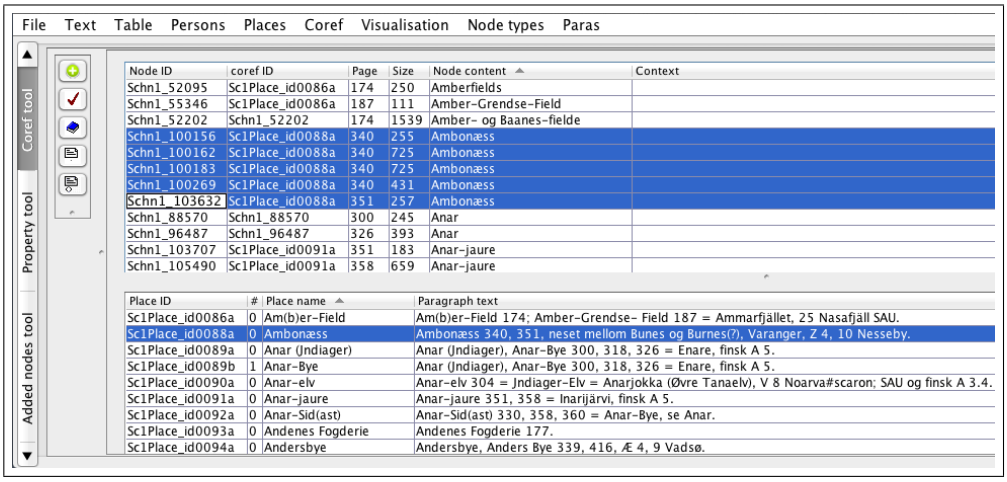


Figure 5: Screenshot from the co-referencing tool. The records from the place register are at the bottom part of the screen, whereas the strings from the place name elements, that is, the entities of the model, are at the top. When **Node ID** is equal to **corefID**, the co-reference is not set for the entity in question. It could still be part of a network, however, because another object may have it as its co-reference. The underlying system stores two way links, but the back links are not directly accessible to the user.

each set of co-references is stored in my system with one value that is presented as the main value, based on a deterministic calculation. If this main value is seen as the thesaurus record, one may say that the implementation is close to the use of a thesaurus or gazetteer. To the user the co-references are still presented as a set of networks of co-reference links.

2 Model building

In the model building, the results of the close reading are stored as triples in a fact database. They only stored at this stage, not analysed based on any idea of truth. They are, however, integrated in the form of co-reference resolution, as we just saw. In the series of distinct stages forming parts of the stepwise formalisation the balance between tasks done by me as a user and by the computer varies from stage to stage:

1. Create the primary model. Man and computer.
2. Formalise into XML fragment. Man.
3. To RDF. Computer.
4. To GML. Computer, tuneable parameters.

In order to use GIS software for map visualisations the data must be expressed in a way which is readable by the software. That is, it must be exported as vector data. The format I use is GML.[13] What is lost in the process of creating the GML data is made visible by the system, it is one of the fall-off stages. In what follows, all the stages of the stepwise formalisation will be described. This presentation should be seen in light of chapter 5 of the thesis (Eide, 2012).

2.1 Parsable contents

This is part of the process from primary model to formalised model. The property window is a tool where each property is listed with information about its source and target entities. The window includes a field for **Parsed type**. The parsed type is a key to the manual part of the stepwise formalisation. It is used to store an XML fragment containing a more formalised version of the original property type. This XML fragment is entered manually by the user. The fact that the information shown in the property window is formatted as a table which can be sorted on any field made the process quite streamlined. Examples of how some properties are formalised can be found in table 1.

Property type	**Parsed type**
part-of	<spacePartOf/>
distance: some 1/4 mile	<spaceDistanceMileOld>0.25</spaceDistanceMileOld>
distance: a good village Mile	<spaceDistanceMileVillage>1.1</spaceDistanceMileVillage>

Table 1: Examples of formalisation of properties.

A similar window exists for added nodes, also with a field for **Parsed type**, with the same sorting options on the table. Added nodes are used for

[13] What GML is was explained on page 3.

a number of different things. Most of these uses do not call for this type of formalisation. Examples of the latter include strings referring to places, persons and events. Other added nodes are in need for further formalisation in a similar way to the properties discussed above. These include length, width and orientation of places.

In parsed fields, numbers are usually entered, but the scale is kept: new mile, old mile, etc., so that different re-calculations can be made. I do not interpret what an old or undefined mile, or a rifle shot, means and how they could be re-calculated to kilometres at this stage.

2.2 Algorithms for stepwise formalisation

This is part of the process from primary model to formalised model. In addition to the data capture tools discussed in the previous section, the system also includes a number of non-interactive algorithms for stepwise formalisation. These algorithms start where the interactive ones end, namely, in the co-referred version of the model in which data have been entered in the **Parsed type** field for some of the entities and for many of the properties.

From the user's point of view, the process is started by running one of the menu choices **Traverse graph**, **Traverse speaker**, or **Traverse speaker paragraphs separately**. The screen output resulting from these subroutines is a list of places with the relationships between them. More importantly, each of the subroutines silently output files of three types based on the part of the model being processed:[14]

- An RDF model linearised as an XML file.
- GML files representing spatial vector data, which can be expressed as map images.
- A text file documenting problems. One important purposes of the text file is to document fall-offs, another one to show features that need to be implemented.

[14]These files will not be stored in the demo version of the application. Refer to the source code to find out how to get to these files.

2.2.1 Towards the RDF triplestore

The first phase of the formalisation algorithm results in an RDF model. No data can be added to the RDF model without being part of a triple. Each of the links existing in the original model are interpreted, one after the other, when the triplestore is built up. The properties all represent triples, consisting of domain, property and range. At this stage, the main value of a co-reference set is chosen for each entity, that is, for domains and ranges. This means that all differences between the various names referring to the same place fall off at this stage.

The algorithms formalising the model can be run for a part of the model only, e.g., the part based on statements from one witness only or the part based on one or several selected paragraphs. At this stage only entities which are either the domain or the range of a triple where either the domain or the range is connected to one of the paragraphs to be included are considered. The ones which are not will fall off, quite naturally, as they are not part of the section of the model for which the algorithm is ran.

The **between** relationship does not fit very well into my triple based system. This led to some problems which had to be solved. The relationship was expressed as a set of two binary relationships, that is, if A is between B and C it is not expressed as sentence 1 but rather as sentence 2. This solution was somewhat clumsy, but it worked.

$$between(A, B, C) \tag{1}$$

$$between(A, B) \wedge between(A, C) \tag{2}$$

In this process the parsed type of entities and properties is retrieved and interpreted. This includes finding values for spatial relations. If a property has no parsed type it falls off and is written to the text file reporting problems. The same will happen if the parsed type is not possible to identify. This is different for entities, however; they do not need to have parsed types to be formalised, as we saw in section 2.1 above. A place, a person, an event, and other types of entities are included without any manually entered parsed type, as long as they are domain or range in at least one included triple.

Everything not fallen off at this stage is included in the RDF model. So everything we have not lost is expressible in RDF. If does not follow from his, however, that everything expressible in RDF from the original model is

still with us. Things that would actually have been possible to express in RDF could still have fallen off.

2.3 From RDF to GML

This is part of the process from formalised model to vector data. The GML files produced in this project contain coordinates for a set of features, together with place names and other types of information linked to each of them. Each feature represents a place. So the GML files store no relationships between places or other topological information. Every place needs its own coordinates. Even if the relationships are not expressed directly in the vector data they are still the key to the creation of the coordinates used to represent the places on the maps.

In order to express a relationship between two places as coordinates for the two of them, we need to go through at least the following steps:

1. Decide if place A is to be expressed as a point, line or polygon.
2. Decide which centre coordinates place A should have.
3. Decide what the relationships to B means; that is, what do "east" and "2 miile" mean? Examples of answers: 90° for the former, 22,400 meters for the latter.
4. Decide if place B is to be expressed as a point, line or polygon.
5. Decide which centre coordinates place B should have, based on 2 and 3.

In this sketch of an example algorithm, we see that only operation 5 is based on a calculation in a strict sense. Put differently: in order to start doing calculations, we need to make many decisions. Many choices must be made in order to continue our movement towards the map.

The algorithm outlined above will need to be used in an iterative way. In order to make a map based on the information we have, things must be adjusted and re-adjusted through several rounds. The following steps take the iterative principle into account:

1. Set start coordinates. If this is the first point, use $(0, 0)$.

2. Find form indicators, make form, plot the feature for the starting entity. If there are no length and width: the start entity is represented by a point.

3. Find all linked entities.

4. Set coordinates for linked spatial entities. Types of properties will decide how the new coordinate will be for each linked entity:

   (a) distance and angle known: point

   (b) angle known and vague or unknown distance: select a point on a line (arbitrary distance)

   (c) distance known and vague or unknown angle: select a point on a circle or part thereof (arbitrary angle)

   (d) both vague: select a point in a filled circle or section (arbitrary distance and angle)

   (e) part of: put the range inside the domain, enlarging the domain if necessary

   (f) between: put the domains of the two between expressions on either side of the range

   (g) both unknown: no connection.

5. Include non-spatial entities as a list of properties to the place.

6. Go to any connected point and repeat.

The outline for this algorithm shows clearly how important the choices which must be made are. Note also that strictly speaking, 4a, 4b, and 4c do not exist, only 4d through to 4g. The former three can, however, be created based on choices. By changing the choices, different possibilities can be tried out (Eide, 2012, 159–162).

2.3.1 Implementation

In stage 1 and 2 of the implementation of the algorithm outlined above, a set of map objects are created. A simplified data model for these types can be found in figure 6. Each place reference read from the RDF model is created

as a **GeoPlace** object. Relationships are created between the place and all the other places it has relationships to in the RDF model. A relationship is expressed as an object of the class **GeoRelationship** and it connects two **GeoPlace** objects. All the aspects of the relationship, such as the distance between the places, the direction between them, and if one is part of the other, are then read into the **GeoRelationship** object. Each **GeoPlace** object stores information about the length, width, and orientation of the place, as well as the name, ID and other auxiliary data.

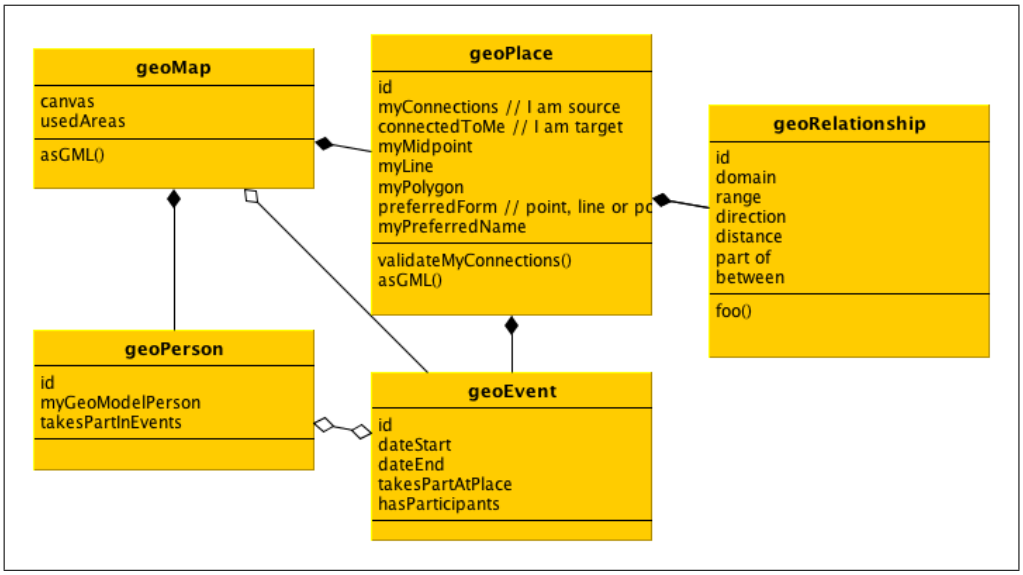


Figure 6: Simplified UML model of the system for storing spatial information. These classes export the GML which form basis for map creation.

Each **GeoPlace** object is also given a layer. One layer represents one map. Before the details of the relationships between the places have been interpreted into a common spatial model, each layer consists of one place. Before the spatial relationships are included in the vector data, each place rests alone in its layer. So even if the distance and direction between two places are given numeric values stored in the **GeoRelationship** objects, e.g., 90° and 4000 metres respectively, these numbers have not yet been used to connect the places spatially.

When all the places and their relationships are included in the map system, the process of interpreting spatial features into spatial co-ordinates starts. This is the implementation of the algorithm from pages 13–14. It

consists of the following steps:

1. The measurements for the place is interpreted and spatial objects are created and connected to the **GeoPlace** object. A point is always made as the centre for the place; if there are length and width a polygon is also made. Lines are not used, rivers are seen as narrow polygons.

2. All directions and distances are interpreted. If the **GeoRelationship** connecting two **GeoPlace** objects have direction or distance or both, the two places are included into the same layer. In that case one of the places is moved to fit the location relative to the other, as is all the other places in its layer.[15] The layers are then combined.

3. All **part of** relationships are interpreted and layers are combined in the same way as above. Any place not big enough for an object being part of it is expanded by a factor defined for each experiment. If a place which was previously only expressed as a point has parts, it is expanded to a standard polygon.

4. All **between** relationships are interpreted, putting any disconnected set of places to pre-defined directions opposite the place in between, at a pre-defined distance. Any relationship between two or all of the places already existing is taken into consideration.

5. Places are rotated as specified in their directions.

6. A final step of manual corrections. When such corrections are done they strictly adhere to explicit rules. The reason for human intervention is that some of the rules needed in the map production are too time consuming to implement for this project. The reason is never a lack of rules.

It is not important that the GML file is written by the computer program, the important thing is that it is written according to a rigorous interpretations of the input data. Thus, and in adherence to the 80/20 rule, I avoid solving algorithmically difficult problems occurring only a few times. When studying GeoModelText, it is important to remember that the algorithm in

[15] As we there are no absolute locations in this system any place can be moved around freely as long as all places it already has a spatial relationship is moved as well.

steps 1–5 above are only partly implemented, they work only for very simple structures. This must be changed if this research implementation is later to be transformed into a production system, or if the further research outlined in Eide (2012, 272–273) is to be done.

2.4 Map based geographical descriptions

In the modelling window seen in figure 4 on page 7 a geographical description can also be entered. This is a description of possible coordinates for the place based on specified maps. This was used in a small test of mapping onto pre-existing maps. This test did not lead to any significant results.

3 Visualisation

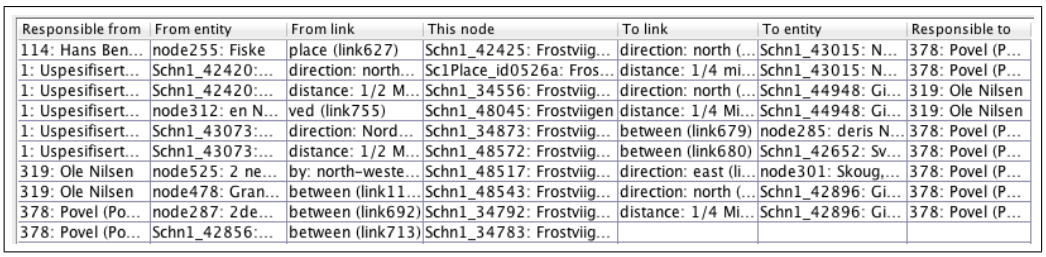
In the research described in chapters 5 and 6 of the thesis (Eide, 2012), several types of visualisation are used, text based as well as figures and images such as graphs and maps. Visualisations were also used in the development process, e.g., in order to monitor how changes applications influenced data conversion. In this section I will give an overview over the visualisation methods used seen from a technical perspective.

3.1 Internal visualisation methods

Several types of entities can be visualised in the computer application. There are five different menu choices for visualising persons. They are used for listing one or all persons with various level of detail and aggregation in the information included. The visualisations are shown in separate windows, and the data can easily be exported for other use.

For places there are also five choices. One of them is a full listing of all places with their co-references. A table of all paragraphs with a set of statistical data and an excerpt from the text is also available, as is versions covering only selected paragraphs or the paragraphs based on the statements made by one witness only. The statistical data presented include the amount of text and the number of person and place names, dates, added nodes, as well as links from and to entities of the table. Added nodes can also be listed in a tabular format.

I implemented one visualisation tool focusing on the properties. A screenshot can be found in figure 7. This tool can be seen as a visualisation of the network at the lowest level of detail. It is harder to understand than system using a graphical libraries, but it is able to present more information at the screen at once. In an application only meant for myself as a user the fact that it is difficult to understand is not a problem, as I know how it works. This shows clearly how the development of tools within a project is different from developing systems for general use.



Responsible from	From entity	From link	This node	To link	To entity	Responsible to
114: Hans Ben...	node255: Fiske	place (link627)	Schn1_42425: Frostviig...	direction: north (...	Schn1_43015: N...	378: Povel (P...
1: Uspesifisert...	Schn1_42420:...	direction: north...	Sc1Place_id0526a: Fros...	distance: 1/4 mi...	Schn1_43015: N...	378: Povel (P...
1: Uspesifisert...	Schn1_42420:...	distance: 1/2 M...	Schn1_34556: Frostviig...	direction: north (...	Schn1_44948: Gi...	319: Ole Nilsen
1: Uspesifisert...	node312: en N...	ved (link755)	Schn1_48045: Frostviigen	distance: 1/4 Mi...	Schn1_44948: Gi...	319: Ole Nilsen
1: Uspesifisert...	Schn1_43073:...	direction: Nord...	Schn1_34873: Frostviig...	between (link679)	node285: deris N...	378: Povel (P...
1: Uspesifisert...	Schn1_43073:...	distance: 1/2 M...	Schn1_48572: Frostviig...	between (link680)	Schn1_42652: Sv...	378: Povel (P...
319: Ole Nilsen	node525: 2 ne...	by: north-weste...	Schn1_48517: Frostviig...	direction: east (li...	node301: Skoug,...	378: Povel (P...
319: Ole Nilsen	node478: Gran...	between (link11...	Schn1_48543: Frostviig...	direction: north (...	Schn1_42896: Gi...	378: Povel (P...
378: Povel (Po...	node287: 2de...	between (link692)	Schn1_34792: Frostviig...	distance: 1/4 Mi...	Schn1_42896: Gi...	378: Povel (P...
378: Povel (Po...	Schn1_42856:...	between (link713)	Schn1_34783: Frostviig...			

Figure 7: Screenshot from the table visualisation tool.

The table used in this visualisation has seven columns. In the middle column the node in focus is presented with all its co-references below. To the right all the property-entity pairs are listed for triples for which the selected entity is the domain, with the person responsible for each of them. To the left a similar set is presented for triples in which the selected entity is the range. By selecting a line and clicking the left or right button, the focus is moved to the entity of choice. The visualisation can be reduced to contain the information from the paragraphs based on the statement from one person only.

Data from all these visualisations can easily by copied to external applications for further analysis. In addition to that, specific export systems for a number of open data formats are included. These export systems are used to export data which can then be imported into external applications.

3.2 External visualisation methods

As we saw in the previous section, some tools for visualisation are included in the application. Earlier attempts to display the networks created in the modelling work in a graphical form as part of the application turned out to

be too time consuming. The use of external tools was a more rational way of developing network and map visualisations. Three data formats are exported for visualisation purposes:

RDF data are used to express the networks of entities and properties. They are visualised using the gruff software package.[16]

SVG[17] data are used for the documentation of rooms of possibilities. They are displayed using the Safari web browser.[18]

GML data are also used for maps, including the more complex ones. They are displayed and interacted with using the qGis GIS package.[19]

In the use of GIS in this project I include vector data only, no raster data are included. GIS introduces a middle layer of vector data that have the potential for being visualised as maps but are not maps in themselves.

References

Eide, Ø. (2004). *Fra SGML til begrunnede påstander om verden : et system for analyse av geografiske resonnementer uttrykt i historiske tekster*. Master thesis in Språk, logikk og informasjon, University of Oslo.

Eide, Ø. (2012). *The area told as a story. An inquiry into the relationship between verbal and map-based expressions of geographical information*. London: PhD thesis, King's College London.

Eide, Ø. and T. Sveum (1998). *Dokumentasjonsprosjektet ved Universitetsbiblioteket i Tromsø. Rapport.* Tromsø: Universitetet i Tromsø, Tromsø museum.

Portele, C. (2007). *OpenGIS® Geography Markup Language (GML) Encoding Standard. Version: 3.2.1.* Open Geospatial Consortium Inc. GML 3.2.1 has been published as ISO 19136:2007.

[16]The RDF is stored in an XML linearised form in the exported RDF-XML file. For the gruff package, see footnote 2 on page 4.

[17]Scalable Vector Graphics, webpage: http://www.w3.org/Graphics/SVG/ (checked 2012-02-07)

[18]Webpage for Safari: http://www.apple.com/safari/ (checked 2012-02-07)

[19]On qGis, see footnote 2 on page 4.

Schnitler, P. (1962). *Major Peter Schnitlers grenseeksaminasjonsprotokoller 1742–1745. 1.* Oslo: Norsk historisk kjeldeskrift-institutt. The reference to this book is shortened to **S1** in the main text.