A Prototype to Extract and Visualize Information from Car Accident Reports in Swedish

Per Andersson

Abstract

Per Andersson, Lunds Tekniska Högskola
Text-to-scene conversion of car accident reports

This Master's Thesis is part of the CarSim (Dupuy et al. 2001) project that analyzes text reports of car accidents and visualizes them in a three dimensional graphical simulation. There are two main parts in the system. An information extraction module analyzes the text to find the vital information about the accident and a graphical simulation module visualizes the information. An XML template that is filled with information about an accident in a structured way provides a link between the two modules.

The information extraction module fills the template and the graphical simulation module reads it and uses the information to visualize the scene. This part of the system is designed to be multilingual. Before, two versions had been developed: one to analyze texts in French and the other for English texts.

This report describes the modifications I brought to the information extraction module and the modules I developed:

- A graphical user interface, which now integrates uniformly the language analyzers for the texts in French and English as well as the new analyzer for the texts in Swedish.
- An analyzer, which extracts information from Swedish texts about the scene of the accident and the actors of the collision.

The analyzer detects the road configuration, the static objects and the dynamic objects. It uses a domain ontology describing the roads, obstacles such as trees and the vehicles. Regular expressions are used to match them in the texts. A local parser is used to identify the collision verbs and the actors of the accident. The analyzer integrates the Granska part-of-speech tagger and phrase-structure rules to describe noun and verb groups. I have developed and tested the system using a corpus of 209 reports gathered from articles describing car accidents that were published in Swedish newspapers.

Abstrakt

Per Andersson, Lunds Tekniska Högskola
Text-to-scene conversion of car accident reports

Detta examensarbete är utfört inom CarSim (Dupuy et al. 2001) projektet. Systemet analyserar rapporter om bilolyckor och visualiserar dem i en tredimensionell grafisk simulering. Systemet består av två huvuddelar: en modul som analyserar texten och letar efter den information som är intressant och en modul för den grafiska simuleringen. En XML-mall i vilken information kan lagras på ett strukturerat sätt utgör en länk mellan de två modulerna.

Den första modulen fyller mallen med information extraherad från texten och den andra modulen läser mallen för att få den information den behöver för att kunna generera scenen grafiskt. Modulen för att analysera texten var innan detta arbete uppdelad i två separata versioner, en för att analysera franska texter och en för engelska texter.

Detta arbete är inriktat på att modifiera den del av systemet som extraherar information ur texten.

De två tidigare versionerna för franska och engelska integrerades till att kunna användas i samma grafiska användargränssnitt. Funktioner för att analysera svenska texter har också utvecklats. Analysatorn för svenska texter har skapats med och testats på ett korpus bestående av 209 rapporter som samlats från artiklar publiserade i svenska tidningar. Den tar fram information om olycksplatsens utseende och om de inblandade parterna. Attribut som systemet hittar är vägkonfigurationen samt statiska och dynamiska objekt. Ordklasstaggaren Granska har integrerats i systemet och innehar en central roll i textanalysen. Denna rapport beskriver de metoder jag utvecklat för att analysera texter skrivna på svenska.

Contents

Chapter 1 Introduction

1.1 Background

CarSim (Dupuy et al. 2001) is a text-to-scene converter. It analyzes texts from car accident reports and creates three-dimensional simulations of the accidents. The first version was made for reports written in French. Svensson and Åkerberg (2002) added a module to process English texts. Schulz (2002) improved the graphical user interface to merge both the information extraction part and the 3D generation into the same window.

CarSim consists of two main parts: the first part extracts information from the text and the second part generates the 3D simulation. An XML template is used to handle the communication between the two parts. The information extraction module fills the template with information about the accident, for example the road configuration and information about the vehicles. The 3D simulation module then animates the scene using information from the template.

The two former versions use different techniques and programming languages to extract information from the text. In the French version, the information extraction module is written in Prolog and in the English version it is mainly written in Java. The two different versions are thus separated in the information extraction part and use different graphical user interfaces. The 3D simulation module is written in Java and is shared by all the information extraction modules regardless of language. That means that, regardless of the language of the report, the extracted information can be visualized by the 3D simulation module.

1.2 Purpose and Goal

The goal of this work is to develop:

- A graphical user interface, which now integrates uniformly the language analyzers for the texts in French and English as well as the new analyzer for the texts in Swedish.
- An analyzer, which extracts information from Swedish texts about the scene of the accident and the actors of the collision.

The purpose of doing a text-to-scene conversion is to make the course of the accident easier and faster to understand. Texts can be quite complex, hard to read or understand and contain extra information that is irrelevant in the context. A graphic simulation gives a good idea of what actually happened.

1.3 Material Description

In order to develop a Swedish version of information extraction, we collected reports describing accidents written in Swedish. The reports come from two different sources. The first corpus contains 209 texts published in Swedish newspapers that have been gathered from their Web sites. The other corpus consists of reports extracted from STRADA[1], the accident database of the Swedish traffic authority (Vägverket).

This module has been developed and tested mainly on the *Nyheter* corpus. Half of the corpus was used during development and the other half was used for tests and evaluation of the system.

[1] Swedish Traffic Accident Data Acquisition.

1.4 Challenges

There are many problems to solve to have a working text-to-scene converter:

- Information extraction from real non-invented texts is a hard and challenging task. A result in which just a few simple texts can be analyzed and simulated correctly is in itself a major task.
- The corpus is limited in the way that the reports not always give the full story or contain all the information that can be looked for. This means that some facts about what happened might be missing and need to be guessed.
- The graphical module is limited and can only synthesize the objects that it finds in the template. In addition, the number of graphical objects is limited. All cars are visualized by the same type of graphical object and the only sign that can be shown is a stopsign.
- The current template is also limited by the fact that not all information about the accident can be added. For example it is not possible to add information about the weather conditions with the current template structure.

These limitations explain the challenges of text-to-scene conversion and the difficulty to obtain a high percentage of correctly analyzed texts. To expect a high throughput might be unrealistic during the timeframe of this work.

1.5 Planning

This work was divided into two major tasks. Chronologically, I started with the integration of the analyzers for the French and English versions into the same graphical user interface. The next step was to develop the analyzer for texts written in Swedish. First methods for finding the static objects and the road configuration were developed. When that part was working satisfactory the work with finding dynamic objects was started. Finally all the information gathered has to be read and arranged in order to fill the template.

1.6 Text-to-Scene Conversion Projects

Other projects that work with conversion between text and pictures include:

- WordsEye (Coyne and Sproat 2001), a system that converts English texts into three-dimensional static scenes that represent the text. WordsEye analyzes the text syntactically and semantically and produces a description of the arrangement of the objects mentioned in the text. The image is then generated from that description. To be able to generate many different kinds of objects and pictures WordsEye uses a large database of graphical objects.
- Nalig (Adorni et al., 1984; Di Manzo et al., 1986), a system that converts simple Italian texts into two-dimensional pictures.
- CogViSys (Arens et al. 2002), a project whose main goal is to convert visual information into a description in text.

Chapter 2 Analyzing Swedish Texts

This chapter describes the development of a module for analyzing Swedish reports. The goal with the analysis is to extract tabular data and fill a template. This data is used to synthesize the scene with information.

2.1 The Structure of the Nyheter Corpus

We created a first corpus of Swedish texts consisting of articles about car accidents published in Swedish newspapers. The articles were collected manually from the websites of the newspapers.

In addition to the accident narrative, additional information provided by the newspapers was kept. The file is built up by the attributes below. Not all attributes are always filled with information or even added, since sometimes the information may not have been available for that report.

The information to annotate the corpus consists of:

REPORT ID	The number identifying the report.
TITLE	The title of the article.
SUMMARY	A short summary of the text.
LOCATION	A specification of where the accident took place, typically a city.
FIGURE	Information about the picture name and the author.
NARRATIVE	The article containing the main text.
SOURCE	The date, author and the newspaper.

XML (Bray et al. 2000) is a markup language developed for documents using structured information. Unlike some other markup languages such as HTML, XML does not have a predefined tag set. The reason why XML is used in this work is that it can represent richly structured documents without limitations while still being relatively simple to implement. HTML has limitations in what it can represent. Other more complex markup languages may be able to represent the same data as XML, but are less common. XML is an accepted and well-documented language used for storing information. There are also many existing tools for manipulating XML files.

The former structure for storing the information in .ana and .ada files worked for the relatively small number of reports available in the French and English versions. However, for the Swedish version, with an initial access to over 200 reports, another structure was needed. The idea to gather all reports in one XML file has several advantages and no real disadvantages. It is easier to overview and find reports. It also makes the implementation easier and cleaner.

The following text is the first report in the `bilkorpus.xml` file.

```
<REPORT ID="R001">
  <TITLE>En person till sjukhus efter olycka vid Lund</TITLE>
  <SUMMARY>En bilolycka med tre bilar inträffade på motorvägen förbi
      Lund på fredagseftermiddagen.</SUMMARY>
  <NARRATIVE>Minst en person skadades och trafiken påverkas just nu
      rejält. Enligt polisens första rapporter har ingen skadats
      allvarligt.</NARRATIVE>
  <SOURCE CLASSIFIED="NO" COPYRIGHT="YES" NAME="Sydsvenskan">
    <DATE>29 november 2002</DATE>
  </SOURCE>
</REPORT>
```

In the example above, the LOCATION and the FIGURE were not available from the website and the tags are left empty and are not added.

The summary and narrative tags delimit the text that is analyzed by the program. That text also needs to be editable in the same way as in the former versions. This means that the program must allow the user to save, edit, delete and create new reports within the XML file.

2.2 Reading the Swedish Texts

This part covers how the program reads the Swedish texts.

2.2.1 JAXP and the Document Object Model

Starting with Java 1.4 JAXP, Java API for XML Processing was included in the core platform. It contains packages for representing and modifying XML documents in Java. There are four main packages that now are described.

- The Javax.xml.parsers package contains interfaces for instantiating SAX and DOM parsers.
- The Javax.xml.transform package uses the XSLT standard for transforming XML document content and representation.
- The org.xml.sax is an eventdriven API for parsing XML documents. A SAX parser does not store or build any representation of the document. Instead, it uses a `ContentHandler` object with methods that describe the content of the document.
- The org.w3c.dom package contains with interfaces for parsing XML documents. The DOM, Document Object Model, parser reads the document and stores the content in a tree of nodes. The generation of the tree requires the data being stored in memory, but once stored the tree can be modified effectively and by any program.

2.2.2 Implementation

The class, `XMLReportAnalyzer` reads the report files. It contains the methods needed to handle the functions for the new XML-structure. It uses two of the packages in the JAXP, namely `Javax.xml.transform` and `org.w3c.dom` including their subpackages. Comparing SAX and DOM gives that SAX is faster for scanning the documents while DOM is more suitable for doing modifications to a document once it is scanned. This application demands many modifications to be done to the document, which makes DOM the preferred choice for handling the parsing.

The XMLReportAnalyzer.GenerateReportTextFromXMLFile() method scans the file containing the reports and builds a tree of org.w3c.dom.Node objects using a DOM-parser. After the document has been converted into this structure, the information extraction part begins. The XMLReportAnalyzer.GenerateReportTextFromXMLFile() iterates through the entire document and for every report finds the nodes which contain the SUMMARY and NARRATIVE. The text inside these nodes are stored in a String array for later use.

The other two methods in XMLReportAnalyzer handle saving and deleting reports. The same procedure as when reading the document is applied. First, the parsed document is searched to find the correct report by looking for the node with the right number. Then, the content in the SUMMARY and NARRATIVE nodes in that report are changed or removed.

The Javax.xml.transform package is used to transform the DOM document representing an XML document back to XML again.

The process described is illustrated in Figure 1.

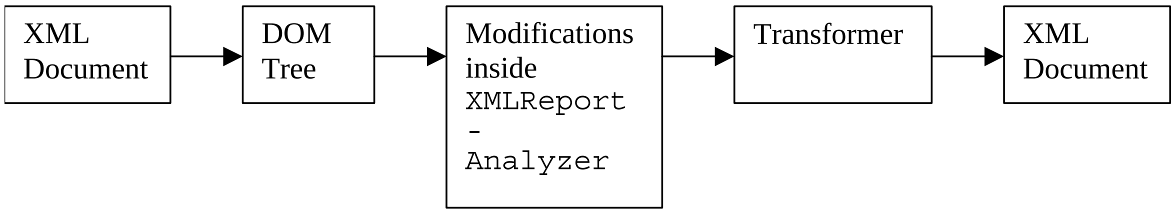


Figure 1. This picture shows the flow from the XML document to its DOM representation with modifications inside the program and then transformation back to the XML format.

2.3 The Template Structure

The information extracted from the texts is stored into a template. It ties the language processing and 3D-simulation modules together. It is a document in XML format that is filled with the information extracted from the text. The template consists of three main parts: static objects, dynamic objects and collisions. The objects added to the template must follow the DTD, Document Type Definitions. The DTD specifies which attributes the objects in the XML document must have and which values they can have.

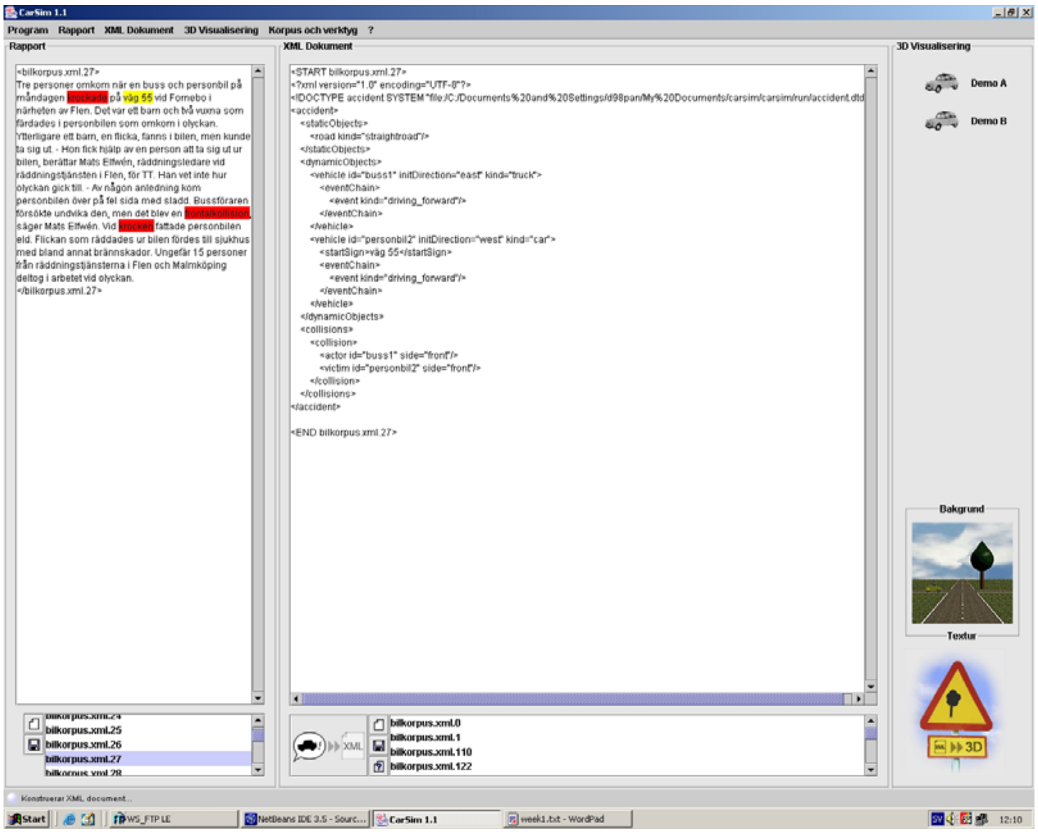
An example of a static object is road configuration, which can be straightroad, crossroads, turn_left or turn_right. Other static objects are signs, traffic lights and obstacles such as trees. The only sign that can be displayed in the graphical simulation is a stop sign. Traffic lights have an id and a colour that can be red, green, orange or inactive. All obstacles and signs other than stop signs are graphically represented as trees and have ids. The id numbers are needed to distinguish between different obstacles and to give them different coordinates. All the static objects, except the road configuration, have coordinates indicating where they are to be placed in the graphical simulation.

The dynamic objects are the moving objects, for instance cars and busses. A dynamic object has an id and an attribute that shows what kind of vehicle it is. This attribute can be car or truck. All large vehicles such as busses are set as trucks due to limitations in the graphical simulation module. A vehicle must also have an initdirection that shows in what direction the vehicle is initially travelling. The initial direction can be east, west, south or north. If the name of the road that the vehicle is travelling on can be extracted from the text it is also added as an attribute of the vehicle object. The final thing that is included in the dynamic object is a list of events the vehicle performs. These events can be driving_forward, turn_left, turn_right, stop, overtake, change_lane_left and change_lane_right.

Id numbers are used to separate different vehicles of the same kind and to map them into the next part of the template, the one concerning collisions.

A collision describes the crash between two dynamic objects or a dynamic object and a static object. It has two elements, an actor and a victim. The names actor and victim do not imply who is responsible for the accident, but is simply a way to name the participants. The dynamic object that hits something is set to actor. The actor and the victim have the same attributes: an id and a side. The side can be front, rear, left, right or unknown and is the part of the object that hits into or that is hit in the collision.

The structure of the template can be seen in Figure 2.



```
<bilkorpus.xml.27>
Tre personer omkom när en buss och personbil på
måndagen krockade på väg 55 vid Fornebo i
närheten av Flen. Det var ett barn och två vuxna som
färdades i personbilen som omkom i olyckan.
Ytterligare ett barn, en flicka, fanns i bilen, men kunde
ta sig ut. - Hon fick hjälp av en person att ta sig ut ur
bilen, berättar Mats Elfwén, räddningsledare vid
räddningstjänsten i Flen, för TT. Han vet inte hur
olyckan gick till. - Av någon anledning kom
personbilen över på fel sida med sladd. Bussföraren
försökte undvika den, men det blev en frontalkollision,
säger Mats Elfwén. Vid krocken fattade personbilen
eld. Flickan som räddades ur bilen fördes till sjukhus
med bland annat brännskador. Ungefär 15 personer
från räddningstjänsterna i Flen och Malmköping
deltog i arbetet vid olyckan.
</bilkorpus.xml.27>
```

```
<START bilkorpus.xml.27>
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE accident SYSTEM "file:/C:/Documents%20and%20Settings/d98pan/My%20Documents/carsim/carsim/run/accident.dtd
<accident>
  <staticObjects>
    <road kind="straightroad"/>
  </staticObjects>
  <dynamicObjects>
    <vehicle id="buss1" initDirection="east" kind="truck">
      <eventChain>
        <event kind="driving_forward"/>
      </eventChain>
    </vehicle>
    <vehicle id="personbil2" initDirection="west" kind="car">
      <startSign>väg 55</startSign>
      <eventChain>
        <event kind="driving_forward"/>
      </eventChain>
    </vehicle>
  </dynamicObjects>
  <collisions>
    <collision>
      <actor id="buss1" side="front"/>
      <victim id="personbil2" side="front"/>
    </collision>
  </collisions>
</accident>

<END bilkorpus.xml.27>
```

Figure 2. The picture shows the template structure in the middle of the picture.

2.4 Building the Semantic Lexicon

The semantic lexicon contains the significant words that can be used to fill the slots of the template. We classified them into five different categories, road words, obstacle words, road configuration words, condition words and motion words.

The words that fall into the road words category are road names. Obstacle words can be trees, poles, signs and other kind of static objects. Words that describe the weather for example *halt*, “slippery”, fall into the condition words category. The road configuration words describe the scene of the accident. It can be crossroads, straight road or a curve. Motion words are divided into three subcategories, namely crash verbs, motion verbs and direction verbs. Typical Swedish verbs that describe an accident, crash verbs, are *krocka*, “crash”, and *kollidera*, “collide”. The motion verbs are words that imply a sideways movement for example *sladda*, “skid”. Direction verbs are verbs that often come in conjunction of a word that gives a direction, *jag färdades norrut*, “I was travelling north.”

We used regular expressions to find the type of words that are fall into the defined categories. A regular expression is a pattern that can be used to find many different words that have a similar construction. The advantage is that with one pattern it is possible to find most of the words of a certain category instead of having to build a large database with all the words to look for. The words are found by searching the text for every regular expression. To find the words the Java native classes `Pattern` and `Matcher` are used.

In a regular expression, symbols with special functions or meanings are used together with literal characters. The following patterns mostly use the same special symbols and Table 1 shows a short description of the most common symbols.

Symbol	**Description**
*	Matches from zero to infinite number of occurrences of the previous characters.
+	Matches from one to infinite number of occurrences of the previous characters.
?	Matches one or zero occurrences of the previous characters.
\s	Matches white space character.
\S	Matches non-white space character.
[abc]	Matches a or b or c.
0-9	Matches any digit.
[^abc]	Matches any character except a, b or c.

Table 1. Symbols used in regular expressions.

For the road words category, there are five different patterns as seen in Table 2. To help understanding the patterns, the first one is now explained:

"E\\s?[0-9]+[^,\\!\\s\\.]*"

The first E means that the first letter of the word should be an E. The following \\s? indicates an optional white space character. After the E and optional white space, comes one or more figures from the [0-9]+ part of the pattern. Finally the end of the word or sentence is found with the [^,\\!\\s\\.]* part of the pattern.

Pattern	Description	Example
"E\\s?[0-9]+[^,\\!\\s\\.]*"	This pattern searches for words that start with *E* and that are then followed by one or more numbers.	*E4* and *E22*.
“([A-Z]|[ÅÄÖ])\\S+väg[^,\\!\\s\\.]*(\\s|\\.|,|\\!)[^(0-9)]”	This pattern searches for words that ends with *väg*, road, and that starts with a capital letter.	*Östanväg*.
"\\S+gatan[^,\\!\\s\\.]*"	This pattern searches for all words that contain *gatan*, the street.	*Södergatan* and *Tingsgatan*.
"\\S*[Vv]äg\\s?[0-9]+[^,\\!\\s\\.]*"	This pattern searches for words with structure *väg* followed by a number.	*länsväg 12*, *riksväg 11*.
"\\s([a-z]|[åäö])\\S+vägen[^,\\!\\s\\.]*"	This pattern searches for words that ends with *vägen*.	*Magistratsvägen*.

Table 2. Patterns and descriptions for regular expressions and road words.

For the obstacle words category there are two different patterns as seen in Table 3.

Pattern	Description	Example
"\\s[Tt]räd[,\\s\\.\\!]"	This pattern searches for words that contain *träd*, tree.	*Träd*.
"[^,\\!\\s\\.]*([Ss]tolpe|([Tt]rafikljus)|(([Ss]topp)+(skylt|likt)))[^,\\!\\s\\.]*"	This pattern searches for words that contain *stolpe*, pole, and *skylt*, sign.	*stoppskylt*

Table 3. Patterns and descriptions for regular expressions and obstacle words.

For the weather condition words category there is one pattern as seen in Table 4.

Pattern	Description	Example
"[^,\\!\\s\\.]*(vattenplaning|halka|halt|torrt|regn)[^,\\!\\s\\.]*"	This pattern finds words that describe the weather condition for the road.	*halka* and *regn*

Table 4. Patterns and descriptions for regular expressions and condition words.

For the road configuration words category there is one pattern as seen in Table 5.

Pattern	Description	Example
"[^,\\!\\s\\.]*((korset|korsning|korsa|(vänsterkurva(n?))|((höger)?kurva(n?)))[^,\\!\\s\\.]*)"	This pattern finds words that describe the road configuration. The configuration can be crossroad, straightroad or a bend.	*korsning, högerkurva*

Table 5. Patterns and descriptions for regular expressions and road configuration words.

For the motion words category, there is one pattern as seen in Table 6.

Pattern	Description	Example
"(([^,\\!\\s\\.]*(krock|krasch|ramma|toucha|studsa|påkör|kolli|(kör[^,\\!\\s\\.]* in i)|(for emot)|törnade)[^,\\!\\s\\.]*)|([^,\\!\\s\\.]*(volta|(kör[^,\\!\\s\\.]* av vägen)|kör[^,\\!\\s\\.]* på|(slog runt)|(slog emot)|(slå runt)|(slå emot))[^,\\!\\s\\.]*)|([^,\\!\\s\\.]*(slunga[^,\\!\\s\\.]*|sladda[^,\\!\\s\\.]*|braka[^,\\!\\s\\.]*|kana[^,\\!\\s\\.]*|rände|ränna|törna[^,\\!\\s\\.]*|smäll[^,\\!\\s\\.]*|for|fara|slog|slå|dunsa[^,\\!\\s\\.]*)[^,\\!\\.]*(in i)[^,\\!\\s\\.]*))"	This pattern searches for words that indicate some kind of movement.	*sladda*

Table 6. Patterns and descriptions for regular expressions and motion words.

2.5 Colorization of the text

The graphical user interface colorizes the words interactively as the texts are selected. The idea behind displaying certain words in predefined colors is to simplify the reading of the reports. In order to quickly understand the course of the accident without reading the whole report, it is possible to just look for the colored keywords. This, in combination with the graphical simulation, simplifies and speeds up the understanding of the accident described in the report. It also helps the reader to identify the interesting parts of the text. Some reports contain a lot of text that does not provide any information about the configuration of the scene or the actors. Typically, these texts involve injuries and casualties as a result of the collisions.

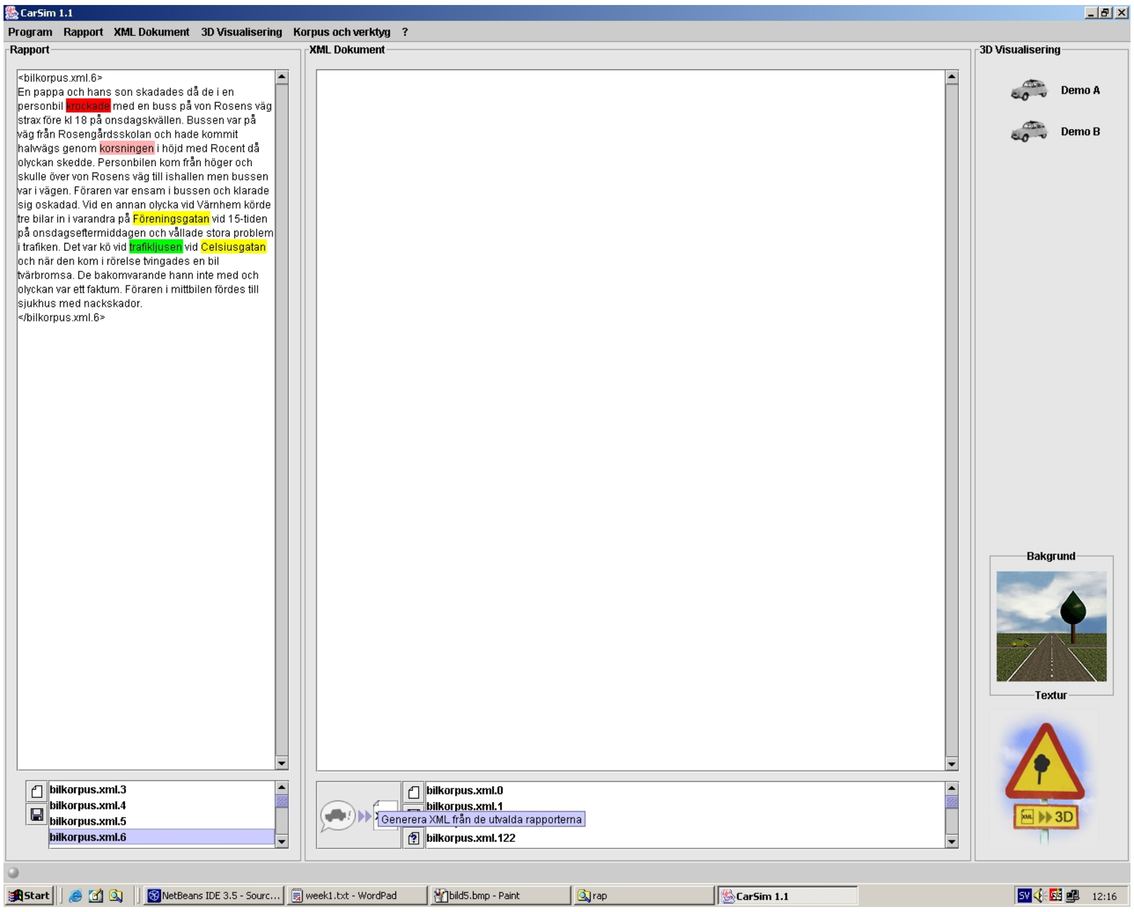
Currently the colorization of the words indicating the weather condition is the only part of the program that takes those variables into account.

Table 7 shows the defined colors for the different word type categories.

Category	Color
Road words	Yellow
Obstacle words	Green
Road configuration words	Pink
Condition words	Magenta
Motion words	Red

Table 7. Definitions of colors for different types of words.

An example of a text that has been colored can be seen in Figure 3.



<bilkorpus.xml.6>
En pappa och hans son skadades då de i en personbil krockade med en buss på von Rosens väg strax före kl 18 på onsdagskvällen. Bussen var på väg från Rosengårdsskolan och hade kommit halvwägs genom korsningen i höjd med Rocent då olyckan skedde. Personbilen kom från höger och skulle över von Rosens väg till ishallen men bussen var i vägen. Föraren var ensam i bussen och klarade sig oskadad. Vid en annan olycka vid Värnhem körde tre bilar in i varandra på Föreningsgatan vid 15-tiden på onsdagseftermiddagen och vållade stora problem i trafiken. Det var kö vid trafikljusen vid Celsiusgatan och när den kom i rörelse tvingades en bil tvärbromsa. De bakomvarande hann inte med och olyckan var ett faktum. Föraren i mittbilen fördes till sjukhus med nackskador.
</bilkorpus.xml.6>

Figure 3. The picture shows the coloring of the text.

2.6 Extraction of the Static Objects and the Road Configuration

This section describes how the static objects and the road configurations are extracted from the reports. This part has been developed in parallel with the colorization of keywords in the text. The regular expressions used to find words describing static objects are used both to colorize and to fill the template with the information about these objects. During development of the program, the colorization of the words has made it easier to verify that the text analysis part actually finds the right words.

Static objects are objects that have a passive role in accidents. They can be hit but cannot hit anything since they cannot move. Static objects that are searched for in the text are trees, stopsigns, trafficlights and obstacles such as poles. Limitations in the graphical simulation module make all obstacles appear as trees in the graphic simulation. This does however not affect the filling of the XML template.

The road configuration is basic information about the scene of the accident. To find it, a list of words was manually extracted from the corpus by reading it and finding the words that describe crossings and turns. There are four different possibilities: it can be a crossing, a straight road or a road that turns either left or right. If a word indicating a turning road is found, the sentence is searched for words that indicate whether it is a left- or rightturn. For example in the sentence *Bussen kraschade in I ett träd på höger sida om vägen*, “The bus crashed into a tree on the right side of the road”, the word *höger*, “right” indicates that the bus left the road to the right and that the tree should be placed on that side. If no such word is found it is defaulted to a rightturn since it has to be set in order for the program to function. The same technique is used for trying to find what side of the road that for example a tree is situated at.

If a static object is found it is added to the static objects part of the template in XML format. If several trees are found in the same text, they are given different coordinates. Road configuration is set to straightroad as default if no other type is found.

For the colorization the regular expressions are used to find words describing roads and roadnames. For this application that information is not useful. A roadname such as *Strandvägen* gives no information about the type of road.

2.7 Extraction of the Words that Indicate Motion and Collisions

The technique that is used to find the static objects with regular expressions is also used to find the words indicating motions and collisions. To find these words and develop regular expressions, the development corpus has manually been read and searched for such words. After the manual extraction, a list of words has been found. The actual list of patterns to search for becomes shorter after including some regular expressions.

Being able to find the words that describe motions and accidents, gives a method to find the sentences that are of importance to extracting information about the dynamical objects.

Verbs that indicate a crash, crash verbs, are important for finding the participants in an accident, ie. the subject and object of the sentence. The structure that needs to be found is: <subject>, <crash verb> and <object>.

Here is an example:

Bilen krockade med en buss
“The car collided with a bus.”

In the example, the car is the subject, the bus is the object, and collided is the crash verb. The generalized pattern is <subject> collided with <object>.

In a sentence, it is possible to find the subject and object if the verb is known. This is accomplished by applying different rules, but requires a grammatical analysis of the text. The verbs that indicate motion can give information about the direction of the vehicle or in what direction it left the road.

Bilen färdades västerut på vägen.
“The car was traveling west on the road.”

Bilen sladdade till höger och lämnade vägen.
"The car skidded to the right and left the road."

The lists of words that imply collisions or motions are stored in text files that are read by the program. Hence it is easy to modify and enlarge the list of words to search for in a future development of the database of reports.

2.8 The Grammatical Analysis of the Text

When a word describing a motion or a collision is found, the next step is to find the subject and the object in the same sentence as the found word. To do this we carry out a grammatical analysis of the text that consists of three steps. The text is split into sentences. Every sentence is tagged by a part-of-speech tagger. Finally grammatical rules are applied to find noun groups that are annotated as subject and object

2.8.1 The Sentenizer Class

The `Sentenizer` class is responsible for tokenizing and dividing the text into sentences. This is needed in order to be able to use the Granska part-of-speech tagger. Granska analyzes and tags one sentence at a time and cannot handle the full text. A full report is split into sentences, which are then analyzed one by one by the tagger. To split the text into sentences the JDK, Java Development Kit, class `BreakIterator` is used. The `BreakIterator` class has methods for finding locations of boundaries within texts. The sentence boundary analysis gives the correct location for the boundaries between the sentences in most cases. It handles quotation marks, parentheses and abbreviations.

Only one construct that does not give a correct location of the sentence boundary has been found in the corpus *Nyheter*. This means that it is necessary to search for that kind of structure. When a sentence starts with a dash after another sentence as in the following pattern, "sentence. - sentence" it is wrongly seen as one sentence. Here follows two examples taken from the corpus. They both have the structure that the `BreakIterator` class cannot handle.

Innan han hann reagera körde Niklas i 90 kilometer i timmen rakt in i en parkerad bil. - Jag hade inte en chans att bromsa när jag insåg att bilen stod parkerad, och att svänga över till höger körfält var en omöjlighet eftersom trafiken var så tät, berättar han.

"Before he could react Niklas crashed into a parked car at 90 kmph. –I had no chance to break when I realized that the car stood parked and to turn over to the right lane was impossible because of the heavy traffic, he says."

Den omkomne 20-åringen var ensam i sin bil och körde i riktning mot Borgholm. I en mötande bil åkte tre kvinnor och fyra män. - Det finns uppgifter om att den ensamme förarens bil har kommit över på fel sida av vägen, men det är någonting vi fortfarande utreder, säger närpolischef Sven-Erik Karlsson vid Kalmarpolisen till TT.

"The 20-yearold that died was alone in his car and was driving towards Borgholm. Three women and four men were traveling in a car in the opposite direction. –There are statements that the driver of the car traveling towards Borgholm came over to the wrong side of the road, but that is still under investigation, says the policeofficer Sven-Erik Karlsson of the Kalmar police to TT.”

The two sentences and any other sentence after them that starts with a dash are seen as one sentence while they in fact are not. The solution to this problem is to check every sentence received from `BreakIterator` for the “. - [A-Z]” structure with a regular expression and if it is found to split it into two or more sentences. The pattern that is used to find the structure in a sentence that `BreakIterator` fails to divide properly looks like this: "."+" "+"-"+" ".

2.8.2 The Granska Part-of-Speech Tagger

Granska (Domeij et al. 1999) is a program for computer-aided examination of language. It is being developed at KTH in Stockholm. It looks like a regular text editor, but also has functions for evaluating the written texts and giving suggestions to correct errors. It will find both spelling errors and grammatical errors.

Granska uses a set of rules to find grammatical errors for example in Swedish an adjective in singular may not be followed by a noun in plural.

A part-of-speech tagger is included in the Granska program (Carlberger and Kann, 1999). It analyzes one sentence at a time and marks every word with its part-of-speech and morphological tags.

The Granska POS tagger has been integrated into the CarSim system and tested both in Windows and Unix runtime environment. The program is written in C++ and therefore introduces some minor portability issues to the CarSim system. To be able to tag the full text from the report, the text first has to be split into sentences by the `Sentenizer` class. After that the first sentence is temporarily stored on disk in a text file. Granska is then told to analyze the text file and the CarSim program reads the output. The tagged sentence is then ready for further analysis and the next sentence is saved on disk for tagging. Figure 4 shows the flow from a full report that is sent to the sentenizer and split into sentences and then analyzed by the Granska tagger.

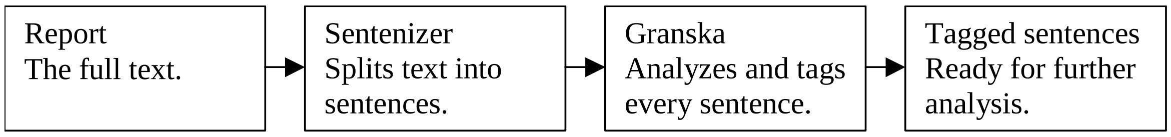

Report
The full text.
Sentenizer
Splits text into sentences.
Granska
Analyzes and tags every sentence.
Tagged sentences
Ready for further analysis.

Figure 4. The picture shows the flow through the system for a sentence getting tagged.

The following text shows the tagged version of the sentence:

En allvarlig bilolycka inträffade vid åttatiden på tisdagskvällen på Linnégatan i Limhamn.

“A serious car accident occurred around 8 pm on Tuesday evening at Linnégatan in Limhamn.”

En *<dt.utr.sin.ind>* ***allvarlig*** *<jj.pos.utr.sin.ind.nom>* ***bilolycka*** *<nn.utr.sin.ind.nom>* ***inträffade*** *<vb.prt.akt>* ***vid*** *<pp>* ***åttatiden*** *<nn.utr.sin.def.nom>* ***på*** *<pp>* ***tisdagskvällen*** *<nn.utr.sin.def.nom>* ***på*** *<pp>* ***Linnégatan*** *<pm.nom>* ***i*** *<pp>* ***Limhamn*** *<pm.nom>* ***.*** *<mad>*

The tags use the SUC, Stockholm-Umeå Corpus, tag set. It takes too much space and makes the text hard to overview to print out the full names of the tags for every word, so abbreviations are used. Table 8 shows the tags and their meaning for the given example sentence.

Word	Tags	Meaning
En	dt.utr.sin.ind	determiner.utrum.singular.indefinite
allvarlig	jj.pos.utr.sin.ind.nom	adjective.positive.utrum.singular.indefinite.nominative
bilolycka	nn.utr.sin.ind.nom	noun.utrum.singular.indefinite.nominative
inträffade	vb.prt.akt	verb.preterite.active
vid	pp	preposition
åttatiden	nn.utr.sin.def.nom	noun.utrum.singular.definite.nominative
på	pp	preposition
tisdagskvällen	nn.utr.sin.def.nom	noun.utrum.singular.definite.nominative
på	pp	preposition
Linnégatan	pm.nom	pronoun.nominative
i	pp	preposition
Limhamn	pm.nom	pronoun.nominative
.	mad	Marks the end of the sentence.

Table 8. A tagged sentence explained.

2.9 Finding Subject and Object By Phrase analysis

A sentence can be divided into different kinds of phrases, for example noun phrases, verb phrases, prepositional phrases, and adverb phrases. The phrases are well defined and can be extracted from a text that has been tagged by a part-of-speech tagger.

If the crash verb in a sentence is known, the possible subject and object can be searched by finding the noun phrases of the sentence. A noun phrase is defined to follow one of three patterns (Megyesi and Rydin, 1999):

1. It can be a single pronoun.
   [PRON]
2. It can also be an optional determiner followed by one or more optional numerals followed by an optional adjective phrase followed by one or more optional numerals followed by at least one noun.
   (DET)([NUM]+)(AP)([NUM]+)[NOUN]+
3. Finally, it can be an optional determiner followed by a possessive pronoun followed by an optional adjective phrase followed by at least one noun.
   (DET)POSSPRON(AP)[NOUN]+

Once all the noun phrases are found, they create a list of possible subjects and objects. However, only noun phrases that are not part of a prepositional phrase can be subjects or objects. A prepositional phrase has a structure of a preposition followed by a noun phrase or a preposition followed by a conjunction followed by a preposition followed by a noun phrase.

The conclusion here is that it is enough to search for the first possible word preceding a noun phrase and see if it is a preposition to determine if it can be a subject or object to the crash verb.

Here is an example of a sentence tagged with the Granska tagger:

En bil färdades västerut på Strandvägen kom över till vänster och krockade med ett träd.

"A car was traveling west on Strandvägen, came over to the left and collided with a tree."

And this is how is looks like after it has been tagged with the Granska part-of-speech tagger:

En *<dt.utr.sin.ind>* ***bil*** *<nn.utr.sin.ind.nom>* ***färdades*** *<vb.prt.sfo>* ***västerut*** *<ab>på <pp>* ***Strandvägen*** *<pm.nom>* ***kom*** *<vb.prt.akt>* ***över*** *<pl>* ***till*** *<pp>* ***vänster*** *<nn.utr.sin.ind.nom>* ***och*** *<kn>* ***krockade*** *<vb.prt.akt>* ***med*** *<pp>* ***ett*** *<dt.neu.sin.ind>* ***träd*** *<nn.neu.sin.ind.nom>* **.** *<mad>*

In this case, the crash verb is *krockade* and the noun phrases are:

Vänster, "left"
En bil, "a car"
ett träd, "a tree"

Here, the only noun phrase not preceded by a preposition is *En bil*, "a car", and is hence by the above rules the only possible subject or object to the crash verb *krockade*, "collided".

2.10 Patterns

There are certain constructs and patterns of sentences that are common and that can be found for different words with similar meaning. For more information about the technique see FrameNet (Johnson et al. 2002).

We have manually designed a set of patterns using the development part of the *Nyheter* corpus. Table 9 lists the structures we found relevant where *fill1 is the set {*våldsamt, häftigt, kraftigt, därvid, in*} and *fill2 is {*annat, annan, mötande, andra, tredje, två, framför*}.

1.	Participant 1	crash verb	fill1*	med	en ett den	fill2*	participant 2
2.	Participant 1	crash verb	fill1*	mot	en ett den	fill2*	participant 2
3.	Participant 1	crash verb	fill1*	i	en ett	fill2*	participant 2

Table 9. Extraction patterns.

Example sentences from the three different main cases:

1. *Bilen krockade våldsamt med en mötande buss*.
   The car crashed into the meeting bus.
2. *Lastbilen kolliderade därvid med den andra stolpen*.
   The truck then collided with the other pole.
3. *Bussen körde in i staketet*.
   The bus drove into the fence.

The fill1 and fill2 lists and the "en, ett, den" are optional and do not have to be a part of the sentence for the pattern to be fulfilled. From these patterns, it is obvious that the words before and after the crash verb likely are the subject and object of the sentence. But in some cases these patterns break the rules stated above about noun phrases not being part of a prepositional phrase. Therefore the program looks for these patterns and if they are found adds the objects to the list of possible objects and subjects even though they are not supposed to be that according to the rules above.

2.11 How the Subjects and Objects are Extracted

To find the subject and the object in a sentence, even when the verb is known, is a hard task. The rules above with phrase analysis and common patterns give two lists of candidates: one with subjects and one with objects. We applied general topological rules governing the place of the subject and object relative to the verb in Swedish. There are some rules that can be applied to where the subject and object can be placed relative to the verb and to each other. Here we use a combination of grammatical and semantic heuristics.

The sentences should have one of the following structures:

- Subject verb object.
- Subject object verb.
- Verb subject object.

The two lists are iterated and every word is checked against a list of words describing vehicles and static crashable objects like trees and poles. Words not found in the lists are disregarded. This makes the two lists shorter, since the ones not matching words describing vehicles and static objects are removed. If there are no words filling all the criteria the lists may be empty.

The algorithm works as follows: If both the lists of candidates before and after the verb are nonempty, the first words, ie. the ones closest to the verb, is set as subject and object in case of a crash verb. If it is a motion verb the subject is be set as the first candidate before and then the sentence is searched for a word describing a direction. If one list is empty and the other list contains two or more candidates it may add the two closest words in the nonempty list as subject and object.

2.12 Finding the Dynamic Objects

Once the relevant parts have been extracted from the sentences found by searching for crash verbs and motion verbs, they have to be saved in some kind of data structure. To handle this the class `VerbPhrase2` is used. It holds information about the verb, the subject, the object and the full text of the sentence. In the case of a motion verb, the object contains information about the direction, while it describes a vehicle or a static object in the case of a crash verb. The `VerbPhrase2` class also has methods for finding what are the parts of the vehicles that were involved in the collision and for finding directions. This can be found by searching for words in the sentence that says what part is involved.

For example, if it is given in the text that a car traveling west crashed into the left side of another car it is assumed that the second car was driving north if no other information about its direction is given.

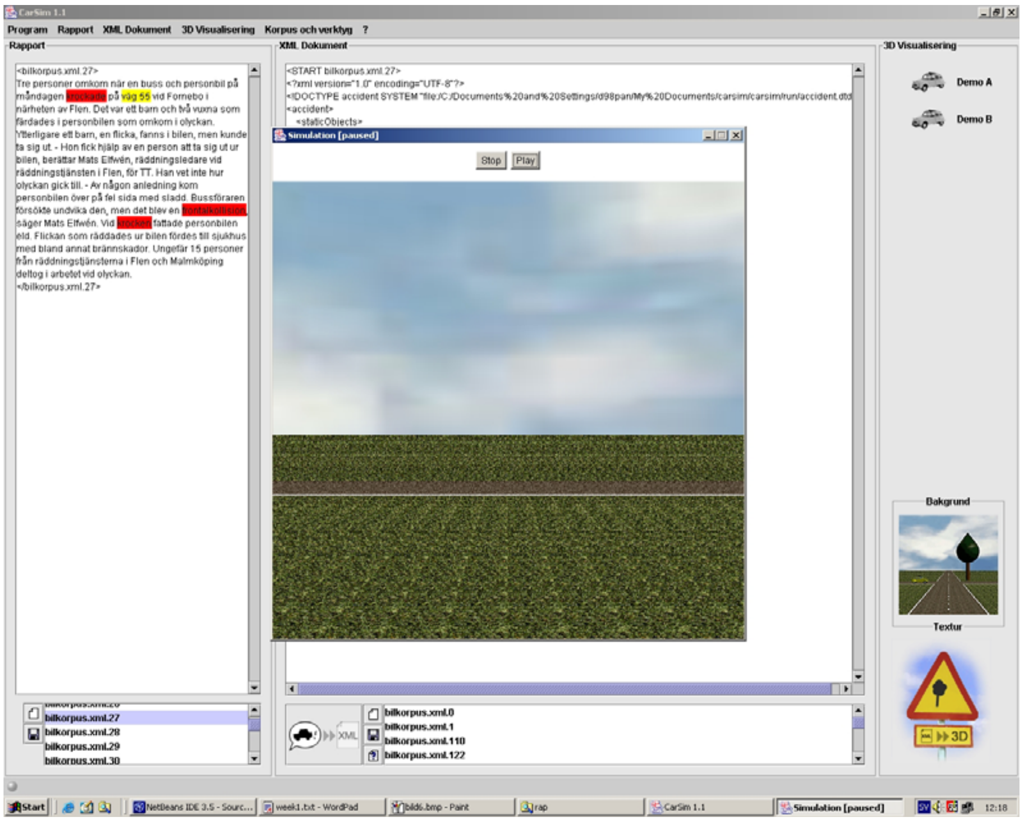
The simulator needs an initial direction for all the dynamic objects in order to know where to place them and in what direction they should initially move.

2.13 Accidents

When all the text is analyzed and the `VerbPhrase2` frames are filled with information from sentences with crash verbs and motion verbs, the dynamic objects and collision structure need to be created. The class `DynamicObject` holds information about the name of the road the vehicle starts on, what kind of vehicle it is (car or truck), its initial direction, the real word found in the text (for example *Volvo*), a list of events that the dynamic object carries out and a boolean parameter that tells if the dynamic object is involved in a collision or not. When an object of type `DynamicObject` is created it has none of this information but the class has methods for finding it. For example, if the subject is called *Volvo* it is matched against a list of words that describes cars and when the match is found it is set to car. If no match is found it is instead set as truck. Not all objects from the verb phrases are dynamic objects since they can also be directions or static objects.

The program uses a simple algorithm to identify the dynamic objects and resolve the co-references. It goes through the verb phrases and identifies dynamic objects from the subjects and objects. The dynamic objects are added to a list after a check that they have not already been created. Dynamic objects and static objects are linked together by their id numbers if they are involved in a collision. The dynamic objects also get events added to their event lists according to the information in the verb phrases that contain motion verbs.

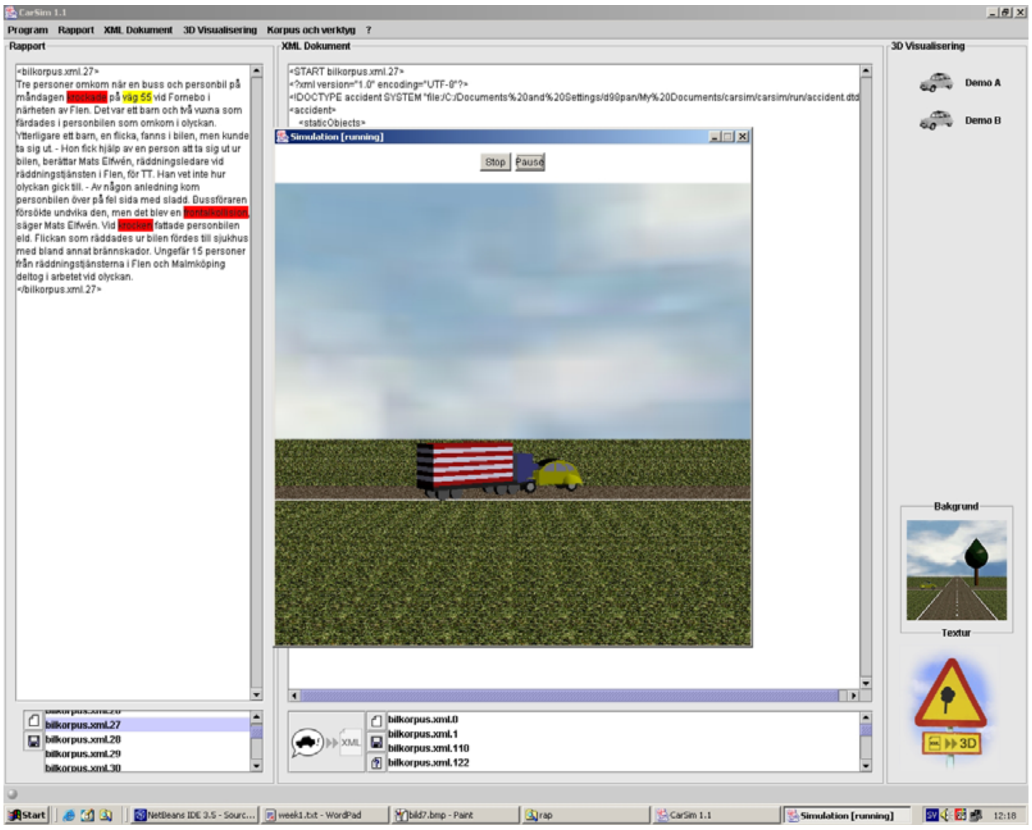
After all information is extracted and filled into the XML template, the graphical simulation module reads the template and creates a visualization of the scene. When the simulation button is pressed a new window, shown in Figure 5, is opened. There are two buttons in that window, one for starting the simulation and one for stopping it. If the play button is pressed the simulation starts and the crash is shown. This is shown in Figure 6.



<bilkorpus.xml.27>
Tre personer omkom när en buss och personbil på måndagen krockade på väg 55 vid Fornebo i närheten av Flen. Det var ett barn och två vuxna som färdades i personbilen som omkom i olyckan. Ytterligare ett barn, en flicka, fanns i bilen, men kunde ta sig ut. - Hon fick hjälp av en person att ta sig ut ur bilen, berättar Mats Elfwén, räddningsledare vid räddningstjänsten i Flen, för TT. Han vet inte hur olyckan gick till. - Av någon anledning kom personbilen över på fel sida med sladd. Bussföraren försökte undvika den, men det blev en frontalkollision, säger Mats Elfwén. Vid krocken fattade personbilen eld. Flickan som räddades ur bilen fördes till sjukhus med bland annat brännskador. Ungefär 15 personer från räddningstjänsterna i Flen och Malmköping deltog i arbetet vid olyckan.
</bilkorpus.xml.27>

```
<START bilkorpus.xml.27>
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE accident SYSTEM "file:/C:/Documents%20and%20Settings/d98pan/My%20Documents/carsim/carsim/run/accident.dtd
<accident>
  <staticObjects>
```

Figure 5. The window shows the graphical simulation of the accident.



<bilkorpus.xml.27>
Tre personer omkom när en buss och personbil på måndagen krockade på väg 55 vid Fornebo i närheten av Flen. Det var ett barn och två vuxna som färdades i personbilen som omkom i olyckan. Ytterligare ett barn, en flicka, fanns i bilen, men kunde ta sig ut. - Hon fick hjälp av en person att ta sig ut ur bilen, berättar Mats Elfwén, räddningsledare vid räddningstjänsten i Flen, för TT. Han vet inte hur olyckan gick till. - Av någon anledning kom personbilen över på fel sida med sladd. Bussföraren försökte undvika den, men det blev en frontalkollision, säger Mats Elfwén. Vid krocken fattade personbilen eld. Flickan som räddades ur bilen fördes till sjukhus med bland annat brännskador. Ungefär 15 personer från räddningstjänsterna i Flen och Malmköping deltog i arbetet vid olyckan.
</bilkorpus.xml.27>

<START bilkorpus.xml.27>
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE accident SYSTEM "file:/C:/Documents%20and%20Settings/d98pan/My%20Documents/carsim/carsim/run/accident.dtd
<accident>
  <staticObjects>

Figure 6. The result after a simulation is run, a crash.

Chapter 3 A Complete Example

To show how the system works, an example is now given that shows the entire process from start to end.

The text below describes the accident that is analyzed in this example.

> *En person skadades lindrigt vid en bilolycka på Slottsgatan. Bilen körde in i ett träd på vänster sida av vägen av okänd anledning. En person satt i bilen och skadades lindrigt. Någon brottsmisstanke finns inte, men polis och ambulans larmades till olycksplatsen strax innan halv två i natt. Man spekulerar i att olyckan orsakats av halka i kombination med en skarp vänsterkurva.*
>
> "One person sustained minor injuries after a car accident on Slottsgatan. The car crashed into a tree on the left side of the road by unknown reason. One person was in the car and was slightly injured. There is no suspicion of crime, but the police and an ambulance were called to the scene of the accident just a moment before half past one last night. The accident was likely due to slippery conditions in combination with a sharp left turn."

The analysis starts with extracting the static objects and coloring the keywords in the text. In this case, the word *vänsterkurva,* "left turn", in the last sentence describes the road configuration and it is set to turn_left. In the same sentence, the word *halka* implies slippery conditions and is therefore colored with magenta. The word *Slottsgatan* is found by the third regular expression that looks for road names and is displayed in yellow colour. There is only one sentence with a crash verb and a motion verb, namely the second one. The words *körde in,* "drove into", from the second sentence are marked in red to show that they describe a crash and motion. A tree is added as a static object from the word *träd,* "tree", in the second sentence.

The sentence

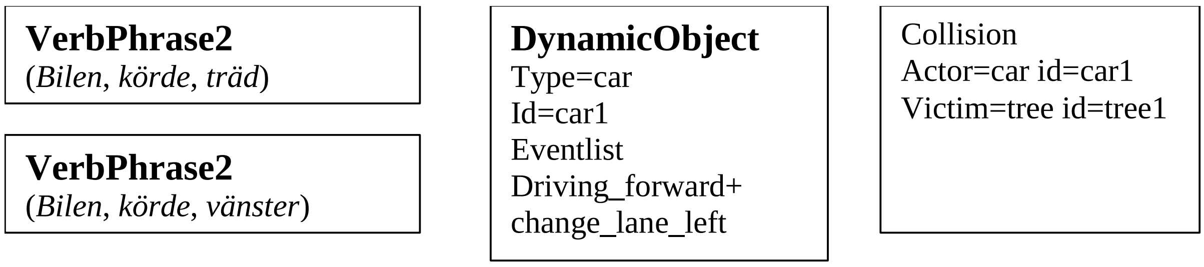
Bilen körde in i ett träd på vänster sida av vägen av okänd anledning.
"The car crashed into a tree on the left side of the road by unknown reason."

is taken from the text by the *Sentenizer*, saved on disk in a text file, and then be tagged by Granska. The tagged sentence is analyzed and the word *Bilen* is added to the subject candidate list after the phrase analysis and the word *träd* is added to the object candidate list when going through the common pattern technique. Since both lists are non-empty and both *Bilen* and *träd* match the lists for possible subjects and objects they result in a verb phrase with *Bilen*, *körde*, *träd* where *körde* is a crash verb.

Since the verb *körde* also matches a motion verb another verb phrase is created with *Bilen*, *körde, vänster*.

When analyzing the verb phrases and trying to create the dynamic objects and resolve the collision, the verb phrases are used. From the first verb phrase, a dynamic object with real name *Bilen* is added as a car since it matches one of the words in the list describing cars. The id number of the dynamic object and the id number of the tree are linked together as being involved in a collision. When analyzing the second verb phrase the subject has the same realword, *Bilen*, as an already existing dynamic object and no new object is therefore created. However the event change_lane_left is added to the existing dynamic object.

The dynamic object finds the road name *Slottsgatan* as the road it is initially driving on and it is also displayed as an attribute.



VerbPhrase2
(*Bilen*, *körde*, *träd*)

VerbPhrase2
(*Bilen*, *körde*, *vänster*)

DynamicObject
Type=car
Id=car1
Eventlist
Driving_forward+
change_lane_left

Collision
Actor=car id=car1
Victim=tree id=tree1

Figure 7. The picture shows the information that is extracted in the example and how it is represented.

Chapter 4 A New and Extensible Graphical User Interface

When this project started, there were two modules for text analysis: one for French texts and one for English texts. They had separate graphical user interfaces, which meant that they could not be run in the same program. In order to make the system more user-friendly and to simplify future expansions, it was necessary to integrate all modules for text analysis into the same interface. The program developed for the English version was used as a basis to create a more generic version.

The previous user interface made it possible to choose a report from a list showing all the reports, edit a report, save a report, create a new report and to generate the XML template from the report. Another list holds all the already generated XML templates and there are buttons for saving, creating and for validating the selected XML template. There are also four drop down menus with various options and functions, such as selecting language, exiting the program, saving and editing files and for starting the three dimensional simulation.

The English reports were read from a hard coded folder name containing text files named *.ana. The generated template files were saved in the same folder and named *.ade. The French texts were also stored using this structure. The corpus consisting of reports about car accidents collected from Swedish newspapers use a different structure. The Swedish reports are all stored in one XML file. To be able to read the reports and generate the template it is necessary to know what type of structure the reports are stored in.

To make the program more general, a new drop down menu consisting of two parts was added. In the first part, it is possible to select the corpus from which the reports should be listed and in the second what tool to use for analyzing the chosen reports. When the program starts, all subfolders to the folder `carsimdata`, which is located in the root directory, are examined. If they are not empty, they are added to a list in the menu with the same names as the found folders. This makes it easy to add a folder containing a new corpus to the program without editing the code. The tools to read and analyze the selected type of reports are static but there can be several different tools for each type of report and language.

Figure 8 and Figure 9 show the functionality added to the graphical user interface. In Figure 8, the menu shows the four corpuses the user can choose: MAIF, NTSB, Nyheter, and Strada. Figure 9 shows that there are two different tools for English, one tool for Swedish, and one for French.

The XML template generation is started by pressing a button in the graphical user interface. If no tool is selected a message appears and asks the user to select one. When a user selects a report type from the menu, the folder containing the selected reports is examined.

Different techniques for reading the report are used depending on if they are stored in XML or .ana format. A folder may contain several XML files, but also some .ana files. This does not affect the program however and all files regardless of format are added to the visible list of reports. The elements in the list are named by the file they come from. If a report comes from an XML file it is given a number that indicates where in the XML file it is located. For example an XML file named `corpus.XML` may contain ten reports. The third report in the file will then get the name `corpus.XML.3` since the numbering starts at one and increases by one for every read report.

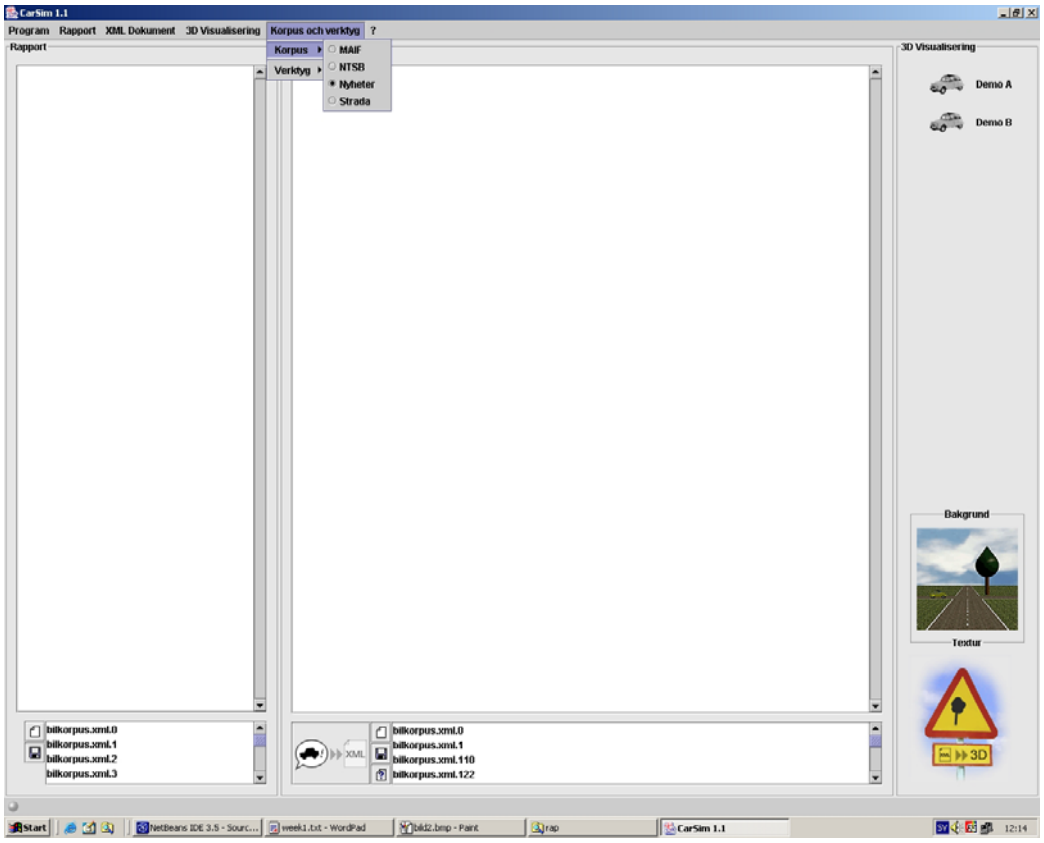


Figure 8. The picture shows the dropdown menu that is used for choosing what corpus to work on.

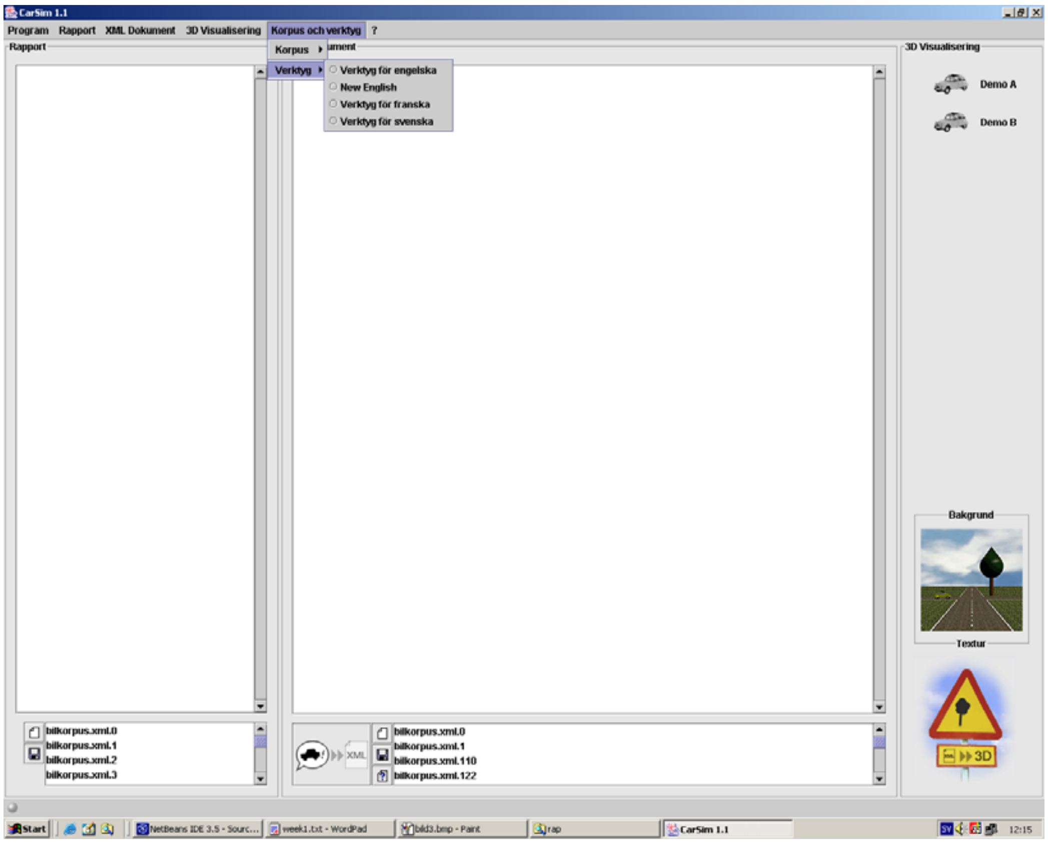


Figure 9. The Figure shows the dropdown menu for selecting what tools to use.

Chapter 5 Discussion

5.1 Result and Evaluation

During the development of the program only half of the 209 reports in the corpus *Nyheter* were used. All the algorithms and methods were run and tested on these reports. Even though the methods are written in a general way to work for all kinds of text, it is natural to expect a higher amount of reports that can be correctly analyzed among those reports compared to another set of reports. Among the reports used during development four are in full correctly analyzed. Some reports are close to being correctly analyzed but have some error in the representation.

To evaluate how well the system works the program was run for the other half of the corpus *Nyheter*. The result can be compared to the numbers achieved with the development half of the corpus and give a measure of how general and robust the system is. One report from the half not used during development is correctly analyzed. A number of other reports are close to being correctly analyzed. It is likely that additional development on this part of the corpus will yield a number of correctly analyzed reports.

Table 10 shows the results from running the first 30 reports in the corpus *Nyheter*. The symbols and headings used are first explained:

The text	The report that is run.
Actors	Shows if the number of actors is correctly extracted.
Crash	Shows if the crash is correctly extracted.
Events	Shows if the events of the dynamic objects are correct.
Initial direction	Shows if the initial direction is correctly extracted.
Coordinates	Shows if the coordinates of the static objects are correct.
Static objects	Shows if the static objects are correctly extracted.
Number of elements found	Shows how many objects that are found in the text.
Correct IE	Shows if the information extracted is correct.
Correct without the events	Shows if the information extracted is correct if the events are not considered.
Realistic	Shows if the template is correctly filled and if the graphical simulation is correct.

A one indicates that the category is correctly extracted and a zero that it is not. If the information cannot be found in the text, then it is indicated by a question mark.

The text	Actors	Crash	Events	Initial directions	Coordinates	Static objects	Number of elements found	Correct IE	Correct without the events	Realistic
Bils.xml.0	0	0	0	?	?	1	0	0	0	0
Bils.xml.1	0	0	0	0	?	1	2	0	0	0
Bils.xml.2	0	0	0	0	?	1	1	0	0	0
Bils.xml.3	0	1	1	1	1	0	3	0	0	0
Bils.xml.4	0	0	0	?	1	1	1	0	0	0
Bils.xml.5	0	0	0	?	?	1	0	0	0	0
Bils.xml.6	0	1	1	1	1	1	4	0	0	0
Bils.xml.7	0	0	0	0	?	?	0	0	0	0
Bils.xml.8	0	0	0	?	?	1	1	0	0	0
Bils.xml.9	0	0	0	0	1	1	1	0	0	0
Bils.xml.10	0	0	0	0	0	1	0	0	0	0
Bils.xml.11	0	0	0	1	0	1	1	0	0	0
Bils.xml.12	0	?	?	?	?	1	0	0	0	0
Bils.xml.13	0	0	0	0	0	1	0	0	0	0
Bils.xml.14	0	0	0	?	?	1	2	0	0	0
Bils.xml.15	0	0	0	?	?	1	0	0	0	0
Bils.xml.16	0	0	0	?	?	1	1	0	0	0
Bils.xml.17	0	0	0	0	0	0	0	0	0	0
Bils.xml.18	0	0	0	0	1	1	2	0	0	0
Bils.xml.19	0	0	0	1	?	1	1	0	0	0
Bils.xml.20	0	0	0	0	?	0	5	0	0	0
Bils.xml.21	0	0	0	0	?	1	3	0	0	0
Bils.xml.22	0	0	0	0	?	1	2	0	0	0
Bils.xml.23	0	0	0	0	?	0	4	0	0	0
Bils.xml.24	0	0	0	0	0	0	4	0	0	0
Bils.xml.25	1	0	0	0	?	1	2	0	0	0
Bils.xml.26	0	0	0	0	?	1	0	0	0	0
Bils.xml.27	1	1	1	1	?	1	2	1	1	1
Bils.xml.28	0	0	0	0	?	1	1	0	0	0
Bils.xml.29	0	0	0	0	?	1	6	0	0	0
Bils.xml.30	0	0	0	0	0	0	3	0	0	0

Table 10. The table shows information about how well the system analyzes reports.

bils.xml.0	There is no crash verb in the text so no accident or actors are found.
bils.xml.1	The system finds two dynamic objects but there is only one mentioned in the text. It finds the road configuration and the road name.
bils.xml.2	The text does not contain a crash verb and the accident is not found.
bils.xml.3	The crash is found correctly, but the system fails to find one dynamic object not involved in the accident. The road configuration is also found, but two trees are found where it should only be one.
bils.xml.4	The text does not contain information about the vehicle that crashes into a tree.
bils.xml.5	The system fails to find the actors and the collision.
bils.xml.6	The road configuration is correctly found. Also the static object traffic light is extracted with the correct colour. The system finds the dynamic objects and the collision, but adds one dynamic object that is not supposed to be there.
bils.xml.7	No keywords are found in the text and the default XML template is generated.
bils.xml.8	The text is hard and contains several collisions.
bils.xml.9	The vehicle that crashes is not mentioned. The system cannot handle sentences like "The man crashed into the tree".
bils.xml.10	The system fails.
bils.xml.11	This is a long text and the system fails.
bils.xml.12	No keywords are found in the text and the default XML template is generated.
bils.xml.13	The only information found is about the roadconfiguration and a roadname.
bils.xml.14	The system finds the wrong actors even though the number is correct. A collision is found, but not correct. Road configuration and roadname are correct.
bils.xml.15	No keywords are found in the text and the default XML template is generated.
bils.xml.16	The system finds one of three dynamic objects and fail to realize the collision.
bils.xml.17	This is a long and hard text and the system fails.
bils.xml.18	This is a long and hard text that contains several different accidents and the system fails.
bils.xml.19	The vechicle that crashes is not mentioned. The system cannot handle sentences like "The man crashed into the tree".
bils.xml.20	This is a long and hard text that contains several different accidents and the system fails.
bils.xml.21	This is a long and hard text that contains several different accidents and the system fails.
bils.xml.22	The text describes a complex accident involving three cars and the system fails. The road configuration and roadnames are correctly extracted.
bils.xml.23	This is a long and hard text that contains several different accidents and the system fails.
bils.xml.24	This is a long and hard text that contains several different accidents and the system fails.
bils.xml.25	The actors are found, but the initial directions and the events are wrong.
bils.xml.26	The text does not contain any crash verbs and no dynamic objects are found.
bils.xml.27	The information is correctly extracted from this report.
bils.xml.28	This is a complex text and the system fails.
bils.xml.29	This is a long and hard text that contains several different accidents and the system fails.
bils.xml.30	This is a long and hard text that contains several different accidents and the system fails.

5.2 Future Development

There are two main parts of the system that need improvements: the template structure and the information extraction.

The current template does not contain all the different scenarios and objects that are needed to give a more correct and reasonable representation of the accident. With more objects and information added to the template, the simulation will be more accurate in the cases where that information can be extracted from the text. All parts of the template would not be filled for every text because sometimes the information is not given. Giving more information about the accident might also attract people working with traffic safety by analyzing accidents. If more information is given and shown about the accident it is easier to draw conclusions about the cause of the accident.

New attributes describing information about the road could be added to the template. They could show the weather condition, temperature, time and light conditions. Other issues that might be of interest are if the road is on a slope and what material is on the surface of the road. Also many active and passive objects could be added such as pedestrians, animals, elks, birds, and other obstacles.

A new more exhaustive template will demand a reworking of the graphical simulation module to be able to represent the new objects and to read the new template structure. Currently there is for example only one car model. One possibility is to add a specific car model for every brand.

The other part of this work, the information extraction module, has room for many improvements. The template is the link between the two main modules and the proposed changes will also affect the information extraction module. New attributes added to the template means that methods for extracting that information from the text must be implemented.

The current extraction of information from the Swedish texts also has room for improvements. There are cases the system cannot handle, for example the sentence:

Två bilar körde in i varandra.
“Two cars crashed into each other.”

The system will in this case fail to find that it is two cars and only find one actor, *bilar*, “cars”, and the crash verb *körde*, “drove”, but no victim.

Chapter 6 Conclusion

This report starts by generally describing language processing and the CarSim system. Then it describes in detail how the language analyzers were integrated into the same graphical user interface and how the tools for analyzing Swedish texts were developed and how they work. The report covers the use of regular expressions and the part of speech tagger Granska.

The methods for finding the information and the structures that store information are described. An example describes step by step how a given text is transformed to a graphical simulation.

In the end of the report, the results are presented and suggestions for future improvements are discussed.

References

Giovanni Adorni, Mauro Di Manzo, Fausto Giunchiglia, Natural language driven image generation. In *Proceedings of COLING 84,* Stanford, California, 495-500, 1984

Ola Åkerberg, Hans Svensson, Development and Integration of Linguistic Components for an Automatic Text-to-Scene Conversion System, Lunds universitet, LTH, *MSc. Thesis*, Lund, August, 2002.

Michael Arens, Artur Ottlik, Hans-Hellmut Nagel, Natural language texts for a cognitive vision system. In van Harmelen, F., ed.: *ECAI2002, Proceedings of the 15th European Conference on Artificial Intelligence*, Lyon, 455 459, 2002.

Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, *Extensible Markup Language (XML) 1.0,* Second Edition, W3C Recommendation, 6 October 2000, `http://www.w3.org/TR/REC-xml`

Johan Carlberger, Viggo Kann, Implementing an efficient part-of-speech tagger, *Software Practice and Experience*, 29, 815-832, 1999.

Bob Coyne, Richard Sproat, Wordseye: An automatic text-to-scene conversion system. In *Proceedings of the Siggraph Conference*, Los Angeles, 2001.

Mauro Di Manzo, Giovanni Adorni, Fausto Giunchiglia, Reasoning about scene descriptions, *IEEE Proceedings, Special Issue on Natural Language*, 74, 1013-1025, 1986

Rickard Domeij, Ola Knutsson, Johan Carlberger, Viggo Kann, Granska – ett effektivt hybridsystem för kontroll av svensk grammatik, *NoDaLiDa'99*, 49-56, December 1999.

Sylvain Dupuy, Arjan Egges, Vincent Legendre, Pierre Nugues, Generating a 3D Simulation of a Car Accident From a Written Description in Natural Language: The CarSim System, *in Proceedings of The Workshop, on Temporal and Spatial Information Processing,* pp. 1-8, ACL 2001 Conference, Toulouse, 7 July 2001.

Christopher R. Johnson, Charles J. Fillmore, Miriam R. L. Petruck, Collin F. Baker, Michael Ellsworth, Josef Ruppenhofer, Esther J. Wood, *FrameNet: Theory and Practice, Version 1.0,* Printed September 13, 2002, `http://www.icsi.berkeley.edu/~framenet/book/book.html`

Beáta Megyesi and Sara Rydin, Towards a Finite State Parser for Swedish, *Proceedings NoDaLiDa'99*, Trondheim, Norway, December 1999.

Bastian Schultz, Development of an Interface and Visualization Components for a Text-to-Scene Converter, Lunds universitet, LTH, *MSc. Thesis*, Lund, 2002.

Abbreviations

DOM	Document Object Model
DTD	Document Type Definitions
JAXP	Java API for XML Processing
JDK	Java Development Kit
KTH	Kungliga Tekniska Högskolan
SAX	Simple API for XML
SUC	Stockholm-Umeå Corpus
XML	eXtensible Markup Language
XSLT	Extensible Stylesheet Language Transformations