

LDOM

Table of contents

1 Linguistic Document Object Model — LDOM.....	2
2 Schematic working.....	2
3 LDOM GUI.....	3

1 Linguistic Document Object Model — LDOM

The LDOM is conceptually modelled after the DOM (Document Object Model) used in web page modelling. The basic idea is to have a programmatic space for representing linguistic structures for the whole document. By modelling the whole document in linguistic terms, a whole new set of operations will become possible, such as:

- whole-text proofing (unknown words can be compared, and names and new terms recognised)
- long distance anaphora resolution and checking
- text and content classification
- whole-text editing operations (change from present to past tense, from one style or spelling to another)
- editing operations on linguistic entities, automatically maintaining linguistic integrity & correctness (change a verb into another with a different subcategorisation frame, and at the same time (suggest to) change the argument(s) accordingly)
- interactive syntax tree manipulation (e.g. fronting of object NP's by drag and drop; switching coordinated NP's by drag and drop, etc)
- interactive correction of parsing errors by graphically manipulating the syntax tree (corrected parse results will give better writing and correction aids, etc)
- term consistency checks
- disambiguated and content-based hyphenation (e.g. SV "bil-drulle" vs "bild-rulle")
- synchronised, simultaneous and interactive machine translation
- more ...

2 Schematic working

In the following, we assume a MVC-based editor/programming environment, such as TextEdit/Cocoa in MacOS X. This should give automatic access to the textual content of the document, on which one can build the wanted linguistic structures.

When a document is opened in a LDOM-aware editor/process, the text objects of the document is - in addition to being rendered on-screen - also synchronised with another object structure: the LDOM. Initially the LDOM only contains very basic textual units, such as paragraphs, headers, lists and table text, and text fragments in other languages. These units are automatically inferred from the document structure by the LDOM code, and will form the basis for further linguistic processing. There could also be some statistical processing of the text at this point.

It is necessary to identify these units, since they require different linguistic processing strategies. Different languages need different linguistic components, and headers, lists and table texts are usually quite different from normal running body texts of complete sentences.

This first processing is done by the LDOM library code, it is language independent, and always available irrespective of language. When it is done, the real linguistic processing

starts. For each identified language, first the language identity is checked against available analysers (the first-pass, probability-based language guesser could have guessed wrong - if so, a more thorough testing against available analysers should correct this). Secondly, each sentence is parsed morphologically and syntactically as far as available tools make it possible.

The actual processing is language specific as well as technology agnostic. That is, for Swedish text the linguistic analysis could be done by a statistical parser, whereas for Finnish text it could be done by a combination of an FST and a CG grammar, giving full dependencies. For certain operations a limited set of compatibility structures are needed, such as the representation of a syntactic tree. This should be limited to only how the branches are associated with each other, not the number names of the branches themselves. This is to facilitate as much theoretical independence as possible, and thus encourage experimentation using different frameworks and theoretical foundations. The theory/implementation specific representation can be retained behind this compatibility layer.

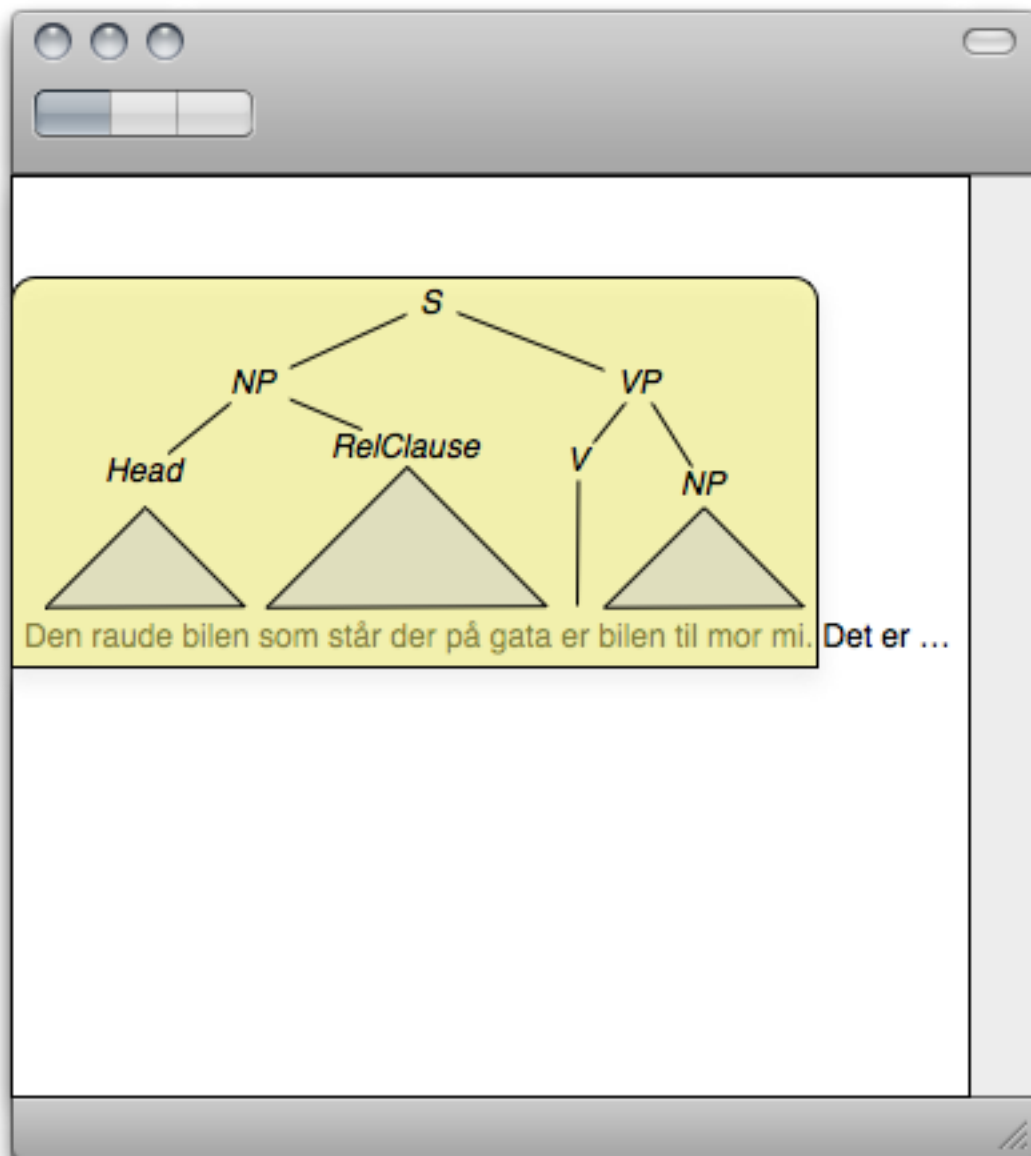
3 LDOM GUI

An important part of the idea about a LDOM is that it should be possible to access it directly through a suitable user interface. It is through this interface that most of the operations listed above are possible. For example, to reorder a number of coordinated NP's (e.g. "this, that and that", each determiner here representing a complex NP with article, modifying adjectives etc), one would - instead of copying and pasting and adjusting the location of the CC in the string at the same time - just bring up the LDOM GUI, and move the NP node around in the syntactic tree by clicking and dragging. The textual representation of the linguistic objects affected by the operation would automatically be adjusted accordingly.

Since syntactic trees can be both huge and complex, the GUI should automatically display only the structures needed and hide the rest. What is needed should be determined based on the location of the pointer, and it could also be triggered manually by some mouse operation (e.g. by double-clicking a certain collapsed tree).

The GUI could be an overlay over the actual text bough up using the contextual menu or with a keyboard shortcut. Within this overlay one would manipulate the syntactic tree, proofread the parsing result, etc. Examples of overlays in existing software are the Dictionary popup in MacOS X, the Find highlighting in Safari and FireFox, etc.

Here's a mockup of what it could resemble:



Examples of operations one could imagine on a node in this tree structure are:

- by right clicking on an NP node, one could get a menu with possible transformations, e.g. from definite to indefinite, from singular to plural, etc
- by clicking the RelClause node, the whole relative clause would be selected, and a single press on the DEL button would remove the whole clause
- by double-clicking (or hovering over) the triangles representing the structure of the NP's and RelClause, the structure would unfold, and make it available for manipulation - by clicking outside the RelClause, the tree structure would be collapsed again, to keep the visible parse tree manageable; several levels of such nesting is possible, given relevant input

In addition to an overlay, one could also imagine some functionality be provided using a dialog interface, or through an info window.

It is important to note that all interactive manipulation of the linguistic structures needs to be supported by the language technology component providing the analysis. If it does not provide transformation operations, it will not be possible for the user to transform e.g. a noun phrase from singular to plural. Selection operations on the other hand (selecting an NP, and cutting and pasting it) does not need any support from the underlying linguistic component, since such operations can be executed using the regular OS functionality.