The complete guide to Pickering

A semantic interpreter for spoken language

Jens Edlund <edlund@speech.kth.se>
Gabriel Skantze <gabriel@speech.kth.se>

The complete guide to *Pickering*: A semantic interpreter for spoken language

by Jens Edlund and Gabriel Skantze

Published 2004

DRAFT

Table of Contents

Preface

This is the manual of the *Pickering* semantic interpreter. The following few pages contain information on the organisation of the book, its target audience, and the writing and layout conventions used in the book.

1. Why read this book?

Pickering is part of the *Higgins* research project at *CTT*, *KTH*, Sweden. *Higgins* is a speech technology project aimed at investigating robustness and error handling techniques in spoken dialogue systems. Several conference articles have been published within the project, both on *Pickering* and on other topics. The authors assume that potential users of *Pickering* may be other speech researchers who's read some *Higgins* publications and are curious to try the interpreter out for themselves.

2. Target audience

Pickering is a semantic interpreter intended for extractingmeanings from spoken or written language.This book, then, is directed to potential users of *Pickering*, such as speech technology researchers, computational linguists and suchlike. The book can be read in part by any potential *Pickering* user, but certain parts are clearly intended for developers rather than casual users. The first part is an overview and should be reasonably accessible to all.

Note, however, that although this book aims at comprehensively documenting the technical details of *Pickering*, it makes no claim of teaching speech technology - readers will have to have formed a basic idea of what they want to use *Pickering* for before reading this book.

3. Organisation of the book

This book is divided into several parts, each of which can be read on its own. Part I, “An overview of the Pickering semantic interpreter” lets you get started by introducing the installation and user interfaces, but skips most of the gory details. It contains general descriptions of the functionality and features of *Pickering*, and includes some simplistic examples of *Pickering* grammars. Finally, it gives an overview of *Pickering*'s visualisation features. ??? is intended for users with some experience of *Pickering*. It contains formal descriptions of the accepted input formats, grammars, and semantics. ??? gives a detailed description on how to use *Pickering*'s visualisation features. The text is example based and contains examples of techniques that should be helpful to *Pickering* users.

4. Conventions

Layout conventions. The final versions of this book are produces using standard methods such as XSL transformations, and yhe authors make no attempt to control the formatting results in detail. Nevertheless, some conventions are upheld. They are descibed below, with examples in-lined in the descriptions.

- specially
- formatted
- itemised
- list

Acronyms and words that are explained in the glossary look like the following: XML. Each chapter starts with an text describing its contents, as well as a a specially formatted itemised list highlighting key areas.

Code examples. Inline and block code examples are formatted as `monotype text`.

Example 1. A code example

```
Block code examples often ❶
        have a list of things worth
        noticing underneath them - this
                list is linked to places in
        the code examples
with callouts ❷.
```

❶ This block is an example of what code examples look like.
❷ This sentence describes what *callouts* are.

Encoding. The book is encoded in 

[http://www.oasis-open.org/docbook/xml/], and is available for perusing as *XHTML*, *PDF* and *PostScript* documents.

PDF version: A table of contents and a table of examples, both linked, are included. References to specifications and programming languages, etc., are given in-line as links to the glossary. Acronyms are also linked to the glossary. Further references are found in the glossary, also encoded as links. *XML* elements are linked to their respective *Higgins*/*Pickering* specification, when applicable. A page based index is suplied at end of the book.

PostScript version: References to specifications and programming languages, etc., as well as acronym explanations, are given in the glossary. The book comes with a table of contents, a table of examples and a page based index (at the end of the book).

XHTML version: A table of contents and a table of examples, both linked, are included. References to specifications and programming languages, etc., are given in-line as links to the

glossary. Acronyms are also linked to the glossary. Further references are found in the glossary, and are encoded as links. XML elements are linked to their respective *Higgins*/*Pickering* specification, when applicable. A linked, section based index is suplied at the bottom of the page.

5. Getting the book

Versions. This is version 0.1.2 (of August 2004) of the book. The latest version is available at http://www.speech.kth.se/higgins/modules/pickering-manual.shtml.

Status. The introduction and first part of the book is in beta (1st draft) status. The second and third parts are largely unwritten, and basically consists of a composition. The book as well as the interpreter are works in progress, so expect minor faults and discrepancies. Please notify the authors should you stumble on anything strange.

6. Acknowledgements

The code of the *Pickering* semantic interpreter is developed and maintained by Gabriel Skantze. The functional requirements and specifications for the interpreter were used in the interpreter are the result of the *Higgins* project in collaboration between Gabriel Skantze & Jens Edlund. Grammars, visualisation code and utilities are coded by Gabriel Skantze, Jens Edlund, and Anna Hjalmarsson. Thanks to Anna Hjalmarsson, Rolf Carlson, and the CHIL project, amongst others, for testing and comments.

We thank Norman Walsh et.al. for their good work with DocBook XSL (by the way, the structure of this book is loosely based on that of [Walsh & Muellner 2001]).

This research was carried out at the *Centre for Speech Technology*, a competence centre at KTH, supported by VINNOVA (The Swedish Agency for Innovation Systems), KTH and participating Swedish companies and organisations.

Part I. An overview of the Pickering semantic interpreter

Introduction

Pickering is a robust interpreter[1], designed for semantic interpretation of natural language. The input may be written language, but the emphasis *Pickering*development is on spoken language in the form of automatic speech recognition (ASR) results. The interpretation results are intended for use in spoken dialogue systems (SDS). Thus, emphasis is placed on the possibility to develop grammars and semantics rapidly and flexibly, as well as providing robust interpretation of distorted input. The robustness methods are placed in the interpreter engine, so that the grammar writer is freed from having to encode robustness in the grammar. All interpreter resources (grammar, lexicon, morphology, style sheets for presentations) are encoded in XML as are the parse trees and the resulting semantic analyses. XML was chosen to make it easier to incorporate the interpreter in a distributed architecture where different components may be implemented in different languages and/or on different platforms, and to enable implementers to write style sheets in CSS or XSLT/XSL to graphically present resources and results using standard components, such as web browsers and XSLT/XSL engines. Transformations to other formats may also be achieved using standard methods, for example XSLT. The XML techniques used in *Pickering* follows the specifications issued by the W3C as closely as possible.

The interpreter uses a method similar to island parsing. It is implemented as a modified chart parser in Mozart/Oz, but no knowledge about Oz programming is required to use it.

[1]Semantic interpreter, or *parser*, if you will. Parser, however, has a great many meanings to different people in the speech technology, computaional linguistics and dialogue systems communities, so we'll stick to interpreter.

Table of Contents

Chapter 1. Features

Highlights

- *Pickering* feature overview
- Flexible semantic representations
- Flexible grammar writing
- Robust semantic interpretation

In this chapter: This chapter gives a brief overview of the features *Pickering* offers. First, some grammar features are presented. More comprehensive information about *Pickering* grammars can be found in Chapter 5, *Overview of Pickering grammars* (a non-formal description) and in ???, which contains a full grammar reference. Furthermore, some of the possibilities for building complex semantics in *Pickering* are presented, followed by what type of input the interpreter is built to handle, and finally, some of the features of the GUI are described along with some of the visualisation possibilities.

1.1. Grammar writing flexibility

- Mixed rule content. Grammar rules may contain a mix of:
  - references to other rules
  - references to lexicon entries
  - explicit words

  Mixing explicit words into rules provides speed for rapid development/testing, but should probably be avoided from a maintainability point of view. The interpreter allows grammar writers to be guided by their needs and common sense, rather than restricting the possibilities.
- Levels of abstraction.
  - A lexicon may be used , but is not mandatory.
  - A morphology may be ised, but is not mandatory.

  Again, the grammar writer is encouraged to make design choices suitable to the task at hand. A large, complex grammar that is to be used for a long time will most likely benefit

from abstraction, whereas hard-coding everything into the rules will get you started faster.

- Modularity. The grammar may be split into a number of files, or modules. These are combined by using include statements, either in each file that needs an inclusion, or in a master file listing all included files.

1.2. Feature grammars

- The interpreter supports agreement of linguistic (such as definiteness and number) or other features over word sequences. Agreeing features are automatically copied into the containing rule, and may be copied into the resulting semantics.
- Non-agreeing input, which is likely to occur in ASR results, will still be accepted, but is ranked lower. This feature can be suppressed.

1.3. Flexible semantics

- The semantics are encoded to any depth and complexity chosen by the grammar writer. Applications where feature-value pair frames, or even simple keywords, are sufficient may use these, whereas more complex applications may use deep structured semantic trees or nested feature-values.
- The interpreter unifies partial semantics according to a semantic template provided by the grammar writer, instead of using lambda expressions. The template states how parts fit together in the specific application.

1.4. Confidence scores [*not implemented*]

Confidence scores [not implemented]. The interpreter will compute a confidence score for each result. ASR word confidence, agreement, number of insertions, and other factors are weighed in.

1.5. Robustness

- Partial results are parsed and delivered. The phrases do not have to cover the whole input string.
- The interpreter automatically allows insertions inside rules and lexical entries. The number of insertions permitted can be coded into the grammar (defaults to two). When used with ASR, where disfluences and out-of-vocabulary (OOV) words may trigger random words to

appear anywhere in the string, this provides robustness.

- If a full parse is not obtained, the best combination of partial results is found. Rules and lexicon entries can be prohibited from being allowed to function as stand-alone results, either on a rule-by-rule, entry-by-entry basis, or using more general techniques.
- Top rule(s) need not be explicitly declared. The interpreter can be made to select the combination of rules and entries that covers the highest number of words in the input while using the fewest number of rules.
- All solutions resulting in non-equivalent semantics can be delivered in an ordered n-best list.

1.6. On-line features

- The interpreter can do incremental parsing. Words can be added one by one. Each addition will result in a new interpretation, without the entire input having to be reprocessed each time. The input string can also be changed partially without forcing a full re-interpretation. The best combination of solutions can be retrieved at any time.
- The interpreter has a web browser interface (currently IE only). Solutions can be viewed graphically in a browser window as they are being found. Style sheets creating a graphical view are included, and personalised style sheets can be coded and linked to the grammar. Switches between style sheets can be made using the minimal graphical user interface (GUI). Default style sheets can also be specified in the grammar, and more then one style sheet may be applied sequentially.
- Batch processing. The interpreter can process a large number of string in one sweep and write the results to a compiled file.
- Rapid testing. A text file with utterances may be read into the GUI. The sentences are presented in a text window, and clicking one of them will automatically interpret it. Changes can be made in the grammar, which can then be reloaded and the effects easily tested.
- Web server mode. *Pickering* can be started as a web server, in which case the string that is to be interpreted is sent to *Pickering* from a web browser, either with the GET or the POST methods. The resulting XML is returned to the web browser.
- Comand line mode. *Pickering* placed in a pipe chain on the command line. It then reads input from STDIN and writes output to STDOUT.

1.7. Extras

The interpreter comes with a number of pre-written stylesheets, both in CSS and XSLT. The stylesheets can be linked in the grammar, or chosen through the interpreter's GUI. The results of these stylesheets may be viewed in standards complient web browsers.

Chapter 2. How to find *Pickering*

Jens Edlund <edlund@speech.kth.se>
Gabriel Skantze <gabriel@speech.kth.se>

Highlights

- *Pickering* distribution
- XML Schema distribution;
- Style sheet distribution

In this chapter: Instructions on how to find the *Pickering* distribution, or parts thereof, as well as a brief version history is included in this chapter.

2.1. Download locations

Pickering is distributed at the web site of the department of speech, music and hearing [http://www.speech.kth.se] at TMH, KTH, on the *Higgins* project web pages: http://www.speech.kth.se/higgins/modules/pickering.shtml. The distribution comes in a zip archive, and this book is available both on-line and for download. The latest version of the Windows distribution and this book can be found at:

- http://www.speech.kth.se/higgins/modules/pickering-win.zip
- http://www.speech.kth.se/higgins/modules/pickering-manual.zip [http://www.speech.kth.se/higgins/modules/pickering-win.zip]

2.2. Contents of the *Pickering* distribution

The *Pickering* distribution of version 0.9.0 contains the following files:

- Pickering.exe - *Pickering* executable.
- PickeringConfiguration.xml - default configuration file
- ExampleGrammar.xml - an example grammar, funnily enough
- PickeringManual.pdf - PDF version of this book
- TestPickeringWebServer/index.html and iTestPickeringWebServer/input.html - example

page for web server mode (input.html is inlined in a frame in index.html)

2.3. XML Schemas and XSL stylesheets

The *Pickering* XML Schemas and XSL stylesheets are still under development, but examples can be found at:

- http://www.speech.kth.se/higgins/2003/ namespace [http://www.speech.kth.se/higgins/2003/]
- Standard transformations for presentation of *Pickering* results [http://www.speech.kth.se/higgins/2003/pickering/result/transformations/parseresult_ie.xslt]

Documented stylesheets and updated schemas are underway.

Chapter 3. Installing *Pickering*

Jens Edlund <edlund@speech.kth.se>
Gabriel Skantze <gabriel@speech.kth.se>

Highlights

- Prerequisits
- Installing *Pickering*
- Partial installations
- On-line materials

In this chapter: This chapter details what needs to be done before *Pickering* can be installed, how to install *Pickering* or parts of it, and how to use the public web resources for *Pickering* provided by TMH.

Note that version 0.9.0 is only available as a Windows binary, so this installation instruction is valid for Windows only. Our intention is to make future versions available for other platforms as well.

3.1. Prerequisits

In order to use Pickering, the following applications must be installed:

- Mozart 1.3 or later [http://www.mozart-oz.org/]
- Perl 5.6 or later [http://www.activestate.com/]

If you want to use *Pickering* in GUI mode, you also need to have the Perl package Win32::OLE installed. See below for instructions on how to get and install Win32::OLE.

To find out if the Win32::OLE package is installed under ActiveState Perl, open a Command Prompt and run: `> ppm > install win32-ole > q`

If Win32::OLE is installed, you will get the message "Note: Package 'Win32-OLE' is already installed", otherwise it will be installed for you.

3.2. Installing *Pickering*

To install *Pickering*, simply unzip the distribution into a directory of your choice. If you want

to be able to run *Pickering* from any directory, you have to add the directory containing Pickering.exe to your PATH environment variable.

3.3. Using public on-line resources

This section will contain information on how to use, download and manipulate the XSL and XML Schema files provided within the *Higgins* project. It has yet to be written.

Chapter 4. Running *Pickering*

Jens Edlund <edlund@speech.kth.se>
Gabriel Skantze <gabriel@speech.kth.se>

Highlights

- *Pickering* GUI
- Command line usage
- CTT Broker interface
- Web server mode

In this chapter: *Pickering* can be run in four different modes: GUI, web server, broker and command line mode. The interfaces for these modes are are presented in this chapter: the GUI and the command line tools. The GUI is presented with screen dumps and examples, and the command line tools with formal synopses.

4.1. Configuring *Pickering*

To start Pickering, simply run Pickering.exe.

If no argument is provided, *Pickering* will look for "PickeringConfiguration.xml" for its configuration. The name of a file can also be provided as an argument to specify another configuration file.

Pickering can be run in four modes:

1. GUI: *Pickering* runs with a GUI, where the user can test the interpreter. From here it is also possible to do batch processing and connect to the CTT Broker [http://www.speech.kth.se/broker/].
2. Web server: *Pickering* is started as a web server without GUI. To test the web server, open TestPickeringWebServer/index.html in a web browser (presently, you need to use Internet Explorer for the style-sheets to work).
3. Broker: *Pickering* is started as a client to the CTT Broker [http://www.speech.kth.se/broker/] without GUI.
4. Pipe: *Pickering* is started without GUI and reads strings to parse from STDIN and writes the result on STDOUT.

These modes are explained more thoroughly (e.g. what the input should look like) later in this chapter.

In all modes, Pickering can be terminated by feeding it an empty line on STDIN.

Pickering is configured through an XML configuration file.

Example 4.1. XML document: *Pickering* configuration file

```
<?xml version="1.0" encoding="iso-8859-1"?>
<module
        name="Pickering"
        xmlns="http://www.speech.kth.se/higgins/2003/pickering/config/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="
                http://www.speech.kth.se/higgins/2003/pickering/config/
                http://www.speech.kth.se/higgins/2003/pickering/config/
        >
   <send>
      <module>ParseResultReceiver</module>❶
   </send>
   <settings>
      <grammar_file>ExampleGrammar.xml</grammar_file>❷
      <mode>gui</mode>❸
      <permitted_top_rules>selected</permitted_top_rules>
      <permitted_top_entries>selected</permitted_top_entries>❹
      <permitted_insertions>2</permitted_insertions>
      <permit_weak_agreement>true</permit_weak_agreement>
      <style-sheets>
         <parse_result>❺
           <href type="text/xsl" name="Parse tree">
              http://www.speech.kth.se/higgins/2003/pickering/result/tr
           </href>
           <href type="text/xsl" name="XML original">
              http://www.speech.kth.se/higgins/xsl/xml2html.xslt
           </href>
         </parse_result>
         <grammar>
           <href type="text/xsl" name="XML original">
              http://www.speech.kth.se/higgins/xsl/xml2html.xslt
           </href>
         </grammar>
      </style-sheets>
      <webserver_port>80</webserver_port>❻
      <broker_host>bach</broker_host>
      <broker_port>2345</broker_port>❼
      <broker_gui_connect>false</broker_gui_connect>
      <broker_incremental_send>false</broker_incremental_send>❽
   </settings>
</module>
```

❶ In broker mode, the Broker Server name the results should be addressed to (see the CTT Broker documentation [http://www.speech.kth.se/broker/] for more information)

❷ The grammar file Pickering should start with.
❸ The mode in which Pickering should be run (see Section 4.1, “Configuring Pickering”). It takes the following values: `gui`, `webserver`, `pipe` or `broker`
❹ These settings affect *Pickering*s processing behaviour. permitted_top_rules an dpermitted_top_entries both take the values `all` (permit all rules or entries to be top nodes in the parse tree), `none` (permit no rules or entries to be top nodes in the parse tree), or `selected` (permit rules or entries with their `top` attribute set to `true` to be top nodes in the parse tree). permitted_insertions states how many insertions *Pickering* allows within phrases. Takes an integer as its value. 0 disables insertions of unmatched words. Finally, allow_weak_agreement takes a boolean value. It declares whether entries listed as agreeing shall be used even if they don't agree, unless there is another, better solution. If these switches are incomprehensible to you, disregard them for now and read the chapter on *Pickering* grammars.
❺ The contents of the style-sheets element declares which style sheets will be available in the GUI version. See the sections on visualisation for more info.
❻ In web server mode, specifies the port on which the web server should listen for connections.
❼ In broker mode, these specify which broker host and port to connect to. Port 1932 is registered with IANA [http://www.iana.org/] as the standard CTT Broker port.
❽ In GUI mode, broker_gui_connect specifies whether *Pickering* should connect to the broker at startup. broker_incremental_send specifies specifies if the results should be sent incrementally when connected to a broker.

4.2. Mozart/Tk GUI

This section describes the Mozart/Tk interface that ships with *Pickering*

4.2.1. Caveats

Platform dependency. The present version of *Pickering* comes with a Mozart/Tk graphical user interface. Currently, it is somewhat tied to WIndows. This is simply because parts of the implementation are in beta, and not yet coded machine independently. The intention is to fix this in future versions. The Windows dependency is tied to the following:

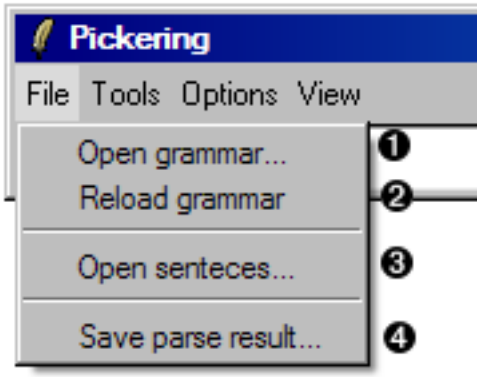
- The XSLT/CSS visualisations are currently displayed by letting a Perl loop invoke IE6 as a COM object. This solution is somewhat of a hack, and is meant to be recoded. Most of the style sheets used are browser independent, however, and an implementation may equally well use, say, Xalan or xslproc and Opera or Mozilla.

- Neither the Mozart/Oz code nor the Mozartl/Tk code is not yet tested under Linux, but there shouldn't be any major problems here - the language is quite platform independent.

4.2.2. Overall description

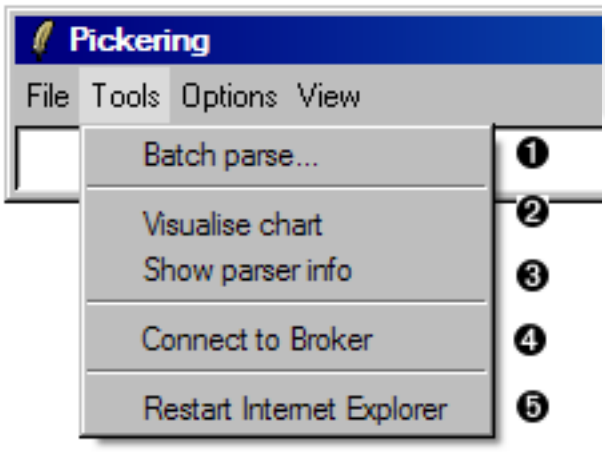
❶ This is the text input field. Text that is typed into this field will be interpreted, either incrementally after each space, or when the parse button is pressed.
❷ The parse button triggers parsing of the text in the input field.
❸ The send button sends the result to its recipient ofer the CTT broker, if one is connected.
❹ The reset button resets *Pickering* and clears the chart.

4.2.3. The File menu



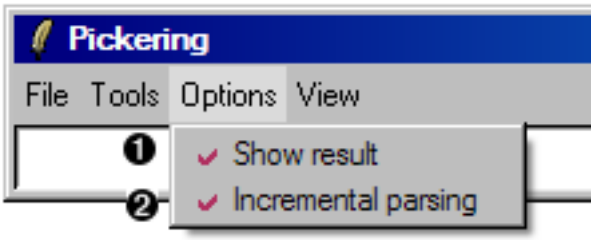
❶ Loads a new grammar file. The current grammar file is unloaded.
❷ Re-reads the current grammar from disk. Handy for grammar development.
❸ Loads a text file with pre-written sentences. The sentences are listed in the centre of the GUI, and when one of them is clicked it is instantly intertpreted. Very handy for grammar development.
❹ Saves the current parse result to file.

4.2.4. The Tools menu



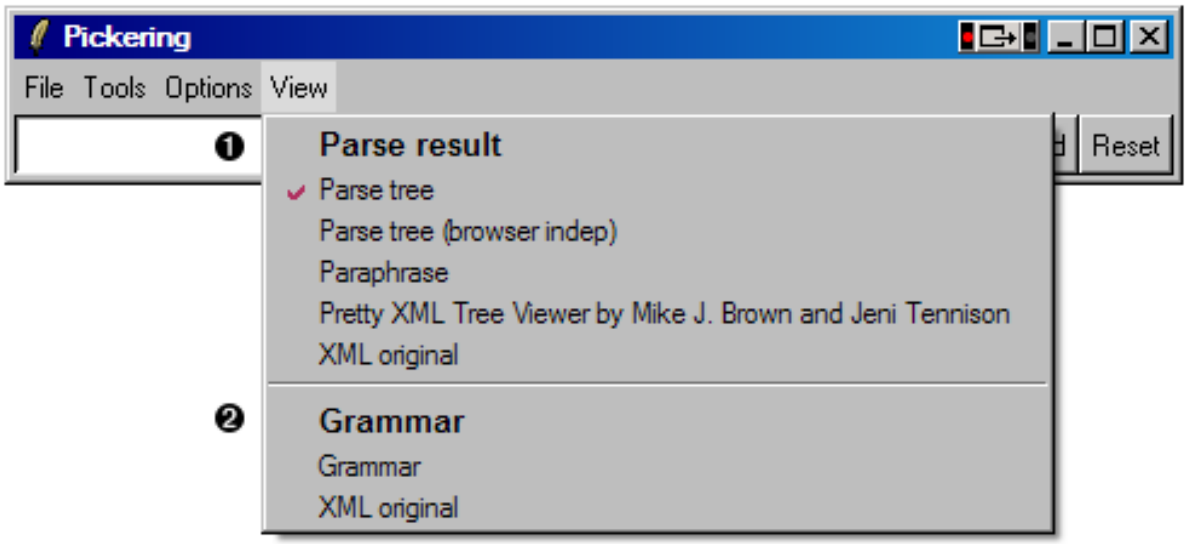
❶ Loads a text or XML file for batch processing.
❷ Displays a native Mozart/Tk visualisation of the current chart.
❸ Displays grammar statistics.
❹ Connects to a CTT Broker.
❺ Restarts Internet Explorer.

4.3. The Options menu



❶ When active,interpretations results are fed to Internet Explorer together with an XSL stylesheet of the users choice.
❷ When active, the interpreter works incrementally, word by word. When inactive, the interpreter only starts when a full utterance is given (or when the user clicks the parse button).

4.4. The View menu



❶ These are style sheets with which the current result may be viewed. The list is of style sheets is read from the configuration file, thus it can be set by the user. When one of these is selected, that style sheet is applied to any results displayed in the Internet Explorer browser window.
❷ These are style sheets with which the currentgrammar may be viewed. The list is of style sheets is read from the configuration file, thus it can be set by the user. Note that as long as one of these is selected, results are not shown in the Internet Explorer browser window - the grammar is, with the selected style sheet.

4.5. Web server interface

When started in web server mode, *Pickering* accepts calls on the standard HTTP port (80) and looks for the parameter `parsestring` in the request. It then interprets its contents and returns the resulting XML to the caller, for example a web browser. The distribution contains a very small web page to exemplify this functionality - just start *Pickering* in web mode and open the file index.html to see how it works.

Presentlly, there is no way to set the incoming port number for *Pickering*, so if you have a web server or something else running on port 80, this is not going to work. This will be fixed in some future version.

4.6. CTT Broker interface

The CTT Broker interface is what we use when we run *Pickering* within dialogue systems. The method is slightly more complex, and a newer release of the CTT Broker than the one available at present is recommended. Information and links to a newer broker release will be included here as soon as there is such a release.

4.7. Command line interface

When *Pickering* runs in pipe mode, it reads text line by line from STDIN until it gets a blank line (which causes it to close). The results are written to STDOUT.

Chapter 5. Overview of *Pickering* grammars

Highlights

- The *Pickering* grammar namespace
- First *Pickering* grammar example
- The `<grammar/>`, `<rules/>`, `<rule/>`, `<match/>` & `<semantics/>` elements
- The `link` attribute
- Extending a grammar with a lexicon
- Implementing a feature grammar
- Extending a lexicon with a morphology

In this chapter: This chapter briefly introduces the *Pickering* grammar formalism and the XML Schema it is described in. This is done by example - the chapter goes through a minimal grammar without getting into too much detail. XML Namespaces are mentioned briefly for completeness.

Grammar writing for *Pickering* is designed to be as flexible as possible. *Pickering* grammars are encoded in XML and the XML vocabulary used by *Pickering* is encoded in an XML Schema, which provides sufficient constraints to facilitate validation (syntax checking) of grammars.

In the most basic case, a *Pickering* grammar consists of nothing but rules. If a grammar is to be used for more complex domains, or in a system that will need to be maintained for any longer period of time, a rule-only grammar will become very cumbersome. *Pickering* offers several methods to increase the flexibility and to ease the construction and maintainance of grammars. Two of them, lexicons and morphologies, are described briefly in this chapter.

A common feature of grammars is to attach feature value pairs on rules, lexicon entries and/or morphology. Linguistic features, for example are often utilised in this manner. *Pickering* supports optionally supports feature grammars, and the basic methods for using them are also described here.

5.1. XML basics

Pickering grammars are encoded in XML and each XML document related to *Pickering* is required to be *well-formed* and *valid*. There are some details about XML files that, although not direclty related to *Pickering*, one must be aware of to understand and use *Pickering*.

XML declarations: It is a good idea to make a habit of placing an XML declaration at the top of every XML document. It tells browsers and tools that the document is intended to be well-formed XML and nothing else, which XML version it is, and which character encoding the document uses. In most cases, and definitely in the case of Swedish documents, the `ISO-8859-1` encoding is the prudent choice. The XML declaration then, looks like this: `<?xml version="1.0" encoding="ISO-8859-1"?>`

XML Namespaces: The XML vocabulary used for a grammar is defined in several XML Namespaces. A namespace is a strictly and globally defined "home" for an XML vocabulary. In this chapter, we will talk briefly about the most essential of these, from a *Pickering* point of view: the `http://www.speech.kth.se/higgins/2003/pickering/grammar/` namespace.

The *Pickering* grammar specification states that every element must have its namespace explicitly declared. In XML this is achieved in two steps:

1. Each namespace as declared. The declaration is usually placed on the top level element of the XML document (the `<grammar/>` element in the case of *Pickering* grammars), and is done using the `xmlns` attribute. A generic namespace declaration looks something like this:

Example 5.1. XML snippet: a generic namespace declaration

```
<ELEMENT❶
  xmlns:NS="UNIQUE_NAMESPACE_ID ❷"/>
                                  ❸
```

❶ Often the top-level element.
❷ `UNIQUE_NAMESPACE_ID` is the unique namespace identifier, for example `http://www.speech.kth.se/higgins/2003/pickering/grammar/`, goes here.
❸ The namespace declaration can be seen as a way of locally aliasing the (often very long) unique namespace identifier. `NS` in this example is the alias, which should preferrably be short.

2. The alias (`NS` above) is used to explicitly qualify elements belonging to the namespace in question by prefixing it to the elements. A colon (:) is used to separate the namespace alias and the element name: `NS:ELEMENT`. In the code examples throughout this book, `g` is used as an alias for the *Pickering* grammar namespace (`http://www.speech.kth.se/higgins/2003/pickering/grammar/`), so the top level grammar element is written `<g:grammar/>`. (Note that explicit namespace declarations are not used in the running text of this book, so inline, the same element is written `<grammar/>`.)

XML Schema declarations: In order to validate XML it must be connected to a formal definition (a grammar for the XML if you will). This can be done in a number of ways (DTDs being

the most common to date). *Pickering* XML is specified in XML Schemas. An XML documented is associated with a schema using the `xsi:schemaLocation` attribute on the top level element. The attribute takes string pairs as values, the first of a pair being the unique identifier (not the alias) of the namespace being specified in the schema, the second an URL (or an URI to be correct) to the XML Schema document holding the specification. More than one pair may be given, if more than one namespace is used. The generic example looks like this:

Example 5.2. XML snippet: a generic XML Schema declaration

```
<TOP_ELEMENT ❶
  xsi:schemaLocation="UNIQUE_NAMESPACE_ID ❷
          SCHEMA_URI ❸ "/>
```

❶ Schemas are associated with an XML document using the `xsi:schemaLocation` attribute on the top-level element. The attribute takes string pairs as its value.

❷ The first part of each value pair is a unique namespace identifier, given as `UNIQUE_NAMESPACE_ID` here.

❸ The second part of each value pair is a URI reference to the file containing the XML Schema, given as `SCHEMA_URI` here.

5.2. Writing a simple *Pickering* grammar

A minimal *Pickering* grammar consists of a set of rules. Each rule has one or more `<match/>` elements and a `<semantics/>` element (will be `<semantics>` in later versions). `<match/>` elements specify what the rule will match against and the `<semantics/>` element specifies the semantics that the rule will return. Here is a simple example from Swedish:

Example 5.3. XML example: a minimal grammar

```
<?xml version="1.0" encoding="ISO-8859-1"?> ❶
<g:grammar
 xmlns:g="http://www.speech.kth.se/higgins/2003/pickering/grammar/" ❷
 xmlns:f="http://www.speech.kth.se/higgins/2003/pickering/grammar/fea
 xmlns:w="http://www.speech.kth.se/higgins/2003/world/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation=" ❸
  http://www.speech.kth.se/higgins/2003/pickering/grammar/

                                          http://www.speech.kth.se/higg

">
 <g:rules> ❹
 <g:rule f:name="landmark"> ❺
  <g:match> ❻
```

```
   en <g:rule f:name="prop" link="1"/> lokal ❼
  </g:match>
  <g:match>
     en <g:rule f:name="prop" link="1" ❽ /> byggnad
  </g:match>
  <g:sem> ❾
   <w:object xsi:type="building">
    <w:properties>
     <g:ref link="1" ❿ />
    <w:/properties>
   </w:object>
  </g:sem>
 </g:rule>
 <g:rule f:name="prop">
  <g:match>gul</g:match>
  <g:sem>
   <w:colour>yellow</w:colour>
  </g:sem>
 </g:rule>
 <g:rule f:name="prop">
  <g:match>stor</g:match>
  <g:sem>
   <w:size>large</w:size>
  </g:sem>
 </g:rule>
</g:rules>
</g:grammar>
```

❶ Standard XML declaration.

❷ The `<grammar/>` top element carries XML Namespace and XML Schema declarations in attributes. The Pickering grammar XML Namespace is `http://www.speech.kth.se/higgins/2003/pickering/grammar/`. Everything that has to do with the grammar specification lies in this namespace. The resulting semantics should be placed in another namespace. In the example, the semantics are placed in `http://www.speech.kth.se/higgins/2003/world/`, but this is up to the grammar writer to decide.

❸ This calls in the Pickering grammar XML Schema.

❹ Grammar rules are always defined within a `<rules/>` element.

❺ Each grammar rule is defined with a `<rule/>` element.

❻ A `<rule/>` element may contain several alternative `<match/>` elements.

❼ A `<match/>` elements may contain `<rule/>` elements, text, or a combination of both.

❾ Only one `<semantics/>` element is allowed. The element defines the semantics produced by the rule.

❽❿ The semantics of a matching rule can be copied into the resulting semantics with the `link` attribute, which tells the interpreter to unify each rule or entry with the `link` attribute set, using the attribute values as ordering numerals. (See the chapter on semantic unification.)

In example above, the parsing of the string "en stor byggnad" would result in the following semantics:

Example 5.4. XML snippet: resulting semantics

```
<w:object xsi:type="building" ❶ >
 <w:properties>
  <w:size>large</w:size> ❷
 <w:/properties>
</w:object>
```

❶ `xsi:type` is a special attribute from the XML Schema namespace. It allows the author to use the same top element within the same namespace to represent non-identical structures. In the *Higgins* domain, it is used to classify objects, since different object types tend to have different properties.

❷ The exact mechanism that merges the semantics from various matching rules are explained later in this documents.

5.3. Adding a lexicon

A *Pickering* grammar can be extended with a lexicon. The lexicon consists of a set of entries, each of which looks just like a rule in the grammar, except that the `<match/>` element in an entry may only contain a textual words - no references to other entries or rules may be included. The lexicon entries correspond to the leaves of a CF.

Using a lexicon has several advantages. From a maintainance point of view, placing all literal words in one place makes it a lot easier to overview, extend and modify a grammar. There is also a significant efficiency involved from a processing point of view: using more entries and less rules permits the implementation to perform more bottom-up processing before the rules are applied.

Here is an example of a lexicon:

Example 5.5. XML example: a minimal lexicon

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<g:grammar
 xmlns:g="http://www.speech.kth.se/higgins/2003/pickering/grammar/"
 xmlns:f="http://www.speech.kth.se/higgins/2003/pickering/grammar/fea
 xmlns:w="http://www.speech.kth.se/higgins/2003/world/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
  http://www.speech.kth.se/higgins/2003/pickering/grammar/
   http://www.speech.kth.se/higgins/2003/pickering/grammar/grammar.xs
">
 <g:lexicon> ❶
  <g:entry f:name="prop">
   <g:match>gult</g:match>
    <g:sem>
     <w:colour>yellow</w:colour>
    </g:sem>
   </g:entry>
   <g:entry f:name="prop">
```

```
   <g:match>stort</g:match>
   <g:sem>
    <w:size>large</w:size>
   </g:sem>
  </g:entry>
   <g:entry f:name="prop">
   <g:match>stor</g:match>
   <g:sem>
    <w:size>large</w:size>
   </g:sem>
  </g:entry>
 </g:lexicon>
</g:grammar>
```

❶ Lexicon entries are always defined within a `<lexicon/>` element. The element can either sit in the same file as the grammar `<rules/>` element, or be defined in a separate file, using *Pickering*'s modularisation methods. These are explained elsewhere.

The rules in a grammar may then reference lexicon entries in the same manner as they do other rules:

Example 5.6. XML snippet: referencing a lexicon entry

```
<g:rule f:name="examplerule">
 <g:match>
  en <g:entry f:name="prop" link="1" ❶ /> byggnad
 </g:match>
 <g:sem>
  <g:unify/>
 </g:sem>
</g:rule>
```

❶ When a lexicon is used, rules may reference lexicon entries.

5.4. Feature grammars

In the previous examples, the matching of rules and entries has been done on the bases of the attribute `f:name`. However, the rules and entries may be extended with an arbitrary number of features, grammatical or otherwise.

The features be given any name, and any number of them may be added. All features listed on an entry or rule in a `<match/>` element must match for the match to be applicable. Note also that all features are placed in the `http://www.speech.kth.se/higgins/2003/pickering/grammar/feature/` namespace, which is an open namespace permitting any attribute name. The feature attributes must be qulified (i.e. have their namespace explicitly stated), however. An example follows:

Example 5.7. XML snippet: linguistic features in a lexicon

```
<g:lexicon>
 <g:entry f:name="noun" f:number="sing" f:definiteness="indef" f:genc
  <g:match>byggnad</g:match>
  <g:sem><w:object xs:type="building"/><g:/sem>
 </g:entry>
 <g:entry f:name="noun" f:number="sing" f:definiteness="def" f:gender
  <g:match>byggnaden</g:match>
  <g:sem><w:object xs:type="building"/><g:/sem>
 </g:entry>
 <g:entry f:name="noun" f:number="plur" f:definiteness="indef" f:genc
  <g:match>byggnader</g:match>
  <g:sem><w:object xs:type="building"/><g:/sem>
 </g:entry>
 <g:entry f:name="noun" f:number="plur" f:definiteness="def" f:gender
  <g:match>byggnaderna</g:match>
  <g:sem><w:object xs:type="building"/><g:/sem>
 </g:entry>
</g:lexicon>
```

These features can be used in referring rules, just like the `name` attribute was in the previous examples:

Example 5.8. XML snippet: matching features in a rule

```
<g:match>
 en <g:entry f:name="prop" f:gender="utr" link="1"/> byggnad
</g:match>
```

Finally, the features may also be transferred into the resulting semantics. Number and definiteness, for example, may be of semantic significance, depending on what the interpreter is used for. The methods used to accomplish this are described later.

5.5. Adding a morphology

We have seen an example of how a word in different forms can be represented in a lexicon and how features can be used on `<match/>` elements.. If different word forms have different features but identical semantics, encoding them over and over is cumbersome and unnecessary. Instead, the lexicon can be extended with a morphology. The morphology consists of `<morphology/>` element containing a set of `<declination/>` elements, which in turn contains `<form/>` elements. Each declination form describes how a suffix change the linguistic features (see below) of an entry. The suffix itself is written as element content, and the features are encoded as attributes, just as in a lexicon entry.

Example 5.9. XML example: a morphology

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<g:grammar
 xmlns:g="http://www.speech.kth.se/higgins/2003/pickering/grammar/"
 xmlns:f="http://www.speech.kth.se/higgins/2003/pickering/grammar/featu
 xmlns:s="http://www.speech.kth.se/higgins/2003/pickering/semantics/"
 xmlns:a="http://www.speech.kth.se/higgins/2003/annotation/"
 xmlns:w="http://www.speech.kth.se/higgins/2003/world/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
  http://www.speech.kth.se/higgins/2003/pickering/grammar/
   http://www.speech.kth.se/higgins/2003/pickering/grammar/grammar.xsd
  http://www.speech.kth.se/higgins/2003/pickering/grammar/feature/
   http://www.speech.kth.se/higgins/2003/pickering/grammar/feature/feat
">
 <g:morphology  ❶ >
  <g:declination id="noun1"  ❷ >
   <g:form f:number="sing" f:definitness="indef"/>  ❸
   <g:form f:number="sing" f:definitness="def">en  ❹ </g:form>
   <g:form f:number="plur" f:definitness="indef">er</g:form>
   <g:form f:number="plur" f:definitness="def">erna</g:form>
  </g:declination>
 </g:morphology>
</g:grammar>
```

❶ A morphology is always defined within a `<morphology/>` element.

❷ Each declination is always defined with a `<declination/>` element. The element carries a mandatory `id` attribute, which must be a unique identifier within the entire grammar. The `id` attribute is used to connect lexicon entries to their declinations.

❸❹ A `<declination/>` element contains a `<form/>` element for each suffix the lexicon entry should be expanded with, and may also contain an empty `<form/>` element, representing the word as it is written in the lexicon entry. The attributes listed on each form are transferred to the expanded lexicon entry.

Example 5.10. XML snippet: referencing a declination

```
<g:entry f:name="noun" f:gender="utr" declination="noun1"  ❶ >
 <g:match>byggnad</g:match>
 <g:sem><w:object xs:type="building"/><g:/sem>  ❷
</g:entry>
```

❶ The morphology is linked into the lexicon by extending the lexicon with the `declination` attribute, the contents of which referes to a declination `id` in the morphology.

❷ Even though one can expand a lexicon entry into several by linking it to a declination, it

can only have one set of semantics.

If the same feature is encoded both on the lexicon entry and the declination form, the declination form takes precedence. The result of the code in the examples above would be that the entry is expanded to four entries, one for each `<form/>` element in the declination.

Chapter 6. Building semantics

Highlights

- *Pickering* results
- Semantic constructions
- Unification of semantics
- Templates and the `<template/>` element
- Controlling unification

In this chapter: The `<semantics/>` element is mandatory in `<rule/>` and `<entry/>` definitions. When the rule or entry is applied, the contents of the `<semantics/>` element is used to build the resulting semantics. The examples in the introductory chapters on rules and lexicons have showed how to encode new semantics and how to copy semantics from matching nodes with the `<unify><ref link="1"/></unify>` construction and its abbreviated form, `<unify/>`. In this chapter we give a more thorough description of how *Pickering* builds semantics.

6.1. Unification and semantic templates

Sometimes, the semantics of a matching part needs to be subordinated the semantics of another. Consider the following grammar (for clarity, we don't use a morphology):

Example 6.1. XML example: building semantics

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<g:grammar
 xmlns:g="http://www.speech.kth.se/higgins/2003/pickering/grammar/"
 xmlns:f="http://www.speech.kth.se/higgins/2003/pickering/grammar/fea
 xmlns:s="http://www.speech.kth.se/higgins/2003/pickering/semantics/"
 xmlns:a="http://www.speech.kth.se/higgins/2003/annotation/"
 xmlns:w="http://www.speech.kth.se/higgins/2003/world/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
  http://www.speech.kth.se/higgins/2003/pickering/grammar/
   http://www.speech.kth.se/higgins/2003/pickering/grammar/grammar.xs
">
 <g:rules>
  <g:rule f:name="landmark">
   <g:match> ❶
    <g:entry f:name="art"/>
```

```
    <g:entry f:name="prop"/>
    <g:entry f:name="lm"/>
   </g:match>
   <g:sem>
    <!--  ❷ semantics go here -->
   </g:sem>
  </g:rule>
 </g:rules>
 <g:lexicon>
  <g:entry f:name="art" f:number="sing" f:gender="utr" f:info="new"> ❸
   <g:match>en</g:match>
   <g:sem><w:object/></g:sem>
  </g:entry>
  <g:entry f:name="prop" f:number="sing" f:gender="utr" f:info="new"> ❹
   <g:match>stor</g:match>
   <g:sem><w:size>large</w:size></g:sem>
  </g:entry>
  <g:entry f:name="lm" f:number="sing" f:gender="utr" f:info="new"> ❺
    <g:match>byggnad</g:match>
    <g:sem><w:object xsi:type="building"/></g:sem>
  </g:entry>
 </g:lexicon>
</g:grammar>
```

❶ The only rule in this grammar describes a noun phrase. It contains a match element that needs three entries to match.

❷ What we may put here is the topic of discussion in the next few paragraphs.

❺ An article, an adjective and a noun is needed. In Swedish, they all carry information on gender, number and definitness, or new/given status if you will.

In the example above, we've left the semantics for the rule identified by `f:name="landmark"` empty. Depending on the application, we may want the resulting semantics to be represented in different ways. Here is an example:

Example 6.2. XML snippet: structured semantics

```
<w:object xsi:type="building">
        <w:properties>
                <w:size>large</w:size>
        </w:properties>
</w:object>
```

Pickering leaves us with a lot of freedom to build any kind of semantics we want. This is not achieved by using lambda expressions, but is based on unification with semantic templates. We start by writing a semantic template:

Example 6.3. XML example: a semantic template

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<g:grammar
 xmlns:g="http://www.speech.kth.se/higgins/2003/pickering/grammar/"
 xmlns:s="http://www.speech.kth.se/higgins/2003/pickering/semantics/"
 xmlns:a="http://www.speech.kth.se/higgins/2003/annotation/"
 xmlns:w="http://www.speech.kth.se/higgins/2003/world/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
  http://www.speech.kth.se/higgins/2003/pickering/grammar/
   http://www.speech.kth.se/higgins/2003/pickering/grammar/grammar.xs
">
 <g:template> ❶
❷<w:object count="*"❸ >
   <w:properties count="1" ❹ >
    <w:size count="1"/>
   </w:properties>
  </w:object>
 </g:template>
</g:grammar>
```

❶ The template is defined in a `<template/>` element, placed directly under the `<grammar/>` top level element

❷ The template contains a simplistic description of the semantic model, which should be kept in a separate namespace.

❸❹ The resulting semantics should be arranged in the same way as the the template. The `count` attribute tells us how many times an element may be repeated. A Kleene star (*) is used if an element may be repeated indefinitely.

The `f:name="landmark"` rule above may now be written as follows:

Example 6.4. XML example: building semantics

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<g:grammar
 xmlns:g="http://www.speech.kth.se/higgins/2003/pickering/grammar/"
 xmlns:f="http://www.speech.kth.se/higgins/2003/pickering/grammar/featu
 xmlns:s="http://www.speech.kth.se/higgins/2003/pickering/semantics/"
 xmlns:a="http://www.speech.kth.se/higgins/2003/annotation/"
 xmlns:w="http://www.speech.kth.se/higgins/2003/world/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
  http://www.speech.kth.se/higgins/2003/pickering/grammar/
   http://www.speech.kth.se/higgins/2003/pickering/grammar/grammar.xsd
">
 <g:template> ❶
  <w:object count="*">
   <w:properties count="1">
    <w:size count="1"/>
   </w:properties>
  </w:object>
 </g:template>
 <g:rules>
  <g:rule f:name="landmark">
   <g:match>
    <g:entry f:name="art"/>
    <g:entry f:name="prop" link="2"/>
    <g:entry f:name="lm" link="1"/>
   </g:match>
   <g:sem>
    <g:unify/> ❷
   </g:sem>
  </g:rule>
 </g:rules>
 <g:lexicon>
  <g:entry f:name="art" f:number="sing" f:gender="utr" f:info="new">
   <g:match>en</g:match>
   <g:sem><w:object/> ❸ </g:sem>
  </g:entry>
  <g:entry f:name="prop" f:number="sing" f:gender="utr" f:info="new">
   <g:match>stor</g:match>
   <g:sem><w:size>large</w:size> ❹ </g:sem>
  </g:entry>
  <g:entry f:name="lm" f:number="sing" f:gender="utr" f:info="new">
    <g:match>byggnad</g:match>
    <g:sem><w:object xsi:type="building"/> ❺ </g:sem>
  </g:entry>
 </g:lexicon>
```

```
</g:grammar>
```

❺ Although the semantics of the entries above cannot be easily unified on their own...
❷ ...it is possible to see how the should be unified...
❶ ...in the light of the template.

The `<unify/>` element above says that all `link`-ed nodes under the `<match/>` element is to be unified in the order stated by their `link` attributes. The order is important, because the second unifying element will have to be placed on the same level or on a level subordinated to the first unifying element.

6.2. Controlling unification

The `<unify/>` element can contain semantics that are stated directly inside it, as well as references to the resulting semantics of matching rules and entries. You may put semantic nodes inside the element for better control of what gets unified and how it is done. For example, you may want to write:

Example 6.5. XML snippet: example `<unify/>` element

```
<g:unify>
        <w:object/>
        <g:ref link="2"/>
</g:unify>
```

In the example above, the literally stated `<w:object/>` and the semantics referenced by `<g:ref link="2"/>` are the only semantics that will be unified. Any other linked semantics are ignored.

Chapter 7. Visualisation

Jens Edlund <edlund@speech.kth.se>

Highlights

- Grammar statistics
- Visualised grammars
- Visualised interpretation results
- Adding custom visualisations

In this chapter: *Pickering* comes with a number of XSLT and CSS style sheets which can be used to visualise grammars, parse trees and resulting semantics. This section briefly introduces the included standard style sheets, and shows how additional, custom style sheets can be included.

7.1. Visualisation of input

TODO

7.2. Grammar visualisation

TODO

7.3. Parse tree visualisation

TODO

7.4. Visualisation of resulting semantics

TODO

Glossary

ASR

Automatic Speech Recognition: collective term for a number of methods to convert an audio speech signal into a textual representation of what was said.

CSS

Cascading Style Sheets: a simple mechanism for adding layout, for example fonts, colors, and spacing, to Web documents[2]. CSS documentation can be found at http://www.w3.org/Style/CSS/.
See Also W3C.

CTT

Centrum fAAAAAr Talteknologi (Centre of Speech Technology): Speech lab at KTH, Stockholm, Sweden. CTT can be found at http://www.speech.kth.se/ctt/.
See Also KTH.

DocBook

DocBook: an *XML*/*SGML* vocabulary for encoding books and papers [3].
See Also SGML, XML.

DocBook XSL

DocBook XSL: *XSL* documents for transforming *DocBook* into *HTML*, and *XSL-FO*, which in turn can be used to produce *PDF* and *PostScript* documents[4].
See Also DocBook, XSL.

DTD

Document Type Definition: a type definition for an XML or SGML file[5].
See Also SGML, XML.

GUI

Graphical User Inteface: a visible interface between a human and a computer, often consisting of monitor, keyboard and mouse.

KTH

Kungliga Tekniska Högskolan (Royal Institute of Technology: Technical University in Stockholm, Sweden. KTH can be found at http://www.kth.se/.

Mozart

The Mozart Programming System: an advanced development platform for distributed applications (http://www.mozart-oz.org/).
See Also Oz.

[2] The definitions have been shortened and rewritten, but their contents are taken from W3C (http://www.w3.org/).
[3] The DTDs for DocBook are maintained by OASIS (http://www.oasis-open.org/), and are available for free. A good manual can be found at http://docbook.org/, and there's a WIKI at http://www.docbook.org/wiki/moin.cgi/
[4] The style sheets and documentation are developed as a SourceForge.net (http://sourceforge.net) project, and can be downloaded from the project home page (http://docbook.sourceforge.net/projects/xsl/). For a comprehensive manual, see ???.
[5] The DTD format is specified in ISO 8879:1986 (the SGML specification), which is owned by *ISO* and not available for free. A good tutorial can be found at http://www.xmlfiles.com/dtd/

OOV

Out Of Vocabulary: word that is not contained in the vocabulary of a speech application (e.g. an *ASR*, parser or semantic interpreter)

Oz

The Oz Programming Language: a programming language supporting declarative programming, object-oriented programming, constraint programming, and concurrency as part of a coherent whole (http://www.mozart-oz.org/).
See Also Mozart.

PDF

Portable document format: is a computer file format designed to publish and distribute electronic documents, developed by Adobe®. The company's blurb is available at http://www.adobe.com/products/acrobat/adobepdf.html.
See Also PostScript.

Perl

Practical extraction and report language: Open Source programming language first released by Larry Wall in 1987. Perl sources are available through http://www.perl.com/.

PostScript

PostScript: a programming language optimized for printing graphics and text and introduced by Adobe® in 1985. The company's blurb is available at http://www.adobe.com/products/postscript/main.html.

SDS

Spoken Dialogue System: a computer system using *ASR* and synthetic speech to communicate with a human.

TMH

Tal, Musik & Hörsel (Speech, Music & Hearing): Speech Technology department at KTH, Stockholm, Sweden. TMH can be found at http://www.speech.kth.se/.
See Also KTH.

URI

Uniform Resource Identifier: A complicated way of saying "internet address", defined in *RFC2396* (http://www.gbiv.com/protocols/uri/rfc/rfc2396.html).
See Also URI.

URL

Uniform Resource Locator: A complicated, and nowadays formally abandoned, way of saying "internet address", defined in the now obsolete *RFC1738* (http://www.gbiv.com/protocols/uri/rfc/rfc2396.html#RFC1738) and *RFC1808* (http://www.gbiv.com/protocols/uri/rfc/rfc2396.html#RFC1808).

Modern definitions are listed under *URI*.
See Also URI.

VINNOVA

Swedish Agency for Innovation Systems: an agency whose mission it is to promote growth by financing the development of innovation systems.. VINNOVA can be found at http://www.vinnova.se/.

W3C

the World Wide Web Consortium: created in October 1994, the W3C develops common protocols that promote the evolution of the Web and ensure its interoperability[2]. W3C can be found at http://www.w3.org/.

XHTML

(eXtensible) HyperText Markup Language: Designed by Tim Berners-Lee some decades ago, HTML and its predecessors remain the leading WWW markup. Modern HTML (XHTML) is formally specified in terms of XML. HTML documentation can be found at http://www.w3.org/MarkUp/.
See Also XML, W3C.

XML

Extensible Markup Language: a simple and flecible markup language derived from SGML[2]. XML documentation can be found at http://www.w3.org/XML/.
See Also SGML, W3C.

XML Namespace

Extensible Markup Language Namespace: a simple method for qualifying element and attribute names used in *XML* documents by associating them with namespaces [2]. Information can be found at http://www.w3.org/TR/2004/REC-xml-names11-20040204/.
See Also XML, W3C.

XML Schema

Extensible Markup Language Schema: a means for defining the structure, content and semantics of *XML* documents [2]. Information can be found at http://www.w3.org/XML/Schema.
See Also XML, W3C.

XSL

Extensible Stylesheet Language: a family of W3C recommendations for defining XML document transformation and presentation, comprised by XSLT, XPATH, and XSL-FO[2]. XSL documentation can be found at http://www.w3.org/Style/XSL/.
See Also XML, W3C.

XSLT

Extensible Stylesheet Language (XSL) Transformations: a language in the XSL family for transforming XML documents into

other XML documents[2]. XSLT documentation can be found at http://www.w3.org/Style/XSL/.
See Also XML, XSL, W3C.

Index

References

Bob Stayton (2003): DocBook XSL: The Complete Guide. Retrieved 2004-07-20 from http://www.sagehill.net/docbookxsl/

Norman Walsh & Leonard Muellner (2001): DocBook: The Definitive Guide. Retrieved 2004-07-20 from http://www.oreilly.com/catalog/docbook/chapter/book/docbook.html