Heart of Gold – an XML-based middleware for the integration of deep and shallow natural language processing components

User and Developer Documentation

September 2009 (V1.5 and SVN trunk)

Ulrich.Schaefer@dfki.de

http://heartofgold.dfki.de

http://www.delph-in.net/heartofgold

Contents

Part 1: Overview..................................................................................................................3
  Module Communication Manager...............................................................................4
  Modules and Components.........................................................................................4
  Sessions and Annotations .....................................................................................5
  NLP Analysis.............................................................................................................6
  Metadata...................................................................................................................6
  Annotation database...............................................................................................7
  Annotation Transformation with class TransformationService.........................8
Part 2: Installation of the Middleware........................................................................9
  Release Notes for Version 1.5..................................................................................9
  Installation, File System Structure........................................................................12
  Sample Installation and Test..................................................................................13
  The Ant Build File build.xml...............................................................................15
Part 3: Implementing Modules and Applications......................................................16
  Writing an application............................................................................................16
  Writing a Module...................................................................................................17
  Transformation Service..........................................................................................18
  Writing XML-RPC Modules and Adapters............................................................19
  Utility class.............................................................................................................19
Part 1: Overview

The Heart of Gold¹ is an XML-based middleware for the integration of deep and shallow natural language processing components. It provides a uniform and flexible infrastructure for building applications that use RMRS-based and/or XML-based natural language processing components.

The main design goals were:
- flexible integration of NLP components
- simple application interface
- RMRS as uniform representation language (XML-encoded)
- open to other XML standoff annotation formats
- integration of non-RMRS-aware NLP components through annotation transformation
- external annotation database for storage and retrieval of computed linguistic analyses (optional)
- network-enabled architecture with distributed components (optional)
- lightweight, platform- and programming language-independent communication through XML-RPC
- based on current technology like XML, XML-RPC, XSLT, XML:DB, XPath

The figure on the next page depicts the general architecture. Heart of Gold acts as mediator between applications and NLP components, abstracting from component-specific interfaces and representations. Applications send queries on text documents to the middleware which in turn passes the queries to one or more components according to an application-specific configuration. The resulting annotation, which can also be taken from the annotation database, is then returned to the application.

¹ for related literature, see http://www.bbc.co.uk/cult/hitchhikers/guide/heartofgold.shtml
Module Communication Manager

Applications communicate with the Heart of Gold middleware through the Module Communication Manager (MoCoMan) via a Java API (Java applications) or XML-RPC (remote applications or applications written in programming languages other than Java).

XML-RPC is a lightweight protocol that is supported – through additional libraries – by most current programming languages on various platforms. It is built on top of HTTP and hence can also be used for communication through firewalls which otherwise would have to be opened for specific ports other than the standard HTTP port. For the same reasons, XML-RPC can also be used for the integration of natural language processing components into the architecture. In other words, XML-RPC provides an easy and portable means to network-enable architecture and applications.

Modules and Components

Initially, an application starts an instance of the Heart of Gold architecture with a configuration setting for the required components. The MoCoMan then starts (or remotely connects to) the appropriate components. From the viewpoint of MoCoMan, components are Modules (local Java-based components) or XmlRpcModules (remote, possibly non-Java, components). I.e., in order to integrate a new component in the architecture, it must
inherit from either Module or XmlRpcModule. The remote counterparts of an XmlRpcModule are called adapters. A Java class can be used to implement remote Java adapters. Moreover, remote adapters can also be implemented in other programming languages and communicate with MoCoMan via XML-RPC. The Module configuration, abstract Module and Registry classes are based on some basic classes of the Memphis architecture [Memphis]. If a component does not provide XML or RMRS output, then translation to and (possibly) from XML or RMRS should be implemented in the Module classes.

Sessions and Annotations

MoCoMan provides a session management so that different input sessions with multiple documents (input texts) can be referenced. MoCoMan manages a collection of sessions where a session consists of a collection of annotation collections (one annotation collection corresponds to one input text or sentence) that contain RMRS/standoff annotations computed by the components of the configured architecture instance. Sessions, annotation collections and annotations are referenced through context-unique IDs. Sessions, annotation collections and computed annotations can optionally be stored in an XML database.

*Drawing 2 Structured annotation storage*
NLP Analysis

After the system configuration is finished, analysis requests can be passed to the MoCoMan. Analysis request parameters comprise
- (initially) an input text or (later) a reference to the input text
- language
- depth of requested analysis (integer)
- token span [not implemented yet; will probably become character span instead]

MoCoMan passes the request to the modules that are configured in the architecture instance and that are appropriate for the requested depth of analysis and language. The RMRS annotations computed by the modules are then returned to the application (and optionally stored in the XML database).

If a query is passed to the MoCoMan that has already been computed (i.e., with the same input text and query parameters), then the pre-computed result is returned by MoCoMan.

---

Default Processing Strategy

- Shallowest component first (e.g. tokenizer).
- Then other components with increasing depth, up to requested depth.
- Fallback to result of previous component if no result from component with requested depth.
- Each component gets the output of previous component as input plus the output from other components if configured.
- The result of the query is the result of the deepest component in the sequence.
- Analyses results from previous components are returned on request (getAnnotation()).

---

Metadata

Metadata on date, time, source, parameters, and processing options of annotation are stored together with the session and annotation collections. Metadata can contain structured XML, e.g.

```xml
<metadata>
  <id>
    <entry name="acid" value="collection0002"/>
    <entry name="component" value="PET"/>
    <entry name="created" value="Do, 4 Dez 2003 18:25:16 +0100"/>
    <entry name="processingtime" value="00:08,140"/>
    <entry name="sessionid" value="session0001"/>
    <entry name="diagnosis" value="OK"/>
  </id>
  <conf>
    <entry name="module.rootelement" value="pet"/>
    <entry name="module.language" value="en"/>
    <entry name="module.depth" value="100"/>
  </conf>
</metadata>
```
diagnosis String starts with either "OK" or "Error".

DTD fragment of Metadata:

<!ELEMENT metadata ( id , conf )>
<!ELEMENT id (entry)* >
<!ELEMENT conf (entry)* >
<!ELEMENT entry EMPTY >
<!ATTLIST entry name NMTOKEN #REQUIRED
value CDATA #REQUIRED >

Annotation database

The Heart of Gold middleware optionally provides a database interface for XML annotation storage. The main purpose is persistent storage of computed annotations for the automatic creation or enrichment of linguistic corpora etc.

The annotation database interface uses XML:DB [XML:DB] which is a vendor-independent interface to XML databases. The current implementation uses the free (but disappointingly slow) Xindice 1.1b [Xindice], but other XML databases such as dbxml, Tamino could be used instead. An interface class is provided that can be implemented in order to support other XML databases.

The Heart of Gold XML database interface (abstract class AnnotationDatabase) supports organisation of XML annotation reflecting the session and annotation collection tree hierarchy of MoCoMan. Standard operations like inserting, deleting collections and XML annotations, and a standardized query language based on XPath [XPath] is supported. Existing annotation can also be modified using the XUpdate query language [XUpdate]. However, this is currently not actively supported by the XML:DB interface.

An important feature of XML databases is indexing of XML document elements with respect to efficient retrieval. Depending on the structure of the annotation, indexers can be defined through the database interface. This should be done when integrating new Modules and can be stored as part of the Module configuration which in turn is part of the annotation metadata.

In the current implementation based on Xindice, the XML database can easily be separated physically from the rest of the architecture. The database can reside on a server different from the middleware server and provide its services through the tomcat web application server. The XML database interface of MoCoMan then acts as a client to the XML database.
For most practical cases, a simple storage and retrieval mechanism based on annotation ID and annotation collection ID is sufficient. It can (and probably will in the future) be implemented on file system basis being a subclass of AnnotationDatabase.

**Annotation Transformation with class TransformationService**

For the integration of non-RMRS-aware components, XSLT [XSLT] can be employed in order to transform component-specific XML output, e.g., of a chunker or a named entity recognition component, into the RMRS format (cf. [WHAT]). The transformation is performed in the module’s process() method. A special class with name TransformationService provides to the standard XSLT transformation with an extension for seamless access to Heart of Gold annotation from within XSLT stylesheets.

*Illustration 1 UML diagram of the Heart of Gold core classes*
Part 2: Installation of the Middleware

Release Notes for SVN trunk Version

V1.5 has moved to tags/1.5 in the subversion repository. The SVN trunk directory now contains the current development version, mainly with changes for the new PET chart mapping interface. It currently requires a PET from https://pet.opendfki.de/repos/pet/branches/cm instead of components-pet-binlib.tar.gz and components-pet-erg.tar.gz. The libraries tar for the HoG trunk version is available from http://heartofgold.dfki.de/pkg/hog-lib.tar.gz.

Release Notes for Version 1.5

We have uploaded a new version of Heart of Gold to http://heartofgold.dfki.de. Under the Download menu, you'll find new versions of the middleware and components packages (for RASP, PET including ERG, GG, JACY grammars, German shallow topoparser).

There is a now also a public Subversion repository for the middleware source code embedded in the trac environment at http://heartofgold.opendfki.de. Checking out the source code is as easy as

svn checkout https://heartofgold.opendfki.de/repos/trunk hog-trunk

for the latest development snapshot or

svn checkout https://heartofgold.opendfki.de/repos/tags/1.5 hog-1.5

for the latest release version (1.5), roughly as of August 2008 with three minor additions since then.

Read access is open to everyone. To get write access, you'll have to register via https://config.opendfki.de.

I would like to encourage you to make use of the trac-based ticketing system for code-related communication, feature and bugfix requests etc. Posting to this mailing list can then be reserved to general (i.e. less technical) discussion or questions, and announcements.

Here is an overview of the new features:

Core middleware

- Directory tree slightly restructured:
  - The conf/LL directories have been split into conf/LL/modules/ and conf/LL/sessions/ to separate module from session configurations (LL=two-letter ISO language code). Example: conf/en/xmlrpcsession.cfg has moved to conf/en/sessions/default.cfg
The files `conf/en/sessions/rasp2.cfg` and `conf/de/sessions/topo.cfg` contain separate session configurations for RASP and the German topoparser, their content can be copied to the respective `default.cfg` files, omitting those lines that are already present (jtok, chunkie).

The `xindice` directory has been removed from the distribution to get rid of dependencies and conflicts with Java XML libraries. This means that there is currently no direct support for an XML annotation database until we find a replacement. As Xindice is slow, we do not consider this a serious loss at the moment.

Server startup and shutdown handling now ant-less:
- `source bin/setup.sh` sets paths once at the beginning
- `then hog start` resp. `hog stop` for starting/stopping the XML-RPC server
- `analyzeGUI/analyzeAll` script (Python) also moved to `bin`, with modified parameter handling
- Default browser for `analyzeGUI` is now `firefox` (can be modified in the script variable `browser`)

Bugfix: now multiple Modules of the same class are supported within a session (e.g. for generic modules such as `SproutModule`, `SdlModule`)
- The Java middleware code now requires JDK 1.5 or 1.6.

Deep Parser/Grammars
- New PET version (cheap, flop, libraries) as of Spring 2008.
- New versions of compiled GG, ERG and JACY grammars as of Spring 2008.
- `PetModule`: now also supports the MRX output format (xml-encoded MRS). As there seem to be at least two different MRX formats (`Utool` is expecting a slightly different syntax), we provide a stylesheet `xsl/mrx/mrx2utool.xsl` that transforms the output of `cheap` for `Utool` (V3.1.1).
- There are now 5 different input formats supported by `cheap/PetModule` (some experimental; details in the PET Delph-in wiki):
  - raw text (option in `conf/LL/modules/pet.cfg`)
  - PET input chart (aka pixml or pic) (option in `conf/LL/modules/pet.cfg`)
  - SPPP (via transformation `xsl/pic/sppp2pic.xsl`)
  - SMAF (option in `conf/LL/modules/pet.cfg`; `xsl/smaf/pixml2smaf.xsl`)
  - FSC (developed in the Checkpoint project by Peter Adolphs). The new XSL stylesheet `xsl/fsc/pic2fsc.xsl` (experimental version) can be used to transform the PET input chart format into the FSC format.

LingPipe2Module
- made compatible with recent LingPipe versions (from 2.4 on)

Rasp2Module
- For RASP3 aka rel 2 (Web download version)
- The RASP-to-RMRS conversion is no longer part of the core RASP distribution, but is contained in a conversion server (part of LKB sources) that has to be started externally before a RASP session is started in Heart of Gold. We provide an executable for this server built using an LKB CVS version snapshot (Spring 2008).
- The new XSL stylesheet xsl/rmrs/removeanchors.xsl can be used to transform the RMRS output of the converter into the `old' RMRS format without <anchor> elements (mainly for getting results viewable with the rmrs2html stylesheet). Due to the underspecified documentation of the new format, this stylesheet can only be called experimental. Spurious multiple args of the same name and free args may appear. This is why this postprocessing transformation can be switched off in rasp.cfg.

LoParModule/Whiteboard Topoparser (for German only)

- This new module is a wrapper for LoPar (PCFG parser by Helmut Schmid, U. Stuttgart, to be downloaded separately) as well as the topoparser pipeline from the Whiteboard system. However, as there is currently no integration with the deep parser as was in Whiteboard, we consider it a standalone shallow module (session). Whether the computed brackets for the (old) GG version are output or not, can be configured in the module cfg file (default: off).

Installation (change: components packages no longer contain the hog directory prefix!)

A preliminary installation script can be downloaded via the URL

https://heartofgold.opendfki.de/repos/trunk/maint/install

Please check the script (comment out unnecessary parts) before running it. Manual installation:

1. Ubuntu/Debian user may install required packages using the following command:
   sudo apt-get install subversion python-tk sun-java6-jdk ant
   ipadic chasen firefox
2. download hog-*.tar.gz and components-*.tar.gz packages (if you use the svn version instead of hog-XXX-src.tar.gz, an additional ant generate_xsl is necessary after step 5)
3. unpack hog-*.tar.gz, then cd hog-XXX and unpack components-*.tar.gz packages there
4. adjust Python (and Java) paths in bin/setup.sh unless they are already in the standard PATH
5. source bin/setup.sh (once per login; extends the search path for commands)

Using the following command line, you can start the server, the browser and the python client with the default workflow for English (JTok, TnT, SProUT, PET, RMRSmerge):

   hog start && firefox && analyzeGUI -m localhost -p 8411 -l en
Shutdown HoG using the command `hog stop`.

EN, DE, JA configurations (including ChunkieRMRS EN+DE, topoparser DE) have been successfully tested on

- SuSE 9.3 bigsm 32 bit on an Intel multiprocessor server
- SuSE 10.2 32 bit on Intel Dual Core (Thinkpad)
- Ubuntu 8.04 on AMD Athlon X2 64 bit (Java@64 bit, rest@32 bit ;)),  
  RASP seemed not to work on this machine, but we did not further  
  investigate; maybe a problem with a missing 32bit library.

Finally, please note that PET, ERG, GG, JACY and the RASP RMRS converter server were built on CVS or subversion snapshots as of Spring 2008, i.e. they may out of sync with the latest versions available.

### Installation, File System Structure

Prerequisites

**Hardware:** i686 architecture (Intel or AMD, ~2 GHz recommended) with 1 GByte of RAM for the full system with 'GUI' clients for German or English (the other languages currently require less memory), if the memory usage of other programs (e.g. window manager) is economical, 512 MB RAM can be sufficient.

1. Linux
2. Sun Java Development Kit (JDK) 1.5 or 1.6
3. Python 2.5 or higher and (for the Python GUI client) tkinter, typically in packages named python and python-tk, sometimes called tkinter
4. for the Python GUI client: Firefox, Seamonkey, Mozilla > V1.3 or Netscape (note that remote controlling the browser works on X11 systems only, i.e. not on MS Windows!)
5. Apache ant (e.g. V 1.7; only required if you need to build HoG on your own, i.e. for compiling Java or generating XSLT code).

In Debian/Ubuntu, you may install the packages using the following command line:

```
sudo apt-get install subversion python-tk sun-java6-jdk ant ipadic chosen firefox
```

From now on in this documentation, paths are specified relative to the root directory for the Heart of Gold installation (path prefix hog/ is no longer included in the tar.gz archives). E.g., `conf/en/modules/pet.cfg` refers to the real directory
This convention is also used throughout the Heart of Gold itself, e.g. in configuration files etc. I.e., absolute paths should be avoided wherever possible (readlink -f and dirname can be used to compute absolute paths; this is e.g. done in the RASP start script).

Currently, the Heart of Gold middleware consists of the following separate packages.

- **hog-XXX-lib.tar.gz** (core binary and library files for Heart of Gold middleware, JTok tokenizer; archive contains root directory `hog-XXX/`).
- source files snapshot from svn: https://heartofgold.opendfki.de/repos/trunk
  Alternatively, a file named `hog-XXX-src.tar.gz` contains the source files of the middleware and modules below the root directory `hog-XXX/` (java/, python/, xsl/, default config files in conf/, ant build file `build.xml`).
- The archives `components-*.tar.gz` contain the `components/` subdirectories for specific components (without the `hog-XXX/` prefix!). The adapters classes (`'modules'`) and configuration files are part of `hog-XXX-src.tar.gz`. Some components not downloadable from DFKI have to be copied, compiled or installed to manually created subdirectories below `components/`, e.g., `lingpipe/`, `freeling/`, `rasp/`.

By checking out the subversion repository and unpacking these files, the following directory structure is established below the root directory:

```
bin/          -- scripts for managing start/build (SVN)
doc/          -- doc files from repository (SVN)
doc/javadoc/  -- for generated javadoc
components/   -- installation directory for components
conf/${lang}/modules/ -- configuration files for modules (SVN)
conf/${lang}/sessions/ -- configuration files for sessions (SVN)
java/         -- java source files (SVN)
lib/java/     -- jar archives
lib/classes/  -- for generated class files
log/          -- log file directory
maint/        -- maintenance scripts etc. (SVN)
php/          -- PHP source of online demo (SVN)
python/       -- python client scripts (SVN)
[xindice/     -- XML DB client installation directory]
xsl/          -- stylesheets for various transformations (SVN)
```

Directories and files like java sources, doc files, default configurations files that are marked with “(SVN)” come from the subversion repository, the other files and directories come from `hog-XXX-lib.tar.gz` and `components-*.tar.gz` (binary, jar and component-specific files).

**Sample Installation and Test**

(Default) Installation for components with English and German resources:
Download the following archives from [http://heartofgold.dfki.de/Download.html](http://heartofgold.dfki.de/Download.html) and extract them:

```bash
tar xzf hog-XXX-src.tar.gz             # all languages
#alternatively:
#svn checkout https://heartofgold.opendfki.de/repos/trunk hog-XXX
# or svn checkout https://heartofgold.opendfki.de/repos/tags/1.6 hog-1.6
tar xzf hog-XXX-lib.tar.gz             # all languages
cd hog-XXX
```

```bash
tar xzf components-pet-binlib.tar.gz   # all languages
tar xzf components-pet-erg.tar.gz     # EN HPSG grammar
tar xzf components-pet-german.tar.gz  # DE HPSG grammar
tar xzf components-sprout.tar.gz      # DE, EN (NE + chunkieRMRS), EL, JA (NE)
tar xzf components-chunkiermrs.tar.gz # DE, EN (SProUT source files only)
tar xzf components-chunkie.tar.gz     # DE, EN
tar xzf components-tnt.tar.gz         # DE, EN
```

```bash
cd hog
```

For English, you may additionally install LingPipe to components/lingpipe and RASP2 to components/rasp2 (see the corresponding module description below for more details). For Japanese, ChaSen is assumed to be installed centrally (part of most current Linux distributions). Likewise FreeLing, available from [http://www.lsi.upc.es/~nlp/freeling/](http://www.lsi.upc.es/~nlp/freeling/).

Before testing the installation, make sure that only the installed components are activated by editing `conf/${lang}/sessions/default.cfg`. These files contain the default configurations for the Python clients that are connected via XML-RPC. E.g., if you don't have RASP, comment the RASP line out in `conf/en/sessions/default.cfg`:

```bash
de.dfki.lt.hog.modules.JTokModule=conf/en/modules/jtok.cfg
de.dfki.lt.hog.modules.TnTModule=conf/en/modules/tnt.cfg
de.dfki.lt.hog.modules.ChunkieModule=conf/en/modules/chunkie.cfg
de.dfki.lt.hog.modules.SdlModule=conf/en/modules/chunkiermrs.cfg
de.dfki.lt.hog.modules.SproutModule=conf/en/modules/sprout.cfg
#de.dfki.lt.hog.modules.LingPipe2Module=conf/en/modules/lingpipe2.cfg
de.dfki.lt.hog.modules.Rasp2Module=conf/en/modules/rasp2.cfg
de.dfki.lt.hog.modules.SdlModule=conf/en/modules/rmrsmerge.cfg
```

For PET (currently English only), a dependency from the configured NE recognizer must be configured manually. I.e., if you use LingPipe for NE recognition, set

```bash
pet.inputannotation=TnTpiXML,LingPipepiXML
```

in `conf/en/modules/pet.cfg`

If you use SProUT (default), set

```bash
pet.inputannotation=TnTpiXML,SProUTpiXML
```

Otherwise, PET won't work. SProUT is required anyway if the chunkieRMRS module is activated.
You may then check the XML-RPC server port (default is 8411) in
conf/mocoman.cfg and the logging configuration in the same file which by
default points to conf/logging/html.cfg (for the first tests, setting log to
DEBUG is recommended; later INFO may be sufficient and more compact).
The actual logging setting can now be selected in the hog start script
bin/hog. Instead of HTML, other logging targets may be chosen, e.g.
console, textfile, chainsaw GUI, sockets, SNMP etc., for details see the log4j
documentation (http://logging.apache.org/log4j).

Once per login, you'll have to set paths etc. using the following command:
source bin/setup.sh

The following commands start the Heart of Gold XML-RPC server in
background, then the web browser (here: Firefox) for viewing results and
then the Python GUI client for English

hog start && firefox && analyzeGUI -m localhost -p 8411 -l en
python/en_test.txt

A German session (latin1 encoded input file) can be started with
analyzeGUI -m localhost -p 8411 -l de -e latin1 python/de_test.txt

There is a non-GUI Python client to process text files similar to analyzeGUI.
It is called analyzeAll and expects linefeed-separated sentences in the input
file.

analyzeAll -m localhost -p 8411 -l en python/en_test.txt

The output can be configured (XML and/or HTML files or no output
selectable for each of the available annotations; configuration at the
beginning of the Python code). The generated files are stored in the current
working directory and have the following file name format

aaaaaa-nnn.{xml,html}

where aaaaaa is the annotation name (aid) and nnn is the sentence number,
e.g. Sprout-002.xml.
For details on preprocessing and sentence splitting, see page 66.

The Ant Build File build.xml

Ant is a platform-independent make utility for (Java-based) applications, for
details see http://ant.apache.org. The following targets are defined in
./build.xml for the Heart of Gold (this is the output of the command ant
-projecthelp):

Buildfile: build.xml Main targets:

aa         shortcut to analyzeAll
all         compile, make jar and javadoc
analyzeAll        analyze all sentences in a given file
chainsaw          run chainsaw log viewer
chunkiermrs       compile chunkie rmrs sdl cascade -Dlang=XX [-Dstdlib.file=X]
clean             delete files generated by this script
compile           compile main project
generate_xsl      generate XSL stylesheet for SProUT2RMRS/PiXml translation
init               misc setup etc.
jar               generate project jar file
javadoc           generate javadoc
mocomanserver     run mocoman as an XML-RPC server
ms                shortcut to mocomanserver
rmrsmerge         compile rmrsmerge sdl cascade [-Dstdlib.file=X]
shutdown          shut down mocoman (via XML-RPC)

E.g., to compile the project, generate a jar file and javadoc, just type ant in the hog directory. The default target is all which performs the mentioned actions.

ant chainsaw &     runs the chainsaw log viewer GUI
Part 3: Implementing Modules and Applications

Writing an application

An example is given in the file
java/de/dfki/lt/hog/test/SproutPetApplication.java
An instance of MoCoMan is created which is configured using the properties file conf/test/sproutpetapp.cfg.
Two lines in this properties file configure the architecture instance to start 3 modules which control 3 local Java components installed in the components/ directory, namely Jtok, SproUT and PET.
de.dfki.lt.hog.modules.SproutModule=conf/en/modules/sprout.cfg
de.dfki.lt.hog.modules.JTokModule=conf/en/modules/jtok.cfg
The class names on the left indicate the local Java modules to create. The config property file names on the right specify startup configuration options for the modules and/or components.
The components are initialized through MoCoMan according to the configuration file. More precisely, a launcher creates the specified Module classes and registers them in the Registry. The init() methods of the Module classes are executed that start the real components.

Applications can also be connected with the MoCoMan through XML-RPC (java class XmlRpcApplication).
After the initializations, texts can be passed to the MoCoMan using the analyze() method.
Session, annotation collections and annotations ID are used to generate and refer to hierarchically organized XML annotations.

Analyze() takes input text, token range and depth of required analysis (integer value, where a higher number means deeper analysis). The MoCoMan then selects modules using the suitable() method, taking into account language and depth of analysis. The text is then passed as input to the process() method of the first selected component. Then, in a cascade, the output annotation of a preceding module is taken as the input annotation of a subsequent module’s process() method until all configured and (through depth and language) suitable components have returned their results.

If annotation database storage is enabled (through configuration of xml:db.usage in conf/mocoman.cfg), computed annotation can be stored persistently in the XML:DB annotation database. Hierarchical structure of sessions, annotation collections and annotation documents is automatically taken over from the MoCoMan.
A web browser can be used to inspect (in a very limited way) the contents of the Xindice database (see Xindice documentation below).
Writing a Module

NLP components can be integrated by implementing a Java class inheriting from the abstract class `de.dfki.lt.hog.Module`. New subclasses should be stored in the `de.dfki.lt.hog.modules` package. The following methods can/should be refined:

- `init(configProperties)`
- `process()`
- `shutdown()`

Configuration of a module can be stored in a config properties file which is automatically passed to the module upon initialization through the Launcher.

The minimal configuration comprises these properties (with sample values)

```
module.name=Sprout
module.depth=40
module.language=en
module.rootelement=SPROUTPUT
```

but can be extended e.g., with a location for a component-specific configuration file in the `.components` subdirectory such as

```
sprout.configfile=components/sprout/Project/en.cfg
```

If the XML annotation database is enabled, indexers for the XML format generated by the module should also be defined here, e.g.,

```
module.annotationindexers=(indexer1 MATCHINFO@start) (indexer2 DISJ)
```

As modules act as mediators between (existing) NLP components and the middleware, input and output conversion, e.g., from or to RMRS should be implemented here in 'adapter' methods. Common conversion methods can be shared by introducing an additional inheritance layer between the Module and their `real-world' subclasses for specific components.
Transformation Service

The TransformationService class supports XSL transformations plus a special handling for access to Heart of Gold-internal annotations for the XSLT (or XPath) document() function. The syntax in XPath expressions is:

```
document('hog://sid/acid/aid')
```

where sid is a sessionID, acid is a collection ID and aid is an annotation identifier such as Chunkie or JTok.
The TransformationService can be used from modules, from client applications, or from the SDL XsltModules (sub)classes.

To use the TransformationService in a Module subclass, create a field variable for a TransformationService object, say transformationService.

```
private TransformationService transformationService = null;
```

In the body of process(), add

```
if (this.transformationService==null) {
    this.transformationService = new TransformationService(getMoCoMan());
    myTransformer = this.transformationService.createTransformer(xslfile);
}
```

```
myTransformer.setParameter("sessionID", sid);
myTransformer.setParameter("collectionID", acid);
myTransformer.setParameter("annotationID", aid);
String result = transformationService.transform(myTransformer,xmlinput);
```

Access to multiple input annotations can be obtained by adding further parameters with unique names for annotationIDs (e.g., annotationID2 etc.).

There are also DOM-based variants of createTransformer() and transform(). The method getTransformerFactory() gives access to the underlying TransformerFactory. Cf. javadoc for TransformationService.

Location of the stylesheets
The XSLT stylesheets are stored in subdirectories of the xsl/ directory of the Heart of Gold distribution, ordered according to the target transformation format, e.g.

- html/: HTML visualization of RMRS structures
- latex/: LaTeX visualization of RMRS structures
- mapping/: named entity type to HPSG type mappings (no stylesheets; configuration input for automatic stylesheet generation for SProUT NER to PIC and RMRS transformation)
- pic/: transformation to PET input chart format
- preproc/: preprocessing of input texts in XML format, sentence splitting
- rmrs/: transformation of e.g. named entity recognizer output formats
(SProUT, LingPipe) to RMRS, partly automatically generated, cf. description of SProUTmodule below

- sdl/: stylesheets for XSLT transformations as part of SDL subarchitecture processing cascades, with project-wise subdirectories
- xml/: auxiliary stylesheets e.g. for XML prettyprinting
- mrx/: mrx conversion (mrx2utool.xsl)
- fsc/: pic to fsc conversion
- smaf/: pic (pet, sprout) to smaf conversion
- sleepy/: auxiliary stylesheets for SleepyModule
- topoparser/: transformation and merging stylesheets for LoParModule/Whiteboard Topoparser

Writing XML-RPC Modules and Adapters

In cases where a component should not share the JVM with MoCoMan, either because it is running on a different machine, or because it is implemented in a programming language other than Java, MoCoMan can communicate with components through XML-RPC. From the viewpoint of MoCoMan, XmlRpcModules behave like Modules. Instead of Module, XmlRpcModule is used as superclass for implementing a module. An example is given in the dummy test application java/de/dfki/lt/hog/test/SampleApplication.java

In this example, the remote TaggerAdaptor (XML-RPC) is started from the application main method (Alternatively, a special start() method could be implemented as part of the XmlRpcModule subclass). Class TaggerAdaptor java/de/dfki/lt/hog/modules/test/TaggerAdapter.java inherits from the abstract class Adaptor which in turn implements the XmlRpcServer interface, i.e., the TaggerAdaptor is the remote counterpart of a Module class called RemoteTagger java/de/dfki/lt/hog/modules/test/RemoteTagger.java that inherits from XmlRpcModule. In other words, RemoteTagger is a proxy for the remote TaggerAdapter. Global configurations for XML-RPC are stored in conf/mocoman.cfg.

MoCoMan configuration file conf/mocoman.cfg contains the configuration for the own XML-RPC server port and the XML:DB database:

```
# server port for XML-RPC communication
xmlrpc.server.port=8411

# a boolean value whether the XML database is used
xmldb.usage=true

# location of XML database
xmldb.location=xmldb:xindice://clavinova:8080/db
```
Utility class

The Utility class (de.dfki.lt.hog.util.Utility) contains static methods and constants for conversion (e.g. XML), formatting, logging standardized output etc.

Logging

Logging in MoCoMan is provided through log4j; for details see http://logging.apache.org/log4j. New local classes should use the same mechanism, cf. e.g. The source code of implemented Modules.

Logging output can be configured to be HTML file, XML file, or a remote log viewer like chainsaw (which is part of log4j) can be used via socket or XML files. The Chainsaw GUI client is part of log4j and can be started with ant chainsaw &.

For configuration details cf. Javadoc of log4j. Here is an example for the default HTML logging defined in conf/logging/html.cfg:

```
# logger configuration for Heart of Gold
# (log4j property file; see http://jakarta.apache.org/log4j/docs/)
#
log4j.rootLogger=INFO, html
log4j.appender.html=org.apache.log4j.FileAppender
log4j.appender.html.layout=org.apache.log4j.HTMLLayout
log4j.appender.html.file=log/hoglog.html
log4j.appender.html.append=false
log4j.logger.de.dfki.lt.sprout=WARN
log4j.logger.de.dfki.lt.hog.modules.PetModule=DEBUG
```

The logging configuration can be easily modified by switching to a different log4j configuration file via the property log.cfg in conf/mocoman.cfg (recommended), build.xml, or via command line:

ant ms -Dlog.cfg=file:/path/to/conf/logging/textfile.cfg &
**XML-RPC API for Application Clients**

The XML-RPC interface published by the MoCoManServer class makes the following functions available. In principle, all public methods of the following Java classes are accessible via XML-RPC:

```java
de.dfki.lt.hog.MoCoMan as mocoman
de.dfki.lt.hog.util.Utility as util
de.dfki.lt.hog.Metadata as metadata
```

For further details please cf. Javadoc (generate with `ant javadoc`). The order of these most-used methods is also the order in which a typical client application will access them. For a working application, e.g. See the python code in `bin/analyzeGUI`. In the following, sid stands for sessionID, acid for annotationCollectionID, aid for annotationID (annotation name).

**String** `sayHello()`

returns the String “Hello” in case the server is running

**String** `createSession(String cfg)`

creates a new session with the specified configuration. The cfg String specifies the configuration file name on the server, relative to the hog/directory. The return value is the session ID generated by the server.

**String** `createAnnotationCollection(String sid)`

creates a new annotation collection which is container for all annotation Heart of Gold will produce for an input text (or input sentence).

Annotation collections can be re-used to save space on the server. The return value is the session ID generated by the server.

**String** `createInitialAnnotation(String sid, String acid, String input, String lang, String clientName)`

Creates two initial annotations in an existing session sid and annotation collection acid from input text 'input':

1. the raw text from the String input with annotationID “rawtext”.
2. The XMLified raw text including metadata with the annotationID “xmltext”. Use this method for initially putting new text into MoCoMan.

String lang contains the ISO 639 language code (cf. Appendix) of the desired analysis language, clientName is an identifier indicating the requesting client or application. The annotation ID 'xmltext' of the generated annotation is returned.

**String** `analyse(String sid, String acid, String aid, int startSpan, int endSpan, int depth, String lang)`

Analyses the input annotation (specified via sid, acid and aid, where aid usually equals to 'xmltext') using the modules configured with `createSession()` up the specified depth and in the specified language lang. The parameters for a span are ignored at the moment. The result
of the analyses, i.e., the deepest annotation that could be computed, is returned as XML string. Other computed annotations can be retrieved with subsequent getAnnotation() calls.

Variant for text encoded binary (recommended as some XML-RPC implementations cannot handle 8bit/multibyte encodings correctly): **analyseBinary** with optional encoding parameter (String, last parameter, optional, default is "utf-8").

**String getAnnotation(String sid, String acid, String aid)**

Retrieves a computed annotation specified via sid, acid and aid, where aid is the name of the Module (value of property module.name in the Module's configuration file) that produced the annotation, or 'rawtext' or 'xmltext'.

Variant for text encoded binary (recommended as some XML-RPC implementations cannot handle 8bit/multibyte encodings correctly): **getAnnotationBinary** with optional encoding parameter (String, last parameter, optional, default is "utf-8").

**String getAnnotation7bit(String sid, String acid, String aid)**

Same as getAnnotation(), but converts any Unicode character > 127 in the markup to XML entities of the form "&#nnnnn;" where nnnnn is the Unicode code. With the binary transport of annotations (via XML-RPC), this should now be obsolete. See “Character Encoding Issues”.

**String xml2htmlTransformer(String sid, String acid, String aid, String stylesheetfilename, String flag)**

Transforms the annotation specified via sid, acid and aid, with an XSLT stylesheet located on the server (stylesheetfilename must be specified relative to the hog/ directory). Currently supported stylesheets are

1. "xsl/html/xml2html.xsl" with flag set to “no” which produces prettyfied XML as HTML
2. "xsl/html/rmrs2html.xsl" with flag set to “yes” which produces the graphical RMRS view as HTML.

The flag indicates that special XML elements are inserted before transformation containing information on the input text on the basis of the 'rawtext' input annotation. Setting this flag to “yes” only makes sense in conjunction with the rmrs2html.xsl stylesheet. The returned string contains the generated HTML code.

**String getStatus()**

Returns a String containing a text representing the status of mocoman (contained annotations in a tree view).

**boolean closeSession(String sid)**

Closes the specified session. Return value is true in case of success.

**boolean shutdown()**
Shuts down the whole MoCoMan server. Return value is true in case of success. Use carefully!
Character Encoding Issues

The Heart of Gold core is implemented in Java and hence XML markup which is internally represented as String (or DOM which is currently not fully supported for all modules) is encoded in 16 bit Unicode (UCS-2). However, raw text or XML markup exchange with external NLP component processes or applications clients often requires conversion to 8 bit character set encodings such as ISO-8859-x, UTF-8, EUC-JP (for Japanese), etc. In the case of Heart of Gold components, the conversion can easily be handled by the wrapping Java module. Java supports character set conversions by simply naming the source or target encoding as an additional parameter in the process or file stream reading or writing methods. If necessary, the input and output character set can be made part of the module configuration, as is the case e.g. in PetModule, ChasenModule, RaspModule, or TnTModule (configuration property <modulename>.inputencoding and <modulename>.outputencoding).

UTF-8 transport of annotations via XML-RPC has been problematic in earlier implementations (prior to June 2006), depending on the system environment (default encoding) and XML-RPC implementations. A workaround consisted of a routine that translated any character with Unicode > 127 in the markup to XML entities of the form “&#nnnn;,” where nnnn is the Unicode code (getAnnotation7bit() and analyse7bit()). This should now be obsolete as the new XML-RPC interface of HoG uses UTF-8 transport via the binary (base-64-encoded byte array) variant of XML-RPC parameter passing. This should be unique and compatible with all XML-RPC implementations (PHP not yet tried), while transporting UTF-8 via the String data type can only be guaranteed to work for US-ASCII (Unicode < 128), although many, but not all implementations assume UTF-8, or can at least be configured in that way.

The Binary versions are currently implemented for MoCoMan XML-RPC server and Python clients only, the Java clients still have to be checked and probably adapted in the same way (Java byte array parameters correspond to the Binary XML-RPC types).
Part 4: Description of Integrated Modules

Overview

The following modules with language resources are currently implemented:

<table>
<thead>
<tr>
<th>Name of component</th>
<th>Purpose</th>
<th>Depth e.g.</th>
<th>Language resources</th>
<th>Implemented in</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTok</td>
<td>tokenizer</td>
<td>10</td>
<td>en, de, it</td>
<td>Java</td>
</tr>
<tr>
<td>ChaSen</td>
<td>morph, WBR</td>
<td>10</td>
<td>ja</td>
<td>C</td>
</tr>
<tr>
<td>TnT</td>
<td>PoS tagger</td>
<td>20</td>
<td>en, de</td>
<td>C</td>
</tr>
<tr>
<td>TreeTagger</td>
<td>PoS tagger</td>
<td>20</td>
<td>en, de, es, it, ...</td>
<td>C</td>
</tr>
<tr>
<td>FreeLing</td>
<td>PoS, morph, NE</td>
<td>20</td>
<td>en, es, ca, gl, it</td>
<td>C++</td>
</tr>
<tr>
<td>Chunkie</td>
<td>chunker</td>
<td>30</td>
<td>en, de</td>
<td>C</td>
</tr>
<tr>
<td>ChunkieRmrs</td>
<td>chunks</td>
<td>35</td>
<td>en, de</td>
<td>XTDL, XSLT, SDL*</td>
</tr>
<tr>
<td>SProUT</td>
<td>morph, NER, IE,...</td>
<td>40</td>
<td>en, de, el, ja</td>
<td>Java</td>
</tr>
<tr>
<td>LingPipe</td>
<td>NER</td>
<td>40</td>
<td>en</td>
<td>Java</td>
</tr>
<tr>
<td>Corcy</td>
<td>coref. resolver</td>
<td>45</td>
<td>en</td>
<td>Python</td>
</tr>
<tr>
<td>LoPar/wbtopo</td>
<td>PCFG</td>
<td>50</td>
<td>de</td>
<td>C, XSLT</td>
</tr>
<tr>
<td>RASP</td>
<td>stat. parser</td>
<td>50</td>
<td>en</td>
<td>C, Lisp</td>
</tr>
<tr>
<td>SDL</td>
<td>subarchitectures</td>
<td></td>
<td></td>
<td>Java</td>
</tr>
<tr>
<td>Sleepy</td>
<td>shallow parser</td>
<td>50</td>
<td>de</td>
<td>OCaml</td>
</tr>
<tr>
<td>PET</td>
<td>deep parser</td>
<td>100</td>
<td>en, de, el, it, ja, no</td>
<td>C, C++, Lisp</td>
</tr>
<tr>
<td>RMRSmerge</td>
<td>merge RMRSes</td>
<td>110</td>
<td></td>
<td>XSLT, SDL/Java</td>
</tr>
</tbody>
</table>

*: ChunkieRmrs is not a proper module, but an instance of a SdlModule, based on the sub-architecture defined by an SDL description.

Sample configurations per language are in conf/${lang}/sessions/default.cfg. These files comprise all modules for which implementations in the specified language ${lang} exist. The corresponding module configuration files are examples, too! Modules can be (de)activated by simply commenting in/out (#) the corresponding configuration line. Note that deactivating a module may have an impact on depending modules as well (their configuration will have to be modified, too; e.g. PET depends on input of NE recognizers).

The Module classes described on the subsequent pages are defined in the package de.dfki.lt.hog.modules.

Their class name has to be mentioned fully in the session configuration file, e.g. conf/${lang}/sessions/default.cfg.
Module dependencies:
DE: PetModule requires JTokModule, TnTModule and SproutModule
  ChunkieRMRS requires JTokModule, TnTModule, ChunkieModule
  and SproutModule
  LoParModule (with Whiteboard Topoparser cascade) requires
  JTokModule, TnTModule (with tnt-topo configuration) and
  ChunkieModule
EN: PetModule requires JTokModule, TnTModule and SproutModule OR
  LingPipe2Module
  ChunkieRMRS requires JTokModule, TnTModule, ChunkieModule
  and SproutModule
  Rasp2Module requires JTokModule
JA: PetModule requires ChasenModule and SproutModule
EL: PetModule requires PicModule and SproutModule
JTokModule

- depth 10
- purpose: Tokenization, Sentence boundary recognition
- supported language resources: de, en, it (others can be defined in a JTok XML configuration file)
- runs JTok API
- init(): initialization with configured resources
- process(): Tokenization with jtok, XML output
- developers: JTok: Jörg Steffen (DFKI), JTokModule: Ulrich Schäfer, Özgür Demir
- path: components/jtok
- Installation: part of middleware-lib.tar.gz

Output DTD:

```xml
<!ELEMENT jtok (metadata p*)>
<!ELEMENT p (tu)+>
<!ELEMENT tu (Token)+>
<!ATTLIST tu id ID>
<!ELEMENT Token EMPTY>
<!ATTLIST Token string CDATA #REQUIRED
  type NMTOKEN #REQUIRED
  offset NMTOKEN #REQUIRED
  length NMTOKEN #REQUIRED>
```

Configuration file conf/en/modules/jtok.cfg:

```ini
# configuration file for JTok module
module.name=JTok
module.depth=10
module.language=en
# root element for XML output
module.rootelement=jtok
# ----- common modules settings end here -----# config file for JTok API
jtok.configfile=components/jtok/conf/jtok.cfg
```
ChasenModule

- depth 10
- purpose: morphology and word boundary recognition for Japanese
- supported language resources: ja
- init(): initialization with configured resources
- process(): start chasen script
- developers: AIST Nara (Matsumoto et al.), http://chasen.aist-nara.ac.jp
  ChasenModule: Ulrich Schäfer
- path: components/chasen
- Installation: central RPM installation assumed, alternatively download & compile sources from AIST, adapt configuration
  conf/ja/modules/chasen.cfg, conf/ja/modules/chasenrc-xml, components/sprout/Project/ja.cfg

Output DTD:
The output format is PiXML DTD (see page ).

It is defined for ChaSen in a configuration file that can be generated as follows (the central installation paths for ChaSen may differ from distribution to distribution).

cp /usr/share/chasen/dic/ipadic/chasenrc hog/conf/ja/modules/chasenrc-xml

set the output format in the copied file hog/conf/ja/modules/chasenrc-xml as follows (on a single line):

(OPTION_FORMAT "<w cstart="\%ps" cend="\%pe"><surface>
\%m</surface><pos tag="\%P-+%Tn-%Fn" prio="\%c"/></w>")

Configuration file conf/ja/modules/chasen.cfg:

module.name=ChaSen
module.depth=10
module.language=ja
# root element for XML output
module.rootelement=chasen
#
# ----- common modules settings end here -----
#
# command line options for ChaSen server
chasen.options=-r conf/ja/modules/chasenrc-xml
#
# path to server binary
chasen.binary=/usr/bin/chasen
#
# path to libraries for server
chasen.libs=/usr/lib/chasen
#
# input encoding
chasen.inputencoding=EUC-JP
SproutModule requires the ChaSen binary as well (and independently of the ChasenModule). The path to chasen in the configuration file for the Japanese NE grammar has to be adapted as follows.

In `hog/components/sprout/Project/ja.cfg`:

```
MorphologyApplicationPath=/usr/bin/chasen
```
### TnTModule

- depth 20
- purpose: Statistical PoS tagging
- supported language resources: de, en (others could be trained)
- developers: TnT: Thorsten Brants (Saarland U.), [http://www.coli.uni-saarland.de/~thorsten/tnt/](http://www.coli.uni-saarland.de/~thorsten/tnt/)  
  TnTModule: Özgür Demir
- path: components/tnt
- Installation: unpack components-tnt.tar.gz

TnT is a statistical PoS tagger. The module uses the JTok tokenization of the Heart of Gold as input and returns XML output of TnT analyses including probability values for PoS tags.

**Output DTD:**

```xml
<!ELEMENT tnt ( metadata tokens ) >
<!ELEMENT tokens ( w )* >
<!ELEMENT w ( p )* >
<!ATTLIST w str CDATA #REQUIRED
cstart NMTOKEN #REQUIRED
cend NMTOKEN #REQUIRED >
<!ELEMENT p EMPTY >
<!ATTLIST p pos NMTOKEN #REQUIRED
p   CDATA #REQUIRED >
```

Example:

```xml
<tnt>
  <metadata>
  <tokens>
    <w str="How" cstart="0" cend="2">
      <p pos="WRB" p="1.000000e+00"/>
    </w>
    <w str="cold" cstart="4" cend="7">
      <p pos="NN" p="6.513877e-01"/>
      <p pos="JJ" p="3.486123e-01"/>
    </w>
    <w str="should" cstart="9" cend="14">
      <p pos="MD" p="1.000000e+00"/>
    </w>
    <w str="a" cstart="16" cend="16">
      <p pos="DT" p="1.000000e+00"/>
    </w>
    <w str="refrigerator" cstart="18" cend="29">
      <p pos="NN" p="1.000000e+00"/>
    </w>
    <w str="be" cstart="31" cend="32">
      <p pos="VB" p="1.000000e+00"/>
    </w>
  </tokens>
</metadata>
```

Additionally, PET input chart XML format is generated according to the PiXML DTD (see PET section).

Configuration file conf/en/modules/tnt.cfg:

```plaintext
module.name=TnT
module.depth=20
module.language=en
# root element name for XML output
module.rootelement=tnt
# ----- common modules settings end here ----- 
# path to tnt startscript
 tnt.script=components/tnt/scripts/tnt.sh
# command line options for tnt
 tnt.options=-z20 -v0 models/wsj
# input encoding
 tnt.inputencoding=ISO-8859-1
# output encoding
 tnt.outputencoding=ISO-8859-1
# name of generated Pet input chart XML annotation
 tnt.piXMLoutputannotation=TnTpiXML
# 
# root element name of PET input chart XML annotation
#
 tnt.piXMLrootelement=pet-input-chart
#
```
FreeLingModule

- depth 20
- purpose: PoS tagger, NER and morphology
- supported language resources: es, ca, en, it, gl
- developers: FreeLing and LKB sppp wrapper: Lluis Padró.
  FreeLingModule, sppp2pic.xsl: Ulrich Schäfer,
- installation: centrally installed FreeLing RPM, LKBwrapper has to be
  compiled into components/freeling/bin (paths must be adapted in
  Makefile, cf. documentation in
  components/freeling/src/wrapper/README), then adapt paths in conf/es/
  modules/freeling.cfg and conf/es/modules/sppp.cfg

Output DTD:
The LkbWrapper (needs to be compiled) natively produces the LKB SPPP
format (documentation: http://wiki.delph-in.net/moin/LkbSppp).
PET input chart XML format is generated according to the PiXML DTD (see
PET section) using the stylesheet xsl/pic/sppp2pic.xsl

Configuration file conf/es/modules/freeling.cfg:

# configuration file for FreeLingModule (Spanish)
# 2006-06-17 ulrich.schaefer@dfki.de
#
module.name=FreeLing
module.depth=20
module.language=es
#
# root element name for XML output
module.rootelement=freeling
#
# ----- common modules settings end here -----  
#
# path to cheap binary
freeling.binary=components/freeling/LKBanalyzer
#
# additional library search path for cheap
#freeling.libs=components/freeling/lib
#
# command line options for cheap
freeling.options=-f conf/es/modules/sppp.cfg
#
# character set encoding for PET input
freeling.inputencoding=ISO-8859-1
#
# character set encoding for PET output
freeling.outputencoding=ISO-8859-1
#
# input annotation for FreeLing
freeling.inputannotation=xmltext
# stylesheet for PET XML input chart generation
freeling.picstylesheet=xsl/pic/sppp2pic.xsl
#

**PicModule**

- depth 10
- purpose: dummy module that generates Pet Input Chart format out of raw text
- supported language resources: - (currently only used for modern Greek)
- developers: Ulrich Schäfer
- path: Java module definition only

Output DTD:
PET input chart XML format is generated according to the PiXML DTD (see PET section).

Configuration file conf/el/modules/pic.cfg:

```
# configuration file for PicModule (modern Greek)
# 2004-10-13 ulrich.schaefer@dfki.de
#
module.name=Pic
module.depth=10
module.language=el
# root element name for XML output
module.rootelement=pet-input-chart
#
# ----- common modules settings end here ------
#
# name of raw text input annotation
pic.inputtextannotation=rawtext
#```
ChunkieModule

- depth 30
- purpose: Statistical chunking
- supported language resources: de, en (others could be trained)
- developers: Chunkie: Wojciech Skut (DFKI) and Thorsten Brants (Saarland U.), ChunkieModule: Özgür Demir
- path: components/chunkie
- Installation: unpack components-chunkie.tar.gz

ChunkieModule uses Jtok tokenization as input and returns XML output of Chunkie chunk analyses including the selected PoS tags from TnT.

Output DTD:

```xml
<!ELEMENT chunkie ( metadata chunks ) >

<!ELEMENT chunks ( s )* >

<!ELEMENT s ( w | chunk )* >
<!ATTLIST s id ID
cstart NMTOKEN #REQUIRED
cend   NMTOKEN #REQUIRED >

<!ELEMENT w ( #PCDATA ) >
<!ATTLIST w pos    NMTOKEN #REQUIRED
cstart NMTOKEN #REQUIRED
cend   NMTOKEN #REQUIRED >

<!ELEMENT chunk ( w )+ >
<!ATTLIST chunk cat    NMTOKEN #REQUIRED
cstart NMTOKEN #REQUIRED
cend   NMTOKEN #REQUIRED >
```

Example:

```xml
<chunkie>
<metadata>
  <id>
    <entry name="created" value="Sa, 12 Mrz 2005 16:46:00 +0100"/>
    <entry name="processingtime" value="00:00,803"/>
    <entry name="sessionid" value="session1"/>
    <entry name="acid" value="collection1"/>
    <entry name="component" value="Chunkie"/>
    <entry name="diagnosis" value="OK"/>
    <entry name="empty" value="false"/>
  </id>
  <conf>
    <entry name="chunkie.script" value="components/chunkie/scripts/chunkie.sh"/>
    <entry name="module.depth" value="30"/>
    <entry name="chunkie.options" value="data/wsj-pos data/wsj-chunk-convert"/>
  </conf>
</metadata>
```
<s id="S0" cstart="0" cend="33">
  <w pos="WRB" cstart="0" cend="2">How</w>
  <w pos="NN" cstart="4" cend="7">cold</w>
  <w pos="MD" cstart="9" cend="14">should</w>
  <chunk cat="NP" cstart="16" cend="29">
    <w pos="DT" cstart="16" cend="16">a</w>
    <w pos="NN" cstart="18" cend="29">refrigerator</w>
  </chunk>
  <w pos="VB" cstart="31" cend="32">be</w>
</s>

Configuration file conf/en/modules/chunkie.cfg:

module.name=Chunkie
module.depth=30
module.language=en

# root element name for XML output
module.rootelement=chunkie

# ----- common modules settings end here -----
# path to chunkie startscript
chunkie.script=components/chunkie/scripts/chunkie.sh
# command line options for rasp
chunkie.options=data/wsj-pos data/wsj-chunk-convert
# input encoding
chunkie.inputencoding=ISO-8859-1
# output encoding
chunkie.outputencoding=ISO-8859-1
LingPipe2Module

- depth 40
- purpose: statistical named entity recognition
- supported language resources: multilingual, depending on trained models
- runs LingPipe (original lingpipe archive copied to components/lingpipe)
- init(): initialization of variables
- process(): runs lingpipe NER and generates XML output
  LingPipeModule: Özgür Demir
- path: components/lingpipe
- Installation: unpack LingPipe distribution into components/lingpipe/

Configuration file conf/en/modules/lingpipe2.cfg:

```
# configuration file for LingPipe2 module

module.name=LingPipe2
module.depth=40
module.language=en

# root element name for XML output
module.rootelement=LINGPIPE

lingpipe2.modelsdir=components/lingpipe/lingpipe-3.5.0/demos/models
lingpipe2.modelfile=ne-en-news-muc6.AbstractCharLmRescoringChunker
lingpipe2.pixmlannotation=LingPipepiXML
lingpipe2.rawannotation=LingPipeRaw
lingpipe2.stylesheet4pic=xsl(pic/en_news_lingpipe2pixml.xsl
lingpipe2.stylesheet4rmrs=xsl/rmrs/en_news_lingpipe2rmrs.xsl
```

DTD

```xml
<!ELEMENT LINGPIPE ( metadata NE* ) >

<!ELEMENT NE EMPTY >
<!ATTLIST NE id NMTOKEN #REQUIRED
type NMTOKEN #REQUIRED
string CDATA #REQUIRED
cstart NMTOKEN #REQUIRED
cend NMTOKEN #REQUIRED >
```

Additionally, PET input chart XML format is generated according to the PiXML DTD (see PET section).
SproutModule

- depth 40
- purpose: rule-based named entity recognition, information extraction
- supported language resources: multilingual, depending on grammar and required morphology etc.
- runs SProUT runtime API
- init(): initialization of SProUT API with configured resources
- process(): SProUT analysis & transformation of XML output to RMRS DTD syntax
- path: components/sprout
- Installation: unpack components-sprout.tar.gz (or use ant sprout2hog if you have access to SProUT development environment)

SproutModule runs an instance of the SProUT interpreter (runtime API) with a configured grammar and other resources. The output of the interpreter is converted to the RMRS and PET Input Chart (or SMAF) formats using XSLT stylesheets that have been generated automatically at compile time from SProUT named entity grammar output type definitions using the ant target

ant generate_xsl [-Dproject=sproutprojectsuffix]

The optional -Dproject=sproutprojectsuffix parameter can be used to specify a suffix name for SProUT named entity grammars, e.g. ltworld or soccer. Currently, this only holds for the English and German project-specific grammars. The corresponding ant command in the SProUT (not HoG!!) source tree (from the SProUT subversion repository) is

ant sprout2hog [-Ddomain=sproutprojectsuffix]

The SProUT NE type to HPSG type mappings are defined in xsl/mappings/. The format for a mapping is

sprout_ne_type=hpsg_type

Types not mentioned in the mapping files are not mapped by the generated stylesheets. If a SProUT NE type is to be translated to RMRS, but not to the PET Input Chart, then the RHS of the mapping must be left empty:

sprout_ne_type=

The SProUT runtime subsystem for use in the Heart of Gold is generated by the SProUT source ant target 'sprout2hog', and currently comprises the runtime jar, 4 named entity grammars for EL, EN, DE, JA, and 8 ChunkieRMRS cascade grammars (in components/sprout). Cf. the SProUT manual and ant documentation for further information.
Multiple and cascaded SProUT grammars can be included via the new SdlModule (page 51).

Configuration file conf/en/modules/sprout.cfg:

```plaintext
module.name=Sprout
module.depth=40
module.language=en
# root element name for XML output
module.rootelement=SPROUTPUT
#
# ----- common modules settings end here -----#
#
# config file for SProUT runtime API
sprout.configfile=components/sprout/Project/de.cfg
#
# stylesheet for transformation of FS-XML to RMRS
sprout.stylesheet=xsl/rmrs/de_types-sprout2rmrs.xsl
#
# feature path to output structure
# if undefined, the root FS (including IN and OUT) is returned
# feature separator in the path can be . or | (as in TFS API)
sprout.outputpath=
#
# name of raw text input annotation for Sprout
sprout.inputtextannotation=rawtext
#
# name of feature structure output annotation
sprout.outputfsannotation=SproutFS
#
# ----- The following configurations are for PET input chart mode only -----#
# The subsequent settings are ignored if sprout.output4pic is not set
#
# name of output annotation for PET input chart (pic) format
# no pic annotation is generated if this value is omitted
sprout.output4pic=SProUTpiXML
#
# stylesheet for transformation of FS-XML (Sproutput) to PET input chart format
# (ignored if sprout.output4pic is not set)
sprout.stylesheet4pic=xsl/pic/de_types-sprout2pixml.xsl
#
# ----- End of configurations for PET input chart mode -----#
```
Sprout output ('SproUTput') DTD:

<!-- Sprout DTD Version 2004
    AUTHOR : uschaefer@dfki.de
    VERSION: 2.1
    DATE:    2004-01-21
-->

<!ELEMENT SPROUTPUT ( metadata DISJ* ) >

<!ELEMENT DISJ ( MATCHINFO )+ >
<!ATTLIST DISJ id ID >

<!ELEMENT MATCHINFO ( FS ) >
<!ATTLIST MATCHINFO id ID #IMPLIED
    rule NMTOKEN #IMPLIED
    cstart NMTOKEN #IMPLIED
    cend NMTOKEN #IMPLIED
    start NMTOKEN #IMPLIED
    end NMTOKEN #IMPLIED >

<!ELEMENT FS ( F )* >
<!ATTLIST FS type CDATA #REQUIRED
    coref NMTOKEN #IMPLIED >

<!ELEMENT F ( FS | SET ) >
<!ATTLIST F name NMTOKEN #REQUIRED >

<!ELEMENT SET ( FS | SET )* >
<!ATTLIST SET coref NMTOKEN #IMPLIED >

SProUT RMRS DTD:
see RMRS DTD on page 74. Structure:

<rmls-list>
  <metadata/>
  <rmls .../>
</rmls-list>

SProUT PiXML DTD:
Additionally, the PET input chart XML format is generated according to the PiXML DTD (see PET section).
Alternatively, the SMAF format can be generated (configuration in sprout.cfg):

Recently, sprout_morph configurations have been added. These are SProUT grammars passing the SProUT (mmorph-based) morphological analysis for each input token.
CorcyModule

- depth 45
- purpose: coreference resolver
- init(): configuration
- process(): runs a python program (via script)
- developed by: Özgür Demir
- path: components/corcy
- Installation: unpack components-corcy.tar.gz

Corcy is a coreference resolver implementation along the lines of [CardieWagstaff]. A copy of the article is to be found in components/corcy/doc. Corcy uses a heuristic clustering algorithm to determine coreference relationships (also uses TnT and Chunkie).

Sample output:

```xml
<corcy>
  <metadata .../>
  <clusters>
    <COREF cluster="0">John</COREF> loves <COREF cluster="2">Mary</COREF> and <COREF cluster="2">her</COREF> <COREF cluster="3">sister</COREF>.
  </clusters>
</corcy>
```

Output DTD:

```xml
<!ELEMENT corcy ( metadata clusters ) >
<!ELEMENT COREF #PCDATA >
<!ATTLIST COREF cluster NMTOKEN #REQUIRED >
```

Configuration file conf/en/modules/corcy.cfg:

```plaintext
module.name=Corcy
module.depth=45
module.language=en
# root element name for XML output
module.rootelement=corcy
# ----- common modules settings end here ----- #
# path to corcy start script
corcy.script=components/corcy/corcy.sh
# command line options for corcy
corcy.options=
# input encoding
corcy.inputencoding=ISO-8859-1
# output encoding
corcy.outputencoding=ISO-8859-1
```
Rasp2Module

- depth 50
- purpose: medium-depth analysis
- runs RASP in a local subprocess, RMRS conversion in separate Socket server
- init() starts RASP process via shell script
- process() sends input text to the RASP process
  http://www.informatics.susx.ac.uk/research/nlp/rasp/
  RASPModule: U. Schäfer
  Adaptations for RASP2: Torsten Marek, thanks to CJ Rupp for his help
- path: components/rasp2
- Installation (requires rasp_rel2.tgz from RASP download page AND components-rasp2.tar.gz from HoG download page)
  - tar xzf components-rasp2.tar.gz
  - cd components/rasp2
  - tar xzf .../rasp_rel2.tgz
  - mv components/rasp2/rasp_int_hog.sh components/rasp2/scripts/
    - in components/rasp2/scripts/rasp_parse.sh replace line
      RASP=/local/scratch/`whoami`/rasp3
    by
      R=$(readlink -f $0)
      RASP=${R%/*/*}
Currently, the RMRS converter server has to be started manually before starting a HoG session using Rasp2Module. The start script is

components/rasp2/rasp-rmrs-converter/start_server

You also may wish to adapt the host name in conf/en/modules/rasp2.cfg.

Configuration file conf/en/modules/rasp2.cfg:

```plaintext
module.name=RASP2
module.depth=50
module.language=en
# root element name for XML output
module.rootelement=rasp
# ----- common modules settings end here -----  
# path to rasp startscript
rasp2.script=components/rasp2/scripts/rasp_int_hog.sh
#  
# character set encoding for RASP input
rasp2.inputencoding=ISO-8859-1
#
# character set encoding for RASP output
rasp2.outputencoding=ISO-8859-1
#  
# stylesheet for postprocessing transformation
# (comment for no transformation)
```
rasp2.postprocstylesheet=xsl/rmrs/remove_rasp_anchors.xsl
#
# host and port of rmrs converter server
rasp2.converter_host=localhost
rasp2.converter_port=8891

RASP DTD (subset of RMRS DTD, cf. Appendix). Structure:

```xml
<rasp2>
  <metadata>
    <rmrs .../>
  </metadata>
</rasp2>
```

To build the RASP RMRS converter server (Lisp Image) from the LKB source tree in components/rasp2/rasp3-rmrs/src/, a Lisp (e.g. Allegro Common Lisp) is required.

```bash
cd components/rasp2/rasp3-rmrs/src/rmrs/rasp3/
built_standalone_image.sh DIRECTORY
```

where DIRECTORY is the target installation directory for the image.
It may be necessary to also adapt paths to the Lisp installation in the script. Adapt variable lkb_home in the start script start_rasp3_rmrs_server.sh to the directory containing the LKB src tree (src/...).

The script start_rasp3_rmrs_server.sh starts the RMRS converter server.

The postprocessing stylesheet xsl/rmrs/remove_rasp_anchors.xsl configurable in rasp2.cfg is currently experimental. It serves to remove the <anchor> elements to make the module output compatible with the (now old) RMRS DTD and the rmrs2html.xsl visualization stylesheet. However, this leads to spurious double ARGs and free_args. The postprocessing stylesheet can be turned off by commenting the configuration line:

```bash
#rasp2.postprocstylesheet=xsl/rmrs/remove_rasp_anchors.xsl
```

in rasp2.cfg.
LoParModule
(LoPar + German topoparser cascade from the Whiteboard project)
- depth 50
- purpose: German shallow topological parsing, combined with chunker, tagger output
- requires preparatory TntModule with configuration file conf/de/modules/tnttopo.cfg (depth 20)
- supported language resources: de
- paths: components/lopar, xsl/topoparser/

Configuration:

the following modules need to be activated (session configuration):
de.dfki.lt.hog.modules.JTokModule=conf/de/modules/jtok.cfg
de.dfki.lt.hog.modules.TnTModule=conf/de/modules/tnttopo.cfg
de.dfki.lt.hog.modules.ChunkieModule=conf/de/modules/chunkie.cfg
de.dfki.lt.hog.modules.LoParModule=conf/de/modules/lopar.cfg

# configuration file for LoPar

module.name=LoPar
module.depth=50
module.language=de

lopar.binary=components/lopar/bin/lopar
lopar.model=-in components/lopar/data/topo-tnt-nbest-relevant/topo-tnt
lopar.options=-viterbi -quiet -quote
lopar.xsldir=xsl/topoparser/
lopar.tmpDir=components/lopar/data/tmp

lopar.topotree.annotationname=TopoTree
lopar.topoflat.annotationname=TopoFlat
lopar.topochunks.annotationname=TopoChunks

# input encoding
lopar.inputencoding=ISO-8859-1
#
# output encoding
lopar.outputencoding=ISO-8859-1
#
# needed because of analysis inside the module uses chunkie
chunkie.moduleDepth=30
#
# stylesheet for postprocessing transformation
# (comment for no transformation)
#lopar.postprocstylesheet=xsl/topoparser/topo2brackets_chunks.xsl

LoPar is a PCFG parser. Its output
Output DTD (roughly; may be incomplete):

```xml
<!ELEMENT LoPar ( metadata, ROOT ) >
<!ELEMENT ROOT ( CL )* >
<!ELEMENT CL ( VF?, LK?, MF?, RK?, NF? ) >
<!ATTLIST CL fn NMTOKEN #REQUIRED >
<!ELEMENT VF ( CHUNK | W )* >
<!ATTLIST VF fn NMTOKEN #REQUIRED >
<!ELEMENT MF ( LK | RK | CHUNK | W )* >
<!ELEMENT NF ( LK | RK | CHUNK | W )* >
<!ELEMENT LK ( VAFIN | VVPP | KOUS | CHUNK | W )* >
<!ATTLIST LK fn NMTOKEN #REQUIRED >
<!ELEMENT RK ( VAFIN | VVPP | CHUNK | W )* >
<!ATTLIST RK fn NMTOKEN #REQUIRED >
<!ELEMENT KOUS ( W )* >
<!ELEMENT VAFIN ( W )* >
<!ELEMENT VVPP ( W )* >
<!ELEMENT CHUNK ( W )* >
<!ATTLIST CHUNK cat NMTOKEN #REQUIRED
   cstart NMTOKEN #REQUIRED
   cend NMTOKEN #REQUIRED >
<!ELEMENT W ( #PCDATA ) >
<!ATTLIST W id ID #REQUIRED
   pos NMTOKEN #REQUIRED
   cstart NMTOKEN #REQUIRED
   cend NMTOKEN #REQUIRED >
```

Installation:
- unpack HoG package components-lopar.tar.gz
- LoPar is not part of this archive, download separately from [ftp://ftp.ims.uni-stuttgart.de/pub/corpora/LoPar/lopar-3.0.linux.tar.gz](ftp://ftp.ims.uni-stuttgart.de/pub/corpora/LoPar/lopar-3.0.linux.tar.gz)
- copy lopar-3.0/bin/lopar to components/lopar/bin. The model and grammar files below data/ are part of components-lopar.tar.gz; they are different from those in the original distribution's data directory and have been produced in the Whiteboard project by Anette Frank.
- Directory structure below components/lopar/:

```
| -- bin/
|   `-- lopar
|      `-- data/
```
The Whiteboard topo parser XSLT cascade is described in [Topo2HPSG] and [WHAT].
Please note that currently the integration of the shallow topoparser cascade with the deep parser (German HPSG grammar) is not implemented in Heart of Gold. The most useful shallow topological annotations produced by the LoParModule is (presumably) TopoChunks, maybe also TopoTree and TopoFlat. The computed brackets in TopoBrackets, however, are not compatible with the recent versions of the German HPSG grammar, moreover, the PET interface has changed compared to the PET version employed in Whiteboard.
TreeTagger
(very experimental; integrated mainly for SleepyModule)
- depth 20
- purpose: Statistical PoS tagging
- supported language resources: de, en, es, it (others could be trained)
- developers: TreeTagger: Helmut Schmid, http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html
  TreeTaggerModule: Özgür Demir
- path: components/treetagger
- Installation: unpack components-treetagger.tar.gz

Configuration:

module.name=TreeTagger
module.depth=25
module.language=de
# root element name for XML output
module.rootelement=treetagger
# ----- common modules settings end here ----- #
# path to treetagger startscript
#treetagger.script=components/treetagger/cmd/tree-tagger-german
# # command line options for treetagger
#treetagger.options=-token -lemma -sgml -pt-with-lemma
# # input encoding
#treetagger.inputencoding=ISO-8859-1
# # output encoding
#treetagger.outputencoding=ISO-8859-1
# # annotation name for synchronization with tokenization
treetagger.tokenization=JTok
#

TreeTagger is a statistical PoS tagger. The module uses the Jtok
tokenization of the Heart of Gold as input and returns XML output of
TreeTagger analyses.

Output DTD:

<!ELEMENT treetagger ( metadata tokens ) >

<!ELEMENT tokens ( t )* >

<!ELEMENT t ( PCDATA )* >
<!ATTLIST t pos NMTOKEN #REQUIRED
  lemma CDATA #REQUIRED
  cstart NMTOKEN #REQUIRED
  cend NMTOKEN #REQUIRED >
Installation
Download TreeTagger, the scripts archive and necessary parameter files. Unpack to hog/components/treetagger/ (subdirs are contained in archives), the directory tree structure should be as follows.

```
|-- FILES
|-- LICENSE
|-- README
|-- README.script
|-- bin
  |-- separate-punctuation
  |-- train-tree-tagger
  `-- tree-tagger
|-- cmd
  |-- filter-chunker-output.perl
  |-- filter-german-tags
  |-- lookup.perl
  |-- mwltaglookup.perl
  |-- tagger-chunker-english
  |-- tagger-chunker-german
  |-- tree-tagger-english
  |-- tree-tagger-french
  |-- tree-tagger-german
  |-- tree-tagger-italian
  `-- tree-tagger-spanish
|-- doc
  |-- nemlap94.ps
  `-- sigdat95.ps
|-- lib
  |-- english-abbreviations
  |-- english-lexicon.txt
  |-- english.par
  |-- german-abbreviations
  |-- german-lexicon.txt
  |-- german.par
  |-- italian.par
  |-- spanish-abbreviations
  `-- spanish-mwls
```

Add the following two lines and edit the variables BIN/CMD/LIB in the treetagger scripts (e.g. cmd/tree-tagger-german) as follows:

```
TTHOME=`dirname \`dirname \``readlink -f $0`\```
cd $TTHOME

BIN=./bin
CMD=./cmd
LIB=./lib
```
SleepyModule

- depth: 50
- purpose: probabilistic shallow parser for German
- runs sleepy (implemented in OCaml)
- init(): start Sleepy binary, load model
- path: components/sleepy
- Installation: Unpack components-sleepy.tar.gz.

SleepyModule parses German sentences by using a probabilistic model (configurable). Currently, Sleepy uses JTok tokenization

```plaintext
module.name=Sleepy
module.depth=50
module.language=de
# root element name for XML output
module.rootelement=sleepy
#
# ----- common modules settings end here -----
#
# path to sleepy startscript
sleepy.script=components/sleepy/scripts/sleepy.sh
#
# command line options for chunkie
#sleepy.options=--model-file ./data/tiger.model --multipass-smooth2 --witten-bell --max-sentence 40 --beam 1000 --pos-beam 3000 --use-parse-cache
#sleepy.options=--max-sentence 40 --turbo --gen-lossy-binarization --beam 1000 --model-file ./data/tiger.nosmooth.model
sleepy.options=--max-sentence 40 --gen-lossy-binarization --beam 1000 --model-file ./data/tiger.nosmooth.model
#
# input encoding
sleepy.inputencoding=ISO-8859-1
#
# output encoding
sleepy.outputencoding=ISO-8859-1
#
# annotation name for synchronization with tokenization
sleepy.tokenization=JTok
#
# stylesheet for postprocessing
sleepy.postprocstylesheet=xsl/sleepy/insertLemma.xsl
#
# name of input annotation to include in postprocessing, e.g.
# TreeTagger
sleepy.postprocinputannotation=TreeTagger
#```

Output DTD:

```xml
<!ELEMENT sleepy ( metadata parse ) >
<!ELEMENT parse ( node )* >
<!ELEMENT node ( #PCDATA | ( node )* ) >
<!ATTLIST node
cat CDATA #REQUIRED
lb CDATA #IMPLIED
lemma CDATA #IMPLIED
id NMTOKEN #REQUIRED
cstart NMTOKEN #REQUIRED
cend NMTOKEN #REQUIRED >
```

Installation:

Here is the directory structure below hog/components/sleepy/:

```
|-- bin
 |  `-- sleepy
|-- data
 |  |-- negra.model
 |  `-- tiger.model
 |     `-- tiger.nosmooth.model
|-- scripts
   `-- sleepy.sh
```
**SdlModule**

- depth: depending on instance
- purpose: general module, sub-architectures with access to other Heart of Gold annotations
- runs a class compiled from an SDL description
- init(): initialization of a compiled SDL description with configured resources
- process(): Sprout analysis & transformation of XML output to RMRS DTD syntax
  ChunkieRMRS cascade: Anette Frank and Kathrin Spreyer
  RMRSmerge cascade: Anette Frank
- path: xsl/sdl (part of middleware-src.tar.gz)
- Installation: Unpack middleware-src.tar.gz. The file components-chunkiermrs.tar.gz contains the SProUT grammar sources (not necessary for runtime).

SdlModule provides a generic interface (as a kind of `meta module`) to sub-architectures defined in SDL with additional access to other annotations/components via the Heart of Gold TransformationService. The Chunkie RMRS cascade is described in [ChunkieRMRS]. For details on SDL, cf. Krieger's DFKI report RR-03-01 [SDL].

Implemented submodules for SDL so far:

- package `de.dfki.lt.sdl.sprout`: SProUT SDL submodules
- package `de.dfki.lt.sdl.xslt`: XSLT SDL submodules (using TransformationService of MoCoMan)

Note that modules need not be integrated as SDL submodules. It is possible to access any (existing) annotation in HoG through the hog:// URI in XPath document() functions from within XSLT stylesheet. The following XsltModule subclasses support this features (indicated by the Encapsulated suffix in class names):

```java
de.dfki.lt.sdl.xslt.XsltModulesDomDomEncapsulated
de.dfki.lt.sdl.xslt.XsltModulesDomStringEncapsulated
de.dfki.lt.sdl.xslt.XsltModulesStringDomEncapsulated
de.dfki.lt.sdl.xslt.XsltModulesStringStringEncapsulated
```

Please note that the parameter to be used in the stylesheet is called uri, while the annotation name to be passed as parameter to XsltModulesXXXEncapsulated has to be named aid (for annotation identifier). The uri parameter value is assembled by the XsltModulesXXXEncapsulated from the current session and annotation collection ID plus the aid parameter.
Auxiliary classes (mocoman package):
AnnotationEncapsulator encapsulates Heart of Gold-specific parameters like sessionID, annotationCollectionID, language, MoCoMan object.

**ChunkieRMRS**

An instance of an SDL-based sub-architecture is the Chunkie RMRS conversion cascade consisting of 4 SProUT interpreter instances and 4 XSLT transformations (xsl/sdl/chunkiermrs and SProUT sources in SProUT source repository; copy available as archive: components-chunkiermrs.tar.gz). 

ant chunkiermrs -Dlang={en,de} compiles an SDL description and the generated java code for Heart of Gold.

Configuration file conf/en/modules/chunkiermrs.cfg (the SDL definition is in xsl/sdl/chunkiermrs/en/chunkiermrs.sdl):

```
module.name=ChunkieRmrs
module.depth=35
module.language=en
module.rootelement=chunkiermrs
# ----- common modules settings end here -----
# name of input annotation (raw text for first cascade/SProUT)
sdl.inputannotation=rawtext
# class name of compiled SDL definition
# (same as class name at beginning of .sdl file)
# can be compiled using 'ant chunkiermrs'
sdl.classname=de.dfki.lt.hog.sdlgen.chunkiermrs_en
```
Output DTD is a subset of RMRS DTD, cf. Appendix.

**RmrsMerge**

Another instance of an SDL-based sub-architecture is the RMRSmerge cascade consisting of 5 XSL transformations (in xsl/sdl/rmrsmerge). It merges a configurable (in rmrsmerge.sdl) secondary RMRS annotation (typically from a named entity recognition component such as SProUT or LingPipe) into a configurable (in rmrsmerge.cfg) main RMRS annotation (typically from RASP or PET) by using character span information and adjusting RMRS variables.

`ant rmrsmerge` compiles an SDL description and the generated java code for Heart of Gold.

Configuration file conf/en/modules/rmrsmerge.cfg (merge PET with SProUT):

```xml
module.name=RmrsMerge
module.depth=110
module.language=en
# root element name for XML output
module.rootelement=merged-rmrs
# ----- common modules settings end here -----  
# name of input annotation (PET or RASP)
```
The SDL definition is in xsl/sdl/rmrmerge/rmrmerge.sdl:

de.dfki.lt.hog.sdlgen.rmrmerge = ( rmrs_ep_rargs2rels + adjust_nespans + merge_ne_to_petrasp + rmrs_rels2ep_rargs + reorder_rmrrs_dtrs )

rmrs_ep_rargs2rels = de.dfki.lt.sdl.xslt.XsltModulesStringDomEncapsulated
("xsl/sdl/rmrmerge/rmrs_ep_rargs2rels.xsl", "SDLx_rargs2rels")

adjust_nespans = de.dfki.lt.sdl.xslt.XsltModulesDomDomEncapsulated
("xsl/sdl/rmrmerge/adjust_nespans.xsl", "SDLx_adjustnespans", "aid", "Sprout")

merge_ne_to_petrasp = de.dfki.lt.sdl.xslt.XsltModulesDomDomEncapsulated
("xsl/sdl/rmrmerge/merge-ne-to-rasp.xsl", "SDLx_netorasp", "aid", "Sprout")

rmrs_rels2ep_rargs = de.dfki.lt.sdl.xslt.XsltModulesDomDomEncapsulated
("xsl/sdl/rmrmerge/rmrs_rels2ep_rargs.xsl", "SDLx_rels2ep_rargs")

reorder_rmrrs_dtrs = de.dfki.lt.sdl.xslt.XsltModulesDomStringEncapsulated
("xsl/sdl/rmrmerge/reorderrmrsdtrs.xsl", "SDLx_reorderdtrs", "aid", "xmltext")

Output DTD is a subset of RMRS DTD, cf. Appendix.
PetModule

- depth 100
- purpose: HPSG parsing
- runs PET HPSG parser (cheap) in a local subprocess
- init() starts cheap with resources configured in pet.cfg in a subprocess, using stdin/stderr for communication
- process() reads input sentence and returns RMRS(es) computed by PET.
- path: components/pet
- Installation: unpack components-pet-binlib.tar.gz and grammar packages

Configuration file conf/en/modules/pet.cfg:

```conf
# configuration file for PET module
# 2003-11-25 ulrich.schaefer@dfki.de
#
module.name=PET
module.depth=100
module.language=en
#
# root element name for XML output
module.rootelement=pet
#
# ----- common modules settings end here -----#
#
# path to cheap binary
pet.binary=components/pet/bin/cheap
#
# additional library search path for cheap
pet.libs=components/pet/lib
#
# working directory (where the grammar is)
pet.grammardir=components/pet/erg
#
# prefix for grammar file
pet.grammarprefix=english
#
# command line options for cheap
#pet.options=-tok=smaf -mrs=rmrx -partial -nsolutions=3 -results=3 -default-les -limit=70000
pet.options=-tok=pic_counts -mrs=rmrx -results=3 -default-les
   -limit=70000
#
# character set encoding for PET input
pet.inputencoding=UTF-8
#
# character set encoding for PET output
pet.outputencoding=UTF-8
```
PET input
There are now 5 different input formats supported by cheap/PetModule (some experimental; details in the PET Delph-in wiki):

- raw text (option in conf/LL/modules/pet.cfg)
- PET input chart (aka pixml or pic) (option in conf/LL/modules/pet.cfg)
- SPPP (via transformation xsl/pic/sppp2pic.xsl)
- SMAF (option in conf/LL/modules/pet.cfg; xsl/smaf/pixml2smaf.xsl)
- FSC (developed in the Checkpoint project by Peter Adolphs). The XSL stylesheet xsl/fsc/pic2fsc.xsl (experimental version) can be used to transform the PET input chart format into the FSC format.

PET input chart DTD
Next page; for details see the PET input chart documentation by Bernd Kiefer.
<!-- PET input chart DTD -->
<!-- {Bernd.Kiefer,Ulrich.Schaefer}@dfki.de -->
<!-- Version 2004-12-21 -->

<!DOCTYPE pet-input-chart [
  <!ELEMENT pet-input-chart ( w | ne )* >
  <!-- base input token -->
  <!ELEMENT w ( surface, path*, pos*, typeinfo* ) >
  <!ATTLIST w         id ID      #REQUIRED
                  cstart NMTOKEN #REQUIRED
                  cend NMTOKEN #REQUIRED
                  prio CDATA   #IMPLIED
                  constant (yes | no) "no" >
  <!-- constant "yes" means: do not analyse, i.e., if the tag contains
      no typeinfo, no lexical item will be build by the token -->
  <!-- The surface string -->
  <!ELEMENT surface ( #PCDATA ) >
  <!-- numbers that encode valid paths through the input graph (optional) -->
  <!ELEMENT path EMPTY >
  <!ATTLIST path     num NMTOKEN #REQUIRED >
  <!-- every typeinfo generates a lexical token -->
  <!ELEMENT typeinfo ( stem, infl*, fsmod* ) >
  <!ATTLIST typeinfo   id ID     #REQUIRED
                  prio CDATA  #IMPLIED
                  baseform (yes | no) "yes" >
  <!-- Baseform yes: lexical base form; no: type name -->
  <!-- lexical base form or type name -->
  <!ELEMENT stem ( #PCDATA ) >
  <!-- type name of an inflection rule-->
  <!ELEMENT infl  EMPTY >
  <!ATTLIST infl    name CDATA   #REQUIRED >
  <!-- put type value under path into the lexical feature structure -->
  <!ELEMENT fsmod  EMPTY >
  <!ATTLIST fsmod   path CDATA   #REQUIRED
                    value CDATA   #REQUIRED >
  <!-- part-of-speech tags with priorities -->
  <!ELEMENT pos  EMPTY >
  <!ATTLIST pos      tag CDATA   #REQUIRED
                    prio CDATA   #IMPLIED >
  <!-- structured input items, mostly to encode named entities -->
  <!ELEMENT ne  ( ref+, pos*, typeinfo+ ) >
  <!ATTLIST ne        id ID      #REQUIRED
                    prio CDATA   #IMPLIED >
  <!-- reference to a base token -->
  <!ELEMENT ref  EMPTY >
  <!ATTLIST ref      dtr IDREF   #REQUIRED >
]>
PET output DTD: RMRS DTD cf. Appendix (page 74)

<pet readings="n">
  <metadata>
    <rmrs .../>
  </metadata>
</pet>

Following is a sample PET input chart for "Dr. Mike Smith went to New York" (this is one possibility to encode it; a second one would be to use the <ne> tags referring to the <w> tokens for multi-token expressions like named entities, phrases etc.

<?xml version="1.0"?
<pet-input-chart>
  <w id="TNT0" cstart="0" cend="2">
    <surface>Dr.</surface>
    <pos tag="NNP" prio="1.000000e+00"/>
  </w>
  <w id="TNT1" cstart="4" cend="7">
    <surface>Mike</surface>
    <pos tag="NNP" prio="1.000000e+00"/>
  </w>
  <w id="TNT2" cstart="9" cend="13">
    <surface>Smith</surface>
    <pos tag="NNP" prio="1.000000e+00"/>
  </w>
  <w id="TNT3" cstart="15" cend="18">
    <surface>went</surface>
    <pos tag="VBD" prio="1.000000e+00"/>
  </w>
  <w id="TNT4" cstart="20" cend="21">
    <surface>to</surface>
    <pos tag="TO" prio="1.000000e+00"/>
  </w>
  <w id="TNT5" cstart="23" cend="25">
    <surface>New</surface>
    <pos tag="NNP" prio="1.000000e+00"/>
  </w>
  <w id="TNT6" cstart="27" cend="30">
    <surface>York</surface>
    <pos tag="NNP" prio="1.000000e+00"/>
  </w>
  <w id="SPR2.1" cstart="0" cend="13" constant="yes" prio="0.5">
    <surface>Dr. Mike Smith</surface>
    <typeinfo id="TIN2.1" baseform="no">
      <stem>$generic_name</stem>
    </typeinfo>
  </w>
  <w id="SPR3.1" cstart="23" cend="30" constant="yes" prio="0.5">
    <surface>New York</surface>
    <typeinfo id="TIN3.1" baseform="no">
      <stem>$generic_name</stem>
    </typeinfo>
  </w>
</pet-input-chart>
Sample Hybrid Workflows for English, German, Japanese

**de**

- Input text
- JTok
- SProUT
- Chunkie
- TnT
- PET XML
- RMRS
- ChunkieRMRS
- SDL cascade
- RMRSmerge
- Shallow RMRS
- Fallback RMRS

**en**

- Input text
- JTok
- SProUT
- Chunkie
- TnT
- PET XML
- RMRS
- PET
- RMRS
- Fallback RMRS
- Merged RMRS

**ja**

- Input text
- ChaSen
- SProUT
- PET XML
- RMRS
- PET
- RMRS
- Fallback RMRS
- Merged RMRS
- Shallow RMRS
Part 5: Python Clients and Pre-Processing

Python XML-RPC client

There are several Python scripts demonstrating the use of the middleware from an XML-RPC client.

A simple, low-level non-GUI client that nicely shows how applications communicate with the MoCoMan server is

`./python/sampleRpcClient.py`

The user can interactively create sessions, annotation collections, annotations, analyse input texts and retrieve computed annotations. Caveat: The purpose of the script is to give examples how to analyse texts and retrieve result programmatically, not to provide a user-friendly UI (for this, use analyzeGUI described below).

The configuration of modules is passed from the Python client to the server, i.e., for each session, a different configuration could be specified (in principle; in the sample script, the config file is fixed in a variable at the beginning).

By default, the same configuration as for the Sprout-Pet-JTok sample Java application is used (conf/test/sproutpetapp.cfg).

A typical session is as follows (user commands in red):

```
hog start  # starts MoCoMan XmlRpc Server (port configuration in ./conf/mocoman.cfg)
./python/sampleRpcClient.py
Message from MoCoManServer on localhost: Hello
Connected to server localhost on port 8411
Command (h for help): h
  cs  create a new session
  cc  create a new collection
  ca  create a new annotation
  a   analyze an annotation
  g   get a computed annotation
  p   print final result of analysis
  s   status report
  x   shutdown mocoman
  q   quit
Command (h for help): cs

New session created: id = 'session1'
Command (h for help): cc
Session ID [session1]:
  New annotation collection added to session1: 'collection1'
Command (h for help): ca
Session ID [session1]:
Collection ID [collection1]:
Input (text): The manager in New York who evaluated his programmers is more competent than the one who had consultants evaluate them
```
New annotation 'rawtext' created and added to 'session1.collection1'
New annotation 'xmltext' generated for xmlified text with metadata.
Command (h for help): a
  Session ID [session1]:
  Collection ID [collection1]:
  Annotation ID [xmltext]:
  Result ready. 'p' to print.
Command (h for help): g
  Session ID [session1]:
  Collection ID [collection1]:
  Annotation ID [xmltext]: JTok
</jtok><metadata created="Di, 13 Jan 2004 10:20:42 +0100"
  processingtime="00:00,01" sessionid="session1" acid="collection1"
  component="JTok" diagnosis="OK"/>
<p>
  <tu id="0">
    <Token string="The" type="TOK" offset="0" length="3" />
    <Token string="manager" type="TOK" offset="4" length="7" />
    <Token string="in" type="TOK" offset="12" length="2" />
    <Token string="New" type="TOK" offset="15" length="3" />
    <Token string="York" type="TOK" offset="19" length="4" />
    <Token string="who" type="TOK" offset="24" length="3" />
    <Token string="evaluated" type="TOK" offset="28" length="9" />
    <Token string="his" type="TOK" offset="38" length="3" />
    <Token string="programmers" type="TOK" offset="42" length="11" />
    <Token string="is" type="TOK" offset="54" length="2" />
    <Token string="more" type="TOK" offset="57" length="4" />
    <Token string="competent" type="TOK" offset="62" length="9" />
    <Token string="than" type="TOK" offset="72" length="4" />
    <Token string="the" type="TOK" offset="77" length="3" />
    <Token string="one" type="TOK" offset="81" length="3" />
    <Token string="who" type="TOK" offset="85" length="3" />
    <Token string="had" type="TOK" offset="89" length="3" />
    <Token string="consultants" type="TOK" offset="93" length="11" />
    <Token string="evaluate" type="TOK" offset="105" length="8" />
    <Token string="them" type="TOK" offset="114" length="4" />
  </tu>
</jtok>
Command (h for help): p
</mrms readings="3"><metadata created="Di, 13 Jan 2004 10:20:42 +0100"
  processingtime="00:00,142" sessionid="session1" acid="collection1"
  component="PET" diagnosis="OK"/>
  <rmrs reading="0" cfrom='-' cto='-'><label
  vid='1'/><ep cfrom='-' cto='--'><ame>RSTR</ame><rargname> ... </ep>
  </rmrs></mrms>
Command (h for help): s
Uptime: 33 minutes 52 seconds 87 milliseconds
logger.host: localhost
logger.path: log
xmldb.usage: false
xmldb.location: xmldb:xindice://clavinova:8080/db
logger.port: 4445
logger.level: DEBUG
logger.file: hoglog.html
logger.format: html
xmlrpc.server.port: 8411
|--session1
  |--collection1
  |   |--result
Shutting down the server (via XML-RPC)

Use the script
hog stop [<host> <port>]

or
ant shutdown [-Dxmlrpc.server.host=localhost] [-Dxmlrpc.server.port=8411]
(defaults values in conf/mocoman.cfg)

GUI Client

A python/TK GUI for choosing sentences from an input file, selecting depth.
- hog start starts MoCoMan XmlRpc Server (port configuration in
  conf/mocoman.cfg)
- configure path to mozilla/netscape/firefox/seamonkey at the beginning of
  the python script (e.g. browser = "seamonkey -remote" if you use
  seamonkey).
- configure the requested annotation names
- start the mozilla, netscape, firefox or seamonkey browser (Unix only;
  Windows Mozilla does not support remote control)
- analyzeGUI [-m hostname] [-p port] [-l lang] [-e encoding] textfile

<lang> is a two-character (ISO 639) language identifier that is used as
prefix for a language-specific session configuration on the server.
en (or nothing) selects conf/en/sessions/default.cfg
de selects conf/de/sessions/default.cfg
no selects conf/no/sessions/default.cfg
etc.
<encoding> is the name of the <textfile>'s character encoding.
analyzeGUI:

Sample output of analyzeGUI:

```
TEXT: Show me a picture of a Nokia phone
TOP: h1

RELs:
- `picture_p` with `h1`
- `of_p` with `h10001`
- `h3 q-q` with `h11`

HCONS:
- `h5 q-eq h11, h9 q-eq h10, h17 q-eq h14, h19 q-eq h21, h27 q-eq h29, h33 q-eq h34`

ING:
- `h21 ing h10001, h28 ing h10002`
```

```
TEXT: Show me a picture of a Nokia phone
TOP: h36

RELs:
- `picture_p` with `h10`
- `of_p` with `h12`
- `h3 q-q` with `h16`

HCONS:
- `h36 q-eq h35, h43 q-eq h41, h6 q-eq h5, h18 q-eq h19`

ING:
- {}
```

```
TEXT: Nokia
TOP: h22

RELs:
- `npp-product` with `h22`

HCONS:
- {}

ING:
- {}
```
**SProUT Feature Structure Viewer (SProUTput applet)**

There is an applet showing SProUT results in graphical AVM notation. It is automatically activated by analyzeGUI when XML files conforming to the SProUTput DTD are to be visualized.

Installation of the JRE Java applet plugin for the web browser (Mozilla, Firefox, Netscape or Seamonkey) if not already done:

```bash
cd ~/.mozilla/plugins
ln -s $JAVA_HOME/jre/plugin/i386/ns610-gcc32/libjavaplugin_oci.so
```

Example (feature structure visualization of SProUTput DTD):

```xml
<applet archive="fsapplet.jar" code="de.dfki.lt.rendering.SproutputApplet"
       width="900" height="700">
  <param name="xmlfile" value="SProUTputFile.xml"> <!-- SProUTput DTD -->
  <param name="encoding" value="utf-8">
</applet>
```

The applet is inserted into an HTML document as follows.
Feature structure and RMRS visualization stylesheets (HTML, \LaTeX)

In the xsl/ subdirectory, there are subdirectories containing XSLT stylesheets for HTML and \LaTeX\ code generation from the SProUTput, XTDL-XML, TFS-XML and RMRS DTDs. The xml2html.xsl stylesheet is taken from the Apache Cocoon project (under Apache Software License 1.1) and only slightly modified for module name extraction in the Heart of Gold.

<table>
<thead>
<tr>
<th>file name</th>
<th>input DTD</th>
<th>output format</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsl/html/rmrs2html.xsl</td>
<td>RMRS</td>
<td>HTML w/ Javascript</td>
</tr>
<tr>
<td>xsl/html/xml2html.xsl</td>
<td>any XML document</td>
<td>HTML w/ Javascript</td>
</tr>
<tr>
<td>xsl/latex/fs2latex.xsl</td>
<td>SProUTput, XTDL, TFS-XML</td>
<td>\LaTeX</td>
</tr>
<tr>
<td>xsl/latex/rmrs2latex.xsl</td>
<td>RMRS</td>
<td>\LaTeX</td>
</tr>
</tbody>
</table>

Most stylesheets take (optional) parameters. Please also see the following web page for more details and documentation of the FS2LaTeX package:

http://www.dfki.de/~uschaefer/fs2latex/

**Figure 1** Example for \LaTeX\ generated from RMRS with rmrs2latex.xsl
Raw Input Text Preprocessing and Sentence Splitting

As the HPSG parser parses sentence-wise, input documents have to be segmented into sentences, and analyze() has to be called separately by sentence in order to analyse a whole document.

Moreover, some of the Heart of Gold components are (still) not very robust with respect to arbitrary characters in input text and cannot directly use XML input files. We shortly address these 3 kinds of problems here.

Problem 1: confusing characters in input text, e.g. (, ) seem to confuse TnT/Chunkie under some circumstances, a " confuses the current German HPSG grammar.

Solution: use a sed script like the following to remove the confusing characters (a more elegant solution would be to preserve the original character positions which this script doesn't do):

```
s#(afp)|(dpa)|(Reuter)|(ap)#g
s/ / /g
s/"|_\|\|*\|()\|--\|)/ /g
s/--//g
s/ 000/000/g
```

Problem 2: Sentence splitting

Solution: Pre-processing can be done by the application, by calling (only) JTok or SProUT or another component on the whole input document first, using the segmentation in the returned XML document to process then deeply sentence by sentence. Sample stylesheet for JTok:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="text"/>
 <xsl:strip-space elements="*"/>

 <xsl:template match="tu">
   <xsl:for-each select="Token">
     <xsl:value-of select="@string"/>
     <xsl:text> </xsl:text>
   </xsl:for-each>
   <xsl:text>
   </xsl:text>
   <xsl:apply-templates/>
 </xsl:template>

 <xsl:template match="text()"/>
</xsl:stylesheet>
```
Problem 3: Input is not plain text, but XML file
Solution: use a XSLT-stylesheet that generate a text file extracting the input
text from the XML input document and inserting a newline character at
sentence boundary. The example stylesheet extracts #PCDATA children
under element path relevant/qa-pair/retrieved/context/sent and stores each
text with <sent> elements in a separate line in an ISO-8859-1-encoded
output file (for use in analyzeGUI or analyzeAll).

<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:strip-space elements="*"/>

  <xsl:template match="relevant">
    <xsl:for-each select="qa-pair">
      <xsl:for-each select="retrieved/context">
        <xsl:for-each select="sent">
          <xsl:value-of select="."/>
          <xsl:text/>
        </xsl:for-each>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>

  <xsl:template match="text()"/>
</xsl:stylesheet>

Both stylesheets can be found in xsl/preproc/
They could be applied by an application client using
xml2htmlTransformer(sid, acid, "JTok", "xsl/preproc/jtok-sbr.xsl", "no")
in this case, the output is not HTML, but plain text).

The Python script

analyzeAll -n [-m host] [-p port] [-l lang] [-e encoding] textfile

can be used to process a textfile containing each sentence to be processed
by PET in a seperate line.

GUI variant:
analyzeGUI [-m host] [-p port] [-l lang] [-e encoding] textfile
Part 6: Literature, Links

see also [http://heartofgold.dfki.de/Publications.html](http://heartofgold.dfki.de/Publications.html)


the Software Engineering and Architecture of Language Technology Systems (SEALTS), HLT-NAACL03, pp. 9-16, Edmonton, Canada.


[LingPipe] [http://www.alias-i.com/lingpipe/](http://www.alias-i.com/lingpipe/)
[Xindice] [http://xml.apache.org/xindice](http://xml.apache.org/xindice)
[XML:DB] [http://www.xmldb.org/xapi](http://www.xmldb.org/xapi)
[XPath] [http://www.w3.org/TR/xpath](http://www.w3.org/TR/xpath)
[XSLT] [http://www.w3.org/TR/xslt](http://www.w3.org/TR/xslt)
[XUpdate] [http://www.xmldb.org/xupdate](http://www.xmldb.org/xupdate)
Part 7: Contributors

*Concept:* Ulrich Callmeier, Andreas Eisele, Ulrich Schäfer, Melanie Siegel

*Implementation:* Robert Barbey, Özgür Demir, Ulrich Schäfer

*JTok, Module configuration and Launcher:* based on code by Jörg Steffen (MEMPHIS project)

*PET extensions (XML input chart):* Bernd Kiefer

*SDL:* Hans-Ulrich Krieger

*RMRS construction from chunks:* Anette Frank and Kathrin Spreyer

*RMRS merging stylesheets, Whiteboard topoparser:* Anette Frank

*rmrs2html.xsl:* Thomas Klöcker and Ulrich Schäfer, with ideas and styles borrowed from Stephan Oepens Javascript code for MRS (lkb.js)

*PHP-based Web demo:* Özgür Demir

*Web Service Wrapper, LingPipe configuration extensions:* Gregory Gulrajani

*LoParModule, Port of the Whiteboard topoparser XSLT pipeline:* Daniel Contag

*Deployment, RASP2Module, RASP RMRS converter server package:* Torsten Marek
APPENDIX 1: XML Annotation Database

Installation and Usage instructions for the XML annotation database

1. Using an existing XML-DB database

We assume that the XML-DB database is already running on host dbhost.

There are two scripts, startdb.sh and stopdb.sh, that start and stop the underlying tomcat application server and the xindice database engine.

In order to avoid conflicts, the database test programs developed so far append the username ($USER) to the root collection database path, e.g. xmldb:xindice://dbhost:8080/db/uschaefer/
Please use this convention in your applications as well (until user access management is available in Xindice).

A sample class that illustrates how to write your own database client is in de/dfki/lt/hog/database/testdb.java.
The main class to employ in applications is de.dfki.lt.hog.database.XMLDBAnnotationDatabase.

It includes methods to open a database, add, remove collections and documents (collections that contain XML annotations), or add, remove, retrieve, or query XML annotations. Two standard query types are supported: XPath and XUpdate. For details see the javadoc that can be generated using the ant script described below.

The database clients can run on a machine different from the database server. You only need some class files from hogskeleton.tar.gz and the sources from the cvs repository. Here is how to install and run your own (test) client:

```
cd hog
ant all    # creates jar, javadoc
ant testdb # test database (R/W access, XPath query)
ant checkdb # check database root collection
```

testdb (de.dfki.lt.hog.database.testdb) inserts some test collections and XML annotation.

checkdb uses the xindice command line administration tool (org.apache.xindice.tools.XMLTools) and outputs the top collection(s) of the database.

You can browse the collection hierarchy in your web browser through the URL http://dbhost:8080/Xindice (the "Ugly debug tool"), but inspection of XML content is not supported.
The database currently will be lost when tomcat is restarted - so exporting data to an XML file is recommended if annotation data is to be kept persistently.

If a method is missing from the XMLDBAnnotationDatabase class that you would like to have, please let me know (instead of writing your own code on the basis of the lower-level class XMLDBAccess).

2. Installation of Xindice from scratch (i.e., from the sources)

Tested with JDK 1.4.1 and 1.4.2 / uschaefe 2003-09-23

Build Xindice V1.1b2-pre as of 2003-07-25

```
tar xzf xindices1-checkout-20030725.tar.gz
mv xml-xindice xindice
cd xindice
#edit build.sh: add JAVA_HOME
./build.sh
./build.sh javadoc # goes to build/api/
```

add JAVA_HOME and XINDEX_HOME to

```
./xindice.sh and ./bin/xindice
```

Build tomcat V4.1.24

```
tar xzf tomcat-4.1.24-LE-jdk14.tar.gz
mv jakarta-tomcat-4.1.24-LE-jdk14 tomcat
cd tomcat
#edit ./bin/catalina.sh: add JAVA_HOME and CATALINA_HOME
cd ..
#insert content of xindice/dist/xindice-1.1b2.xml before </Host> in
tomcat/conf/server.xml
ln -s xindice/dist/xindice-1.1b2.war tomcat/webapps/xindice-1.1b2.war
```

Here is an example for a startup script (startdb.sh)

```
hog=/local/deepthought/CoreArchitecture
dtusers="uschaefer eisele siegel"
#ps -p `cat $hog/xindice/logs/xindice.pid`
umask 002
cd $hog/tomcat/bin
./startup.sh
cd $hog/xindice
./xindice.sh start
echo "sleeping for 10 seconds..."
sleep 10
cd ./bin
for dtmember in $dtusers ; # add user collections
```
do
  echo "adding collection for $dtmember"
  ./xindice ac -c /db -n $dtmember
done

and shutdown script (stopdb.sh)

hog=/local/deepthought/CoreArchitecture
umask 002
cd $hog/tomcat/bin
./shutdown.sh
cd $hog/xindice
./xindice.sh stop

# currently, the database should be exported before tomcat shutdown
# as a tomcat restart seems to destroy its content
APPENDIX 2: RMRS DTD
(by Ann Copestake)


<!ELEMENT rmrs-list (rmrs)*>

<!ELEMENT rmrs (label, (ep|rarg|ing|hcons)*)>
<!ATTLIST rmrs
cfrom CDATA #REQUIRED
to CDATA #REQUIRED>

<!ELEMENT ep ((realpred|gpred), label, var)>
<!ATTLIST ep
cfrom CDATA #REQUIRED
to CDATA #REQUIRED>

<!ELEMENT realpred EMPTY>
<!ATTLIST realpred
lemma CDATA #REQUIRED
pos (v|n|j|r|p|q|c|x|u) #REQUIRED
sense CDATA #IMPLIED>

<!ELEMENT gpred (#PCDATA)>

<!ELEMENT label EMPTY>
<!ATTLIST label
vid CDATA #REQUIRED>

<!ELEMENT var EMPTY>
<!ATTLIST var
sort (x|e|h|u|l) #REQUIRED
vid CDATA #REQUIRED
num (sg|pl|u) #IMPLIED
cfrom CDATA #REQUIRED
to CDATA #IMPLIED
pers (1|2|3|1-or-3|u) #IMPLIED
gender (m|f|n|m-or-f|u) #IMPLIED
divisible (plus|minus|u) #IMPLIED
cogn-st (type-id|uniq-id|fam|activ|in-foc|uniq-or-less|uniq-or-fam|fam-or-activ|active-or-more|fam-or-less|uniq-or-fam-or-activ|fam-or-more|activ-or-less|uniq-or-more|u) #IMPLIED
tense (past|present|future|non-past|u) #IMPLIED
telic (plus|minus|u) #IMPLIED
protracted (plus|minus|u) #IMPLIED
stative (plus|minus|u) #IMPLIED
incept (plus|minus|u) #IMPLIED
imr (plus|minus|u) #IMPLIED
boundedness (plus|minus|u) #IMPLIED
refdistinct (plus|minus|u) #IMPLIED>
<!ELEMENT rarg (rargname, label, (var|constant))>
<!ELEMENT rargname (#PCDATA)>
<!ELEMENT constant (#PCDATA)>
<!ELEMENT ing (ing-a, ing-b)>
<!ELEMENT ing-a (var)>
<!ELEMENT ing-b (var)>
<!ELEMENT hcons (hi, lo)>
<!ATTLIST hcons
  hreln (eq|leq|outscopes) #REQUIRED >
<!ELEMENT hi (var)>
<!ELEMENT lo (label|var)>
APPENDIX 3: ISO 639 Codes

(only selected codes, mostly European and Asian)

Technical contents of ISO 639:1988 (E/F)
"Code for the representation of names of languages".
The Registration Authority for ISO 639 is Infoterm, Osterreichisches
Normungsinstitut (ON), Postfach 130, A-1021 Vienna, Austria.

<table>
<thead>
<tr>
<th>Code</th>
<th>Language</th>
<th>Code</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>af</td>
<td>Afrikaans</td>
<td>gl</td>
<td>Galician</td>
</tr>
<tr>
<td>ar</td>
<td>Arabic</td>
<td>ha</td>
<td>Hausa</td>
</tr>
<tr>
<td>be</td>
<td>Byelorussian</td>
<td>he</td>
<td>Hebrew</td>
</tr>
<tr>
<td>bg</td>
<td>Bulgarian</td>
<td>hi</td>
<td>Hindi</td>
</tr>
<tr>
<td>br</td>
<td>Breton</td>
<td>hr</td>
<td>Croatian</td>
</tr>
<tr>
<td>ca</td>
<td>Catalan</td>
<td>hu</td>
<td>Hungarian</td>
</tr>
<tr>
<td>cs</td>
<td>Czech</td>
<td>id</td>
<td>Indonesian</td>
</tr>
<tr>
<td>cy</td>
<td>Welsh</td>
<td>is</td>
<td>Icelandic</td>
</tr>
<tr>
<td>da</td>
<td>Danish</td>
<td>it</td>
<td>Italian</td>
</tr>
<tr>
<td>de</td>
<td>German</td>
<td>ja</td>
<td>Japanese</td>
</tr>
<tr>
<td>el</td>
<td>Greek</td>
<td>ka</td>
<td>Georgian</td>
</tr>
<tr>
<td>en</td>
<td>English</td>
<td>kl</td>
<td>Greenlandic</td>
</tr>
<tr>
<td>eo</td>
<td>Esperanto</td>
<td>kn</td>
<td>Kannada</td>
</tr>
<tr>
<td>es</td>
<td>Spanish</td>
<td>ko</td>
<td>Korean</td>
</tr>
<tr>
<td>et</td>
<td>Estonian</td>
<td>ku</td>
<td>Kurdish</td>
</tr>
<tr>
<td>eu</td>
<td>Basque</td>
<td>la</td>
<td>Latin</td>
</tr>
<tr>
<td>fa</td>
<td>Persian</td>
<td>lt</td>
<td>Lithuanian</td>
</tr>
<tr>
<td>fi</td>
<td>Finnish</td>
<td>lv</td>
<td>Latvian</td>
</tr>
<tr>
<td>fr</td>
<td>French</td>
<td>mk</td>
<td>Macedonian</td>
</tr>
<tr>
<td>fy</td>
<td>Frisian</td>
<td>mo</td>
<td>Moldavian</td>
</tr>
<tr>
<td>ga</td>
<td>Irish</td>
<td>mt</td>
<td>Maltese</td>
</tr>
<tr>
<td>gd</td>
<td>ScotsGaelic</td>
<td>nl</td>
<td>Dutch</td>
</tr>
<tr>
<td>no</td>
<td>Norwegian</td>
<td>oc</td>
<td>Occitan</td>
</tr>
<tr>
<td>pl</td>
<td>Polish</td>
<td>pt</td>
<td>Portuguese</td>
</tr>
<tr>
<td>rm</td>
<td>Rhaeto-Romance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ro</td>
<td>Romanian</td>
<td>ru</td>
<td>Russian</td>
</tr>
<tr>
<td>sa</td>
<td>Sanskrit</td>
<td>sh</td>
<td>Serbo-Croatian</td>
</tr>
<tr>
<td>sk</td>
<td>Slovak</td>
<td>sl</td>
<td>Slovenian</td>
</tr>
<tr>
<td>sq</td>
<td>Albanian</td>
<td>sr</td>
<td>Serbian</td>
</tr>
<tr>
<td>sv</td>
<td>Swedish</td>
<td>tr</td>
<td>Turkish</td>
</tr>
<tr>
<td>uk</td>
<td>Ukrainian</td>
<td>vi</td>
<td>Vietnamese</td>
</tr>
<tr>
<td>yi</td>
<td>Yiddish</td>
<td>zh</td>
<td>Chinese</td>
</tr>
</tbody>
</table>
Heart of Gold FAQ

Q: Which Java version do I need?
1.4.2 currently

Q: JRE or JDK14?
JDK required

Q: Which platform should I use?
If you would like to see a running HoG quickly, try to get a linux system.

The middleware itself is Java and hence platform-independent (although there may be problems with process pipe mechanism used for some component adapters which has been reported not to be stable under Windows).
JTok, Sprout and LingPipe are pure Java should also run under Windows without changes.

However, we currently only provide linux executables in the components-*.tar.gz files of PET, TnT and Chunkie.

PET once worked perfectly with Solaris, but hasn't been tested with new additional libraries (Xerces, Boost, ECL) yet.
If you would like to compile from the sources (download from http://heartofgold.dfki.de/PET.html ), try with g++ 3.3.1.
Boost, ICU and Xerces libraries should compile without problems. Bernd told me that ECL might cause problems and could require some adaptations, but he would be happy to include patches for Solaris.

Q: Are LKB and PET compatible? I mean, can I develop my grammar in LKB and then use it with HoG?

A: Yes, that's the case. Develop with LKB, then generate a grammar image with flop (this program is not part of the packages on the heart of gold download page, but the sources can be downloaded via the PET-src-link).

However, handling of NEs and unknown words with guessed PoS is best done within Heart of Gold (you will have to define generic/prototypical lexicon entries for them).

Q: How do I get these nice HTML, LaTeX and FS applet visualization tools?
A: See pages 64 and 65.

Q: I don't see HTML visualizations of analysis results in my browser using analyzeGUI.
Make sure the browser variable at the beginning of the Python code is set to a working browser on your machine. Note that the browser needs to be running before pressing the analyze button. Moreover, remote controlling the browser only works on Linux/Unix this way. A recent Mozilla-compatible browser is required (Firefox, Seamonkey, Mozilla or Netscape).

**Q: I need MRSes instead of RMRSes from PET.**

Heart of Gold currently only supports the RMRS output mode (XMLified MRS is planned for a future release). For an interim solution, see the batch mode solution for the next question (cheap -mrs produces MRS instead of RMRSes).

**Q: How do I combine PET input charts offline (batch)?**

Here is a quick & dirty script, expecting a preceding analyzeAll with TnTpiXML and SProUTpiXML annotations written as XML files (configured at the beginning of analyzeAll), PET should be configured off in the session configuration - otherwise, it would run in vain and waste your time.

```bash
#!/bin/bash
analyzeAll -m localhost -p 8411 -l de sentences.txt
/bin/rm pet-input-charts
echo "<a/>" > dummy.xml
for file in TnTpiXML-*.xml ;
  do
echo "<?xml version='1.0'??>" >> pet-input-charts
cat xsl/pic/pet-input-chart.dtd >> pet-input-charts
gxsltproc --stringparam urilist "$file,`echo $file | sed -e "s/TnT/SProUT/g"`" xsl/pic/combinepixml.xsl dummy.xml >> pet-input-charts
echo >> pet-input-charts
done
echo >> pet-input-charts
cat pet-input-charts | LD_LIBRARY_PATH=components/pet/lib \
  components/pet/bin/cheap -tok=xml_counts -mrs -default-les -limit=70000 \
  -results=3 components/pet/german/german > mrses.txt
```

**Q: I have problems with encodings and the Python clients. What's wrong?**

This may be the case when you use an old version of the Python clients. Solution: upgrade to the new versions.
Heart of Gold will now transport both input text and generated annotations as binary objects with Unicode encoding instead of just text via XML-RPC.

**Q: TnT doesn't seem to run because of insufficient permissions.**
For some unknown reason, TnT requires group-executable permissions, i.e. being in the same group as the TnT executable should help (man newgrp).

**Q: HoG does not work with UTF-8 (Fedore Core 1 problem only)**
Stephan Oepen reported a problem with Fedora Core1 and Python TK (i.e., independent of the Heart of Gold Java code!). Here the Python and the TKinter seem to use incompatible encodings (error message: "SystemError: Py_UNICODE and Tcl_UniChar differ in size") - Details:

http://mail.python.org/pipermail/python-list/2003-November/195577.html