Software Design Description

for the

metasearch System

Written by:

Ralph Debusmann
University of the Saarland
Computational Linguistics
Email: rade@coli.uni-sb.de

October 1999

Table of Contents

1. SCOPE

1.1Identification

The metasearch system supplements the Mulinex multilingual search engine with a highly configurable metasearch facility. It utilizes the existing Mulinex search engine environment to translate search queries from one of three currently available source languages (English, French and German) to up to three target languages (also English, French and German). Following that step, the metasearch package sends the query translations to a (preferrably multilingual) search engine configured beforehand. If available after a configurable amount of time, their output is presented.

The metasearch system is written in Java (jdk 1.1.x), as is the Mulinex search engine.

1.2System Overview

The metasearch system is mainly made up of the Java package "meta", which is embedded into the main Mulinex package "mulinex". Furthermore, it makes use of the Mulinex (sub−)package "foundations" and adds the class "MetaSearch" to the "config" (sub−)package.

1.3Document Overview

This document concentrates on the components the metasearch system adds to the Mulinex search engine environment, viz. the packages "meta" and "config" ("MetaSearch" class). After briefly describing the architectural design (Section 2.1), the system components (2.2), the concept of execution (2.3) and the interface design (2.4), the detailed design of the metasearch system is explicated in Section 3.

2. SYSTEM−WIDE DESIGN DECISIONS

1.1Architectural Design

As the metasearch system is embedded into the Mulinex search engine environment, its architectural design is similar to the other packages within Mulinex.

The system’s core class is meta.MetaSearchManager. It generates instances of the classes meta.MetaSearchThread and meta.MetaSearchControlThread, the latter exerting control over the up to three (one for each available target language) instances of the former (meta.MetaSearchThread).

meta.MetaSearchManager also utilizes the classes meta.MetaHTMLPages to generate search engine result presentation pages and meta.MetaCleanup to restrict the amount of space they consume.

Search engine access is accomplished by the meta.MetaAccessor class (called from meta.MetaSearchThread instances). meta.MetaAccessor can access any (preferrably multilingual) search engine that has been configured beforehand in the "ini/MetaSearch.ini" configuration file.

2.2Components

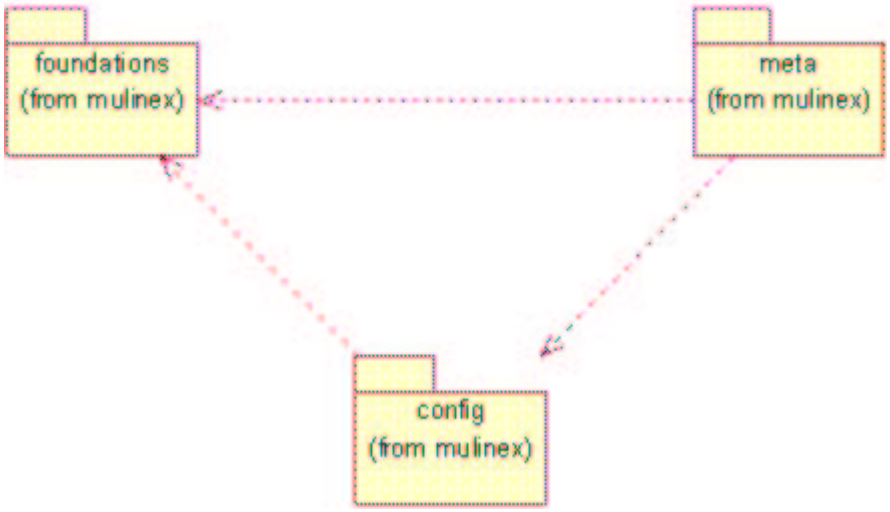

Figure 1: metasearch Architecture

CSCs local to metasearch

- mulinex
  This is the main Mulinex package into which the metasearch system is embedded. It provides the ability to translate the search query into one of three currently available target languages.
- meta
  This package constitutes the core of the metasearch system. It contains all classes and corresponding methods to access the search engine selected by the user multilingually.
- foundations
  This package provides several classes and corresponding basic methods the "meta" package relies on.
- config
  The metasearch system adds to this package the class "MetaSearch", which contains code to read and interpret the metasearch configuration file ("ini/MetaSearch.ini").

2.2.1Package mulinex

A description of the package “mulinex” lies beyond the scope of this document.

2.2.2Package meta

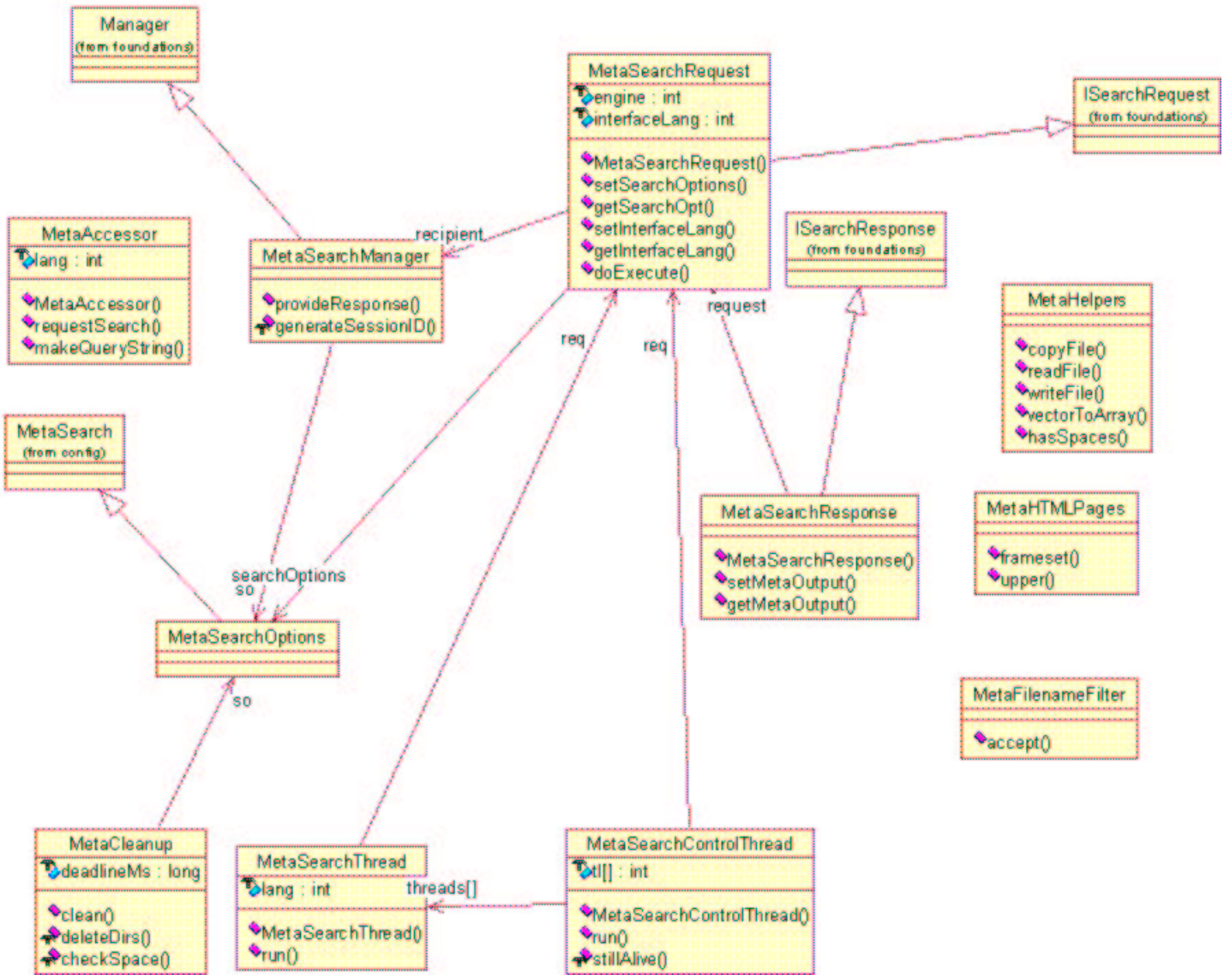

Figure 2: meta Architecture

Exported units of meta

- Class MetaSearchRequest

  ** Kickstarts MetaSearchManager.*
- Class MetaSearchManager

  ** Metasearch hub class (starts search threads and returns search response).*

- Class MetaSearchOptions

  ** Wrapper for config.MetaSearch.*
- Class MetaSearchResponse

  ** Sets/gets metaOutput.*
- Class MetaHelpers

  ** Helpers used by several classes of this package.*
- Class MetaSearchThread

  ** Kickstarts target language−specific search engine−access.*
- Class MetaCleanup

  ** Cleans up session−directories. For every new query a new "session" is created on disk, represented * by a directory "<session−ID>". There is a restriction on either the amount of space these directories * consume or their number. Only one of these restrictions can be active at a time. If this active*
  ** restriction is violated, the clean−method deletes the oldest session directories until the above*
  ** restriction is again obeyed.*
- Class MetaHTMLPages

  ** Generates HTML−pages for the search results−frameset.*
- Class MetaSearchControlThread

  ** Exerts control on the MetaSearchThreads (stops them after the timeOutMs−timeout has occured).*
- Class MetaAccessor

  ** Accesses meta search engine.*
- Class MetaFilenameFilter

  ** Implements FilenameFilter to filter session directories ("id*").*

2.2.3Package foundations

Figure 3: foundations Architecture

Exported units of foundations

- Class ISearchRequest

- Class ISearchResponse
- Class Manager

2.2.4Package config

Exported units of config

- Class MetaSearch

2.3Concept of Execution

The metasearch system is initiated by the meta.MetaSearchResponse.doExecute() method. From here, the meta.MetaSearchManager.provideResponse() method is called. This central method of the core class meta.MetaSearchManager at first commences a harddisk cleanup (meta.MetaCleanup), thereby restricting the amount of space consumed by the result presentation pages generated in sessions before. Thereafter, up to three MetaSearchThreads are started accessing the search engine chosen from the number of preconfigured search engines (one for each target language, meta.MetaAccessor handles the access itself). A control thread (meta.MetaSearchControlThread) is then started to keep track of the amount of time the threads started beforehand are requiring to come up with a search result. After a configurable timeout all threads that have not yet returned with a result from the search engine are stopped.

2.4Interface Design

The design of the interfaces to the metasearch system is described in the following sections.

2.4.1Interface Identification

The metasearch system's user interface is an HTML query enter page taken from the Mulinex search engine presentation repository. It is available in all of the three currently supported languages.

The metasearch system interface into the Mulinex search engine environment is located in the class workflow.SearchRequestSolver. Herein, the method providePresentation() at first generates an instance of the meta.MetaSearchRequest class. Parameters passed to the metasearch system are:

1) the search query (encapsulated in an foundations.Query object)
2) an int (integer number) denoting the search engine chosen for the metasearch process (out of the preconfigured engines in "ini/MetaSearch.ini")

Thereafter, the metasearch system is initiated from workflow.SearchRequestSolver.providePresentation() by a call to the meta.MetaSearchRequest.doExecute() method.

3. METASEARCH DETAILED DESIGN

3.1Package mulinex

This package is the main package and builds the fundament for the other packages described in this document. A further description of it lies beyond the scope of this document.

3.2Package meta

3.2.1Class MetaSearchRequest

* Kickstarts MetaSearchManager.

3.2.1.1MetaSearchRequest Design Specification/Constraints

3.2.1.1.1MetaSearchRequest Methods

3.2.1.1.1.1Method doExecute

```
MetaSearchResponse doExecute();
```

** Kickstarts MetaSearchManager.*

***

** @return a MetaSearchResponse from MetaSearchManager.*

3.2.1.1.1.2Method getInterfaceLang

```
int getInterfaceLang();
```

** Returns interface language.*

***

** @return the current interface language.*

3.2.1.1.1.3Method getSearchOpt

```
MetaSearchOptions getSearchOpt();
```

** Returns search options.*

***

** @return the current MetaSearchOptions*

.

3.2.1.1.1.4Method MetaSearchRequest

```
 MetaSearchRequest(Query q, int e);
```

* Class constructor.

*

* @param q a Query.

* @param e the selected search engine−ID.

3.2.1.1.1.5Method setInterfaceLang

```
void setInterfaceLang(Integer il);
```

* Sets interface language.

*

* @param il the new interface language.

3.2.1.1.1.6Method setSearchOptions

```
void setSearchOptions(MetaSearchOptions so);
```

* Sets search options.

*

* @param so the new MetaSearchOptions.

3.2.2Class MetaSearchManager

* Metasearch hub class (starts search threads and returns search response).

3.2.2.1MetaSearchManager Design Specification/Constraints

3.2.2.1.1MetaSearchManager Methods

3.2.2.1.1.1Method generateSessionID

```
String generateSessionID();
```

* Generates unique session ID.

*

* *@return String representation of a unique session ID.*

3.2.2.1.1.2Method provideResponse

```
MetaSearchResponse provideResponse(MetaSearchRequest req);
```

* *Cleans session directory up, starts MetaSearchThreads, starts*

* *MetaSearchControlThread, waits until the first of the MetaSearchThreads*

* *has returned and returns search results.*

*

* *@param req the current MetaSearchRequest.*

* *@return a MetaSearchResponse containing the search results.*

3.2.3Class MetaSearchOptions

* Wrapper for config.MetaSearch

.

3.2.4Class MetaSearchResponse

* Sets/gets metaOutput.

3.2.4.1MetaSearchResponse Design Specification/Constraints

3.2.4.1.1MetaSearchResponse Methods

3.2.4.1.1.1Method getMetaOutput

```
String getMetaOutput();
```

* *Gets metaOutput.*

*

* *@return String representation of the current metaOutput.*

3.2.4.1.1.2Method MetaSearchResponse

```
 MetaSearchResponse(MetaSearchRequest req);
```

* *Class constructor.*

*

*** @param req the current MetaSearchRequest.***

3.2.4.1.1.3Method setMetaOutput

```
void setMetaOutput(String str);
```

*** Sets metaOutput.***

*

*** @param str a String representing the new metaOutput.***

3.2.5Class MetaHelpers

* Helpers used by several classes of this package.

3.2.5.1MetaHelpers Design Specification/Constraints

3.2.5.1.1MetaHelpers Methods

3.2.5.1.1.1Method copyFile

```
void copyFile(String source, String dest);
```

*** Copies a file.***

*

*** @param source a String representing the source file path.***

*** @param dest a String representing the destination file path.***

3.2.5.1.1.2Method hasSpaces

```
boolean hasSpaces(String str);
```

*** Checks whether a string contains at least one space−character.***

*

*** @param str a String to check for at least one space−character.***

*** @return res a boolean indicating whether the String contained a space or not.***

3.2.5.1.1.3Method readFile

```
String readFile(String file);
```

** Reads a file into a String.*

***

** @param file a String representing the path of the file to read.*

** @return String representation of the file contents.*

3.2.5.1.1.4Method vectorToArray

```
int[] vectorToArray(Vector vec);
```

** Converts a Vector containing Integer entries into an int−array.*

***

** @param vec a Vector containing Integer entries.*

** @return an int−array converted from the input Vector vec.*

3.2.5.1.1.5Method writeFile

```
void writeFile(String file, String contents);
```

** Writes a String to a file.*

***

** @param file a String representing the path of the file to write.*

** @param contents a String representing the contents of the file to write.*

3.2.6Class MetaSearchThread

* Kickstarts target language−specific search engine−access.

3.2.6.1MetaSearchThread Design Specification/Constraints

3.2.6.1.1MetaSearchThread Methods

3.2.6.1.1.1Method MetaSearchThread

```
MetaSearchThread(MetaSearchRequest req, int lang, String sessionID);
```

** Class constructor.*

***

** @param req the current MetaSearchRequest.*

** @param lang the target language this thread is going to query the search engine for.*

** @param sessionID a String representing the current session−ID.*

3.2.6.1.1.2Method run

```
void run();
```

** Kickstarts target language−specific search engine−access.*

3.2.7Class MetaCleanup

** Cleans up session−directories. For every new query a new "session" is*
** created on disk, represented by a directory "<session−ID>". There is a*
** restriction on either the amount of space these directories consume or*
** their number. Only one of these restrictions can be active at a time. If*
** this active restriction is violated, the clean−method deletes the oldest*
** session directories until the above restriction is again obeyed.*

3.2.7.1MetaCleanup Design Specification/Constraints

3.2.7.1.1MetaCleanup Methods

3.2.7.1.1.1Method checkSpace

```
long checkSpace();
```

** Finds out how much space the existing sessions consume.*

***

** @return a long representing the amount of space the existing sessions ("id*"−directories) consume.*

3.2.7.1.1.2Method clean

```
void clean(MetaSearchOptions so);
```

** Cleans up session−directories.*

***

** @param so the current MetaSearchOptions.*

3.2.7.1.1.3Method deleteDirs

```
void deleteDirs();
```

** Deletes session directories.*

3.2.8 Class MetaHTMLPages

* Generates HTML−pages for the search results−frameset.

3.2.8.1 MetaHTMLPages Design Specification/Constraints

3.2.8.1.1 MetaHTMLPages Methods

3.2.8.1.1.1 Method frameset

```
String frameset(int l, String sessionID, MetaSearchOptions so);
```

** Generates search results−HTML−frameset.*

***

** @param l the target language the frameset is to be generated for.*

** @param sessionID a String representing the current session−ID.*

** @param so the current MetaSearchOptions.*

** @return String representation of the generated frameset.*

3.2.8.1.1.2 Method upper

```
String upper(int[] tl, int l, MetaSearchRequest req, int il);
```

** Generates upper HTML−frame of the search results−frameset (the lower frame contains the results themselves).*

***

** @param tl an int−array of target language−IDs.*

** @param l the target language the page is to be generated for.*

** @param req the current MetaSearchRequest.*

** @param il the current interface language.*

** @return String representation of the generated frame.*

3.2.9 Class MetaSearchControlThread

* Exerts control on the MetaSearchThreads (stops them after the timeOutMs−timeout has
* occured).

3.2.9.1 MetaSearchControlThread Design Specification/Constraints

3.2.9.1.1 MetaSearchControlThread Methods

3.2.9.1.1.1Method MetaSearchControlThread

```
MetaSearchControlThread(MetaSearchRequest req, int[] tl,
MetaSearchThread[] threads, String sessionID);
```

** Class constructor.*

***

** @param req the current MetaSearchRequest.*

** @param tl an int−array of target language−IDs.*

** @param threads a MetaSearchThread−array.*

** @param sessionID a String representing the current session−ID.*

3.2.9.1.1.2Method run

```
void run();
```

** Exerts control on MetaSearchThreads.*

3.2.9.1.1.3Method stillAlive

```
int[] stillAlive();
```

** Finds out how many MetaSearchThreads are still alive (that is have not yet returned).*

***

** @return an int−array containing the target language−IDs of all threads which are still alive.*

3.2.10Class MetaAccessor

* Accesses meta search engine.

3.2.10.1MetaAccessor Design Specification/Constraints

3.2.10.1.1MetaAccessor Methods

3.2.10.1.1.1Method makeQueryString

```
String makeQueryString();
```

** Converts QueryTerm−objects into query−String.*

***

** @return String representation of the Vector terms of QueryTerm−objects.*

3.2.10.1.1.2Method *MetaAccessor*

```
MetaAccessor(Vector terms, int lang);
```

** Class constructor.*

***

** @param terms a Vector of QueryTerm−items.*

** @param lang the search engine access target language.*

3.2.10.1.1.3Method *requestSearch*

```
String requestSearch(MetaSearchRequest req);
```

** Prepares query string and downloads results from meta search engine.*

***

** @param req the current MetaSearchRequest.*

** @return String representation of the search results HTML−page.*

3.2.11Class MetaFilenameFilter

* Implements FilenameFilter to filter session directories ("id*").

3.2.11.1MetaFilenameFilter Design Specification/Constraints

3.2.11.1.1MetaFilenameFilter Methods

3.2.11.1.1.1Method accept

```
boolean accept(File dir, String name);
```

** Implements accept−method to filter session directories ("id*").*

***

** @param dir a File representing the file to be accepted or not.*

** @param name a String representing the name of the File to be filtered.*

** @return a boolean indicating whether the file is to be accepted or not.*

3.3Package foundations

3.3.1Class ISearchRequest

Subclassed by meta.MetaSearchRequest.

3.3.2Class ISearchResponse

Subclassed by meta.MetaSearchResponse.

3.3.3Class Manager

Subclassed by meta.MetaSearchManager.

3.4Package config

3.4.1Class MetaSearch

Reads settings from "MetaSearch.ini"−file.

4. CONCLUSION

As is, the metasearch system is a considerable enhancement for the Mulinex search engine. By utilizing established search engines like AltaVista, Excite or HotBot, the metasearch system greatly increases the amount of data Mulinex can be used to retrieve information from. Thereby, it frees Mulinex from the constraint of only being able to look for information in its own, custom data base.

No piece of software is ever wholly finished. Hence, there are several directions into which further development of the metasearch system could diverge. A major enhancement would be the facility to utilize multiple search engines in parallel to retrieve information. So far, only one search engine can be used at a time.

Adoption of the above approach of using multiple search engines in parallel would trigger several problems. An important one is how to present the results of the diverse search engines in one clean, consistent way. It seems that merging the results together into one single results presentation is the most feasible solution here. However, since different search engines use different criteria to sort their results into and because every search engine presents its results in a different way, such a merge operation would prove to be rather costly to implement.

An easier solution to the merging problem would be to simply obviate it. Instead of trying to merge the results, a simpler possible enhancement to the metasearch system might simply let the user choose between the separate result pages of the several dissimilar search engines utilized.