Project ref. no.	*IST-1999-11748*
Project acronym	**LIMBER**
Project full title	**Language Independent Metadata Browsing of European Resources**

Security (distribution level)	*Public (report) – Internal (Demonstrator)*
Contractual date of delivery	*M19 July 2001*
Actual date of delivery	*25 November 2001*
Deliverable number	*D-8.2*
Deliverable name	*LIMBER Server Tools*
Type	*Report & Demonstrator*
Status & version	*Final*
Number of pages	*38*
WP contributing to the deliverable	*WP8*
WP / Task responsible	*Intrasoft International*
Other contributors	*CLRC*
Author(s)	Nikos Giannelos, Michael Wilson
EC Project Officer	*Mr. Kimmo Rossi*
Keywords	*Cross-language information retrieval, multilingual thesaurus, Thesaurus Management System, TMS, Text Classification*
Abstract (for dissemination)	The overall LIMBER system enables multilingual data retrieval through a data access system by providing a Thesaurus Management System (TMS) as a basis for translation and searching over a set of metadata. The first part of this report describes the server side of the TMS. This is a development from the report in deliverable D8.1 which included the specification and design details.The second part of this report includes a description of a tool to add keywords from a thesaurus to the metadata records to promote both mono-lingual and multi-lingual retrieval of the metadata records, and subsequently the data itself.

1. Contents

2. Executive Summary

The overall LIMBER system enables multilingual data retrieval through a data access system by providing a Thesaurus Management System (TMS) as a basis for translation and searching over a set of metadata.

The first part of this report describes the server side of the TMS. This is a development from the report in deliverable D8.1 which included the specification and design details.

The second part of this report includes a description of a tool to add keywords from a thesaurus to the metadata records to promote both mono-lingual and multi-lingual retrieval of the metadata records, and subsequently the data itself.

3. The LIMBER Thesaurus Management System Server.

The purpose of this chapter is to present the functionality and usage of the LIMBER Thesaurus Management System (TMS).

3.1 Introduction

The LIMBER TMS consists of a tool to develop multilingual thesauri and a terminology server for cataloguers and for distributed access to heterogeneous electronic collections. The distinct features of the TMS are its capability to store, develop and access multiple thesauri and their interrelations under one database schema, to create any relevant view thereon and to specialize dynamically any kind of relation into new ones.
The LIMBER TMS graphical user interface (also called **Graphical Analysis Interface - GAIN**) allows the unconstrained navigation within and between different thesauri and the execution of predefined queries and graphical views to identify concepts for cataloguing or database queries, to identify translations or equivalent expressions for information access in heterogeneous environments, and to control the quality and logical consistency of a system of interlined thesauri during the development.
The LIMBER TMS server can be integrated in a distributed, heterogeneous environment. As a central, eventually repeated component, it can replace the cumbersome implementation and population of thesaurus management features in collection databases and library systems, due to access through its programmatic interface. It further allows automatic term expansion and translation in distributed access environment. This use requires consistency of the equivalence relations established between thesauri.

3.2 Installation

The LIMBER TMS is distributed in a CD. Installation is performed simply, by double clicking on the “setup.exe” file, located in the CD’s root directory.
After the completion of the installation, the LIMBER TMS, along with the reduced HASSET thesaurus, is ready for use.

3.3 Using the LIMBER TMS Server – GAIN

3.3.1 General Description

The Graphical Analysis Interface (hereafter called *GAIN*) will be described in detail in the next few chapters. Gain is used to access and manipulate the LIMBER TMS locally (i.e. from the same computer where the TMS is installed).

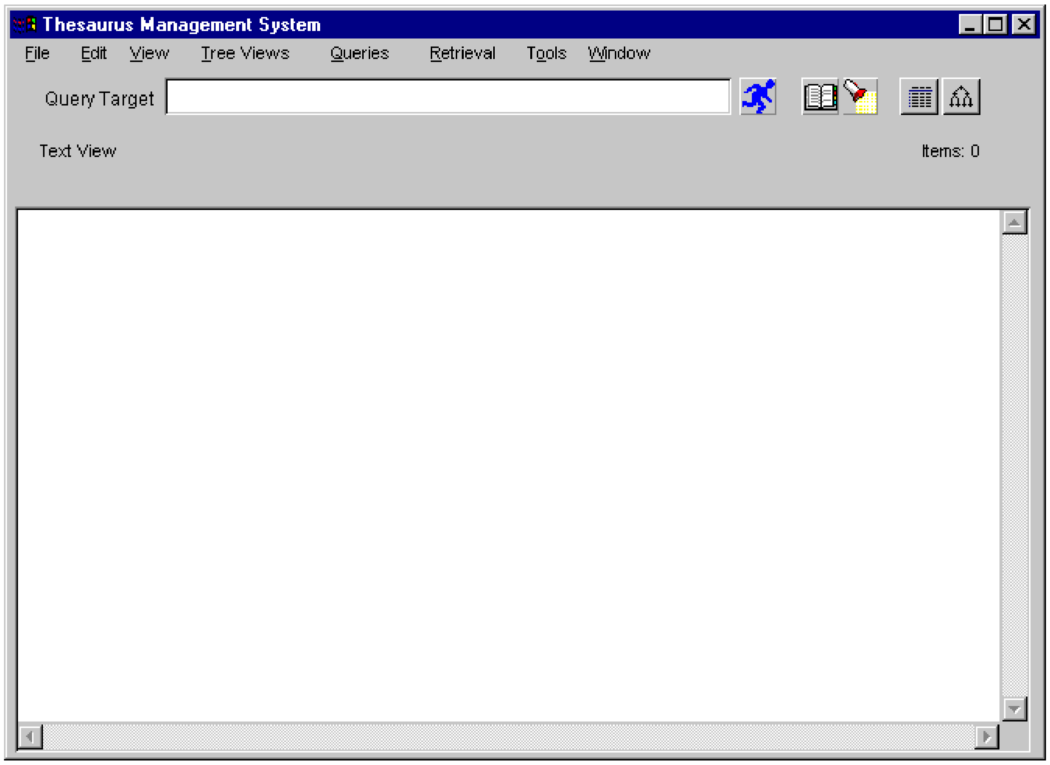


Figure 1 GAIN main window

GAIN cooperates with the query processor of the LIMBER TMS base. Information can be retrieved from the TMS, by executing one of a set of built-in queries, which are offered as a menu of choices by the interface. The query processor extracts data from the TMS base and displays the result on the screen. The result can be seen in two ways: graphically, on the window of the graphical subsystem or textually, on a text-window. There are many types of predefined queries; some of them are graphical (display the result in graphical mode), while others are textual (display the result in textual mode). The current selection of the query type is displayed by the query type field, which is always visible (see Figure 1). A query may have one parameter on which it operates or may have none. This parameter, referenced in the following as *Query Target*, must be an object existing in the TMS base. The Query Target is always visible and can be changed in multiple ways by the user.

The GAIN main window is divided in three basic areas (see Figure 1):

- the *Menu-bar*, which provides all the built-in queries and a set of operations on the visual representation of the query results.
- the *Query Info area*, which includes the *Query Target* area and a toolbar for the most frequently used operations.
- the *Query Results area*, which displays the results (graphical or textual) of the queries to the LIMBER base (hereafter referred as “Text Area” and “Graph Area”)

Except of the main window a number of pop-up windows are triggered by the menubar or the toolbar selections:

- the *Object Card window*, which displays the textual description of an object.
- the *Global View window*, which displays the global view of the graph presented in the graphical window.
- the *Options window,* which enables the user to set its preferences for the fonts, colors and text messages of the user interface area.
- the *History window*, which includes a list of the last executed queries.

3.3.2 Queries

The LIMBER TMS provides three menus of built-in queries: a) the *Tree Views,* a menu of queries whose results are displayed in graphical mode, b) the *Queries*, a menu of queries whose results are displayed in textual mode and c) the *Retrieval,* a menu of queries by classification facets, whose results are displayed in textual mode.

3.3.3 Tree Views menu

The graphical queries are performing search in depth, and they are performed on a specific target. They provide visual information about connection between objects. The query *StarView* is an equivalent graphical representation of an *Object Card.*

Figure 2 shows the *Tree View* menu. Figure 3 shows the *Global View* window, which displays the global view of the graph presented in the *Graph Area* in Figure 2.

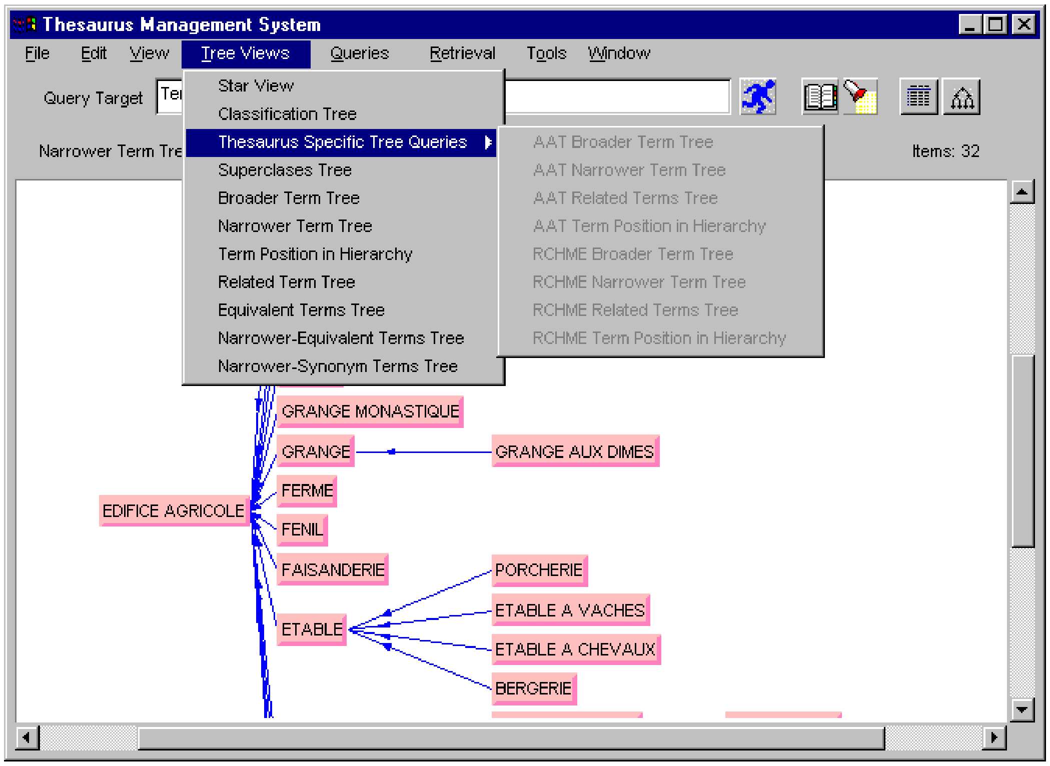


Figure 2 *Tree View* menu displays graphical query results

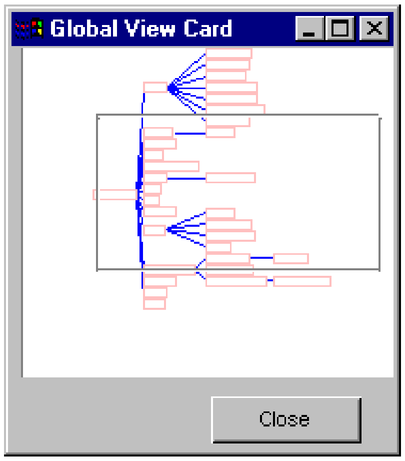
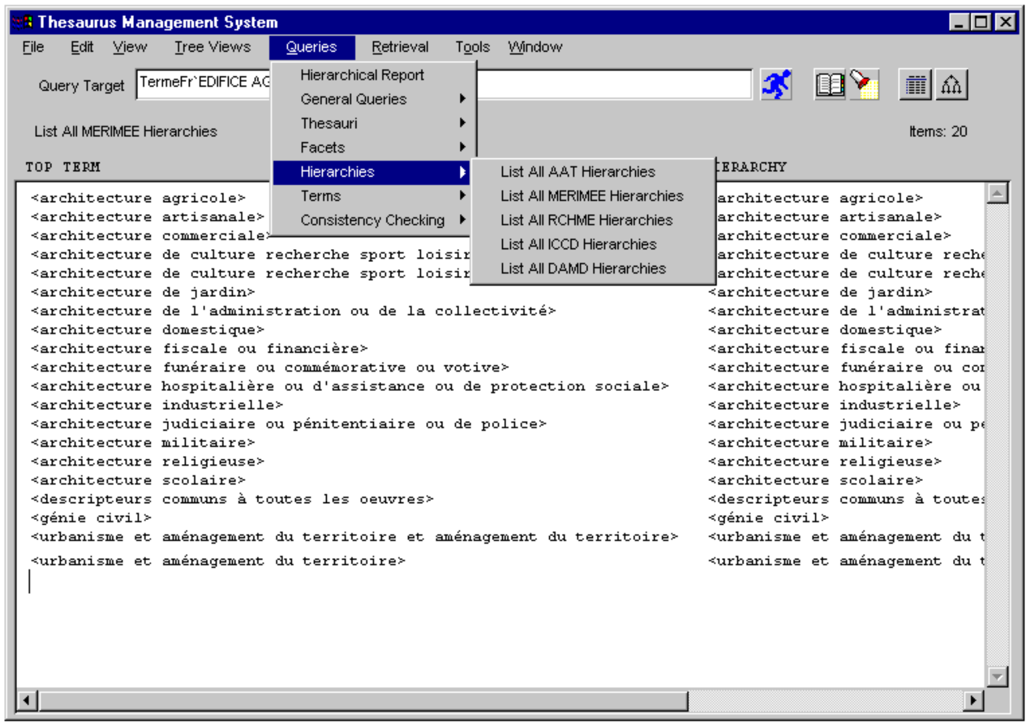


Figure 3 *Global View* window shows which part of the graph is visible.

3.3.4 Queries menu

The results of the *Queries* are displayed in the *Text area* in columns (see Figure 4). This menu provides queries about all facets, or all facets by a specific parameter (*QueryTarget*). The queries about all facets (do not apply on a specific *Query Target)* have the prefix "*List All*".



Query Target TermeFr`EDIFICE AG

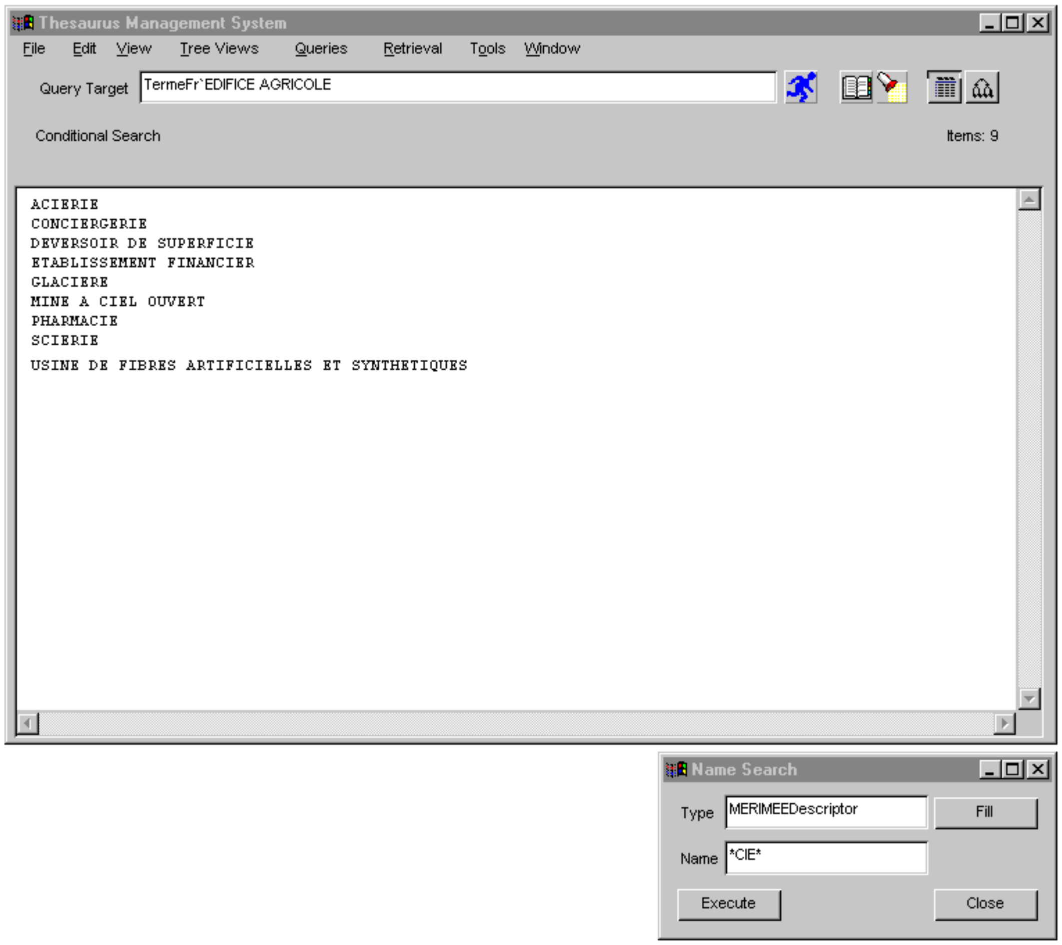
List All MERIMEE Hierarchies Items: 20

```
TOP TERM                                                                ERARCHY
<architecture agricole>                                                 architecture agricole>
<architecture artisanale>                                               architecture artisanale>
<architecture commerciale>                                              architecture commerciale>
<architecture de culture recherche sport loisir                         architecture de culture reche
<architecture de culture recherche sport loisir                         architecture de culture reche
<architecture de jardin>                                               <architecture de jardin>
<architecture de l'administration ou de la collectivité>               <architecture de l'administrat
<architecture domestique>                                              <architecture domestique>
<architecture fiscale ou financière>                                   <architecture fiscale ou finar
<architecture funéraire ou commémorative ou votive>                    <architecture funéraire ou cor
<architecture hospitalière ou d'assistance ou de protection sociale>   <architecture hospitalière ou
<architecture industrielle>                                            <architecture industrielle>
<architecture judiciaire ou pénitentiaire ou de police>                <architecture judiciaire ou pe
<architecture militaire>                                               <architecture militaire>
<architecture religieuse>                                              <architecture religieuse>
<architecture scolaire>                                                <architecture scolaire>
<descripteurs communs à toutes les oeuvres>                            <descripteurs communs à toute:
<génie civil>                                                          <génie civil>
<urbanisme et aménagement du territoire et aménagement du territoire>  <urbanisme et aménagement du t
<urbanisme et aménagement du territoire>                               <urbanisme et aménagement du t
```

Figure 4 *Queries* menu displays textual results

3.3.5 Retrieval menu

This menu provides queries about all facets by combinations of all others. The user can "fill" the specified facet from the *QueryTarget.*



Query Target: TermeFr`EDIFICE AGRICOLE

Conditional Search

Items: 9

```
ACIERIE
CONCIERGERIE
DEVERSOIR DE SUPERFICIE
ETABLISSEMENT FINANCIER
GLACIERE
MINE A CIEL OUVERT
PHARMACIE
SCIERIE
USINE DE FIBRES ARTIFICIELLES ET SYNTHETIQUES
```

Type: MERIMEEDescriptor

Name: *CIE*

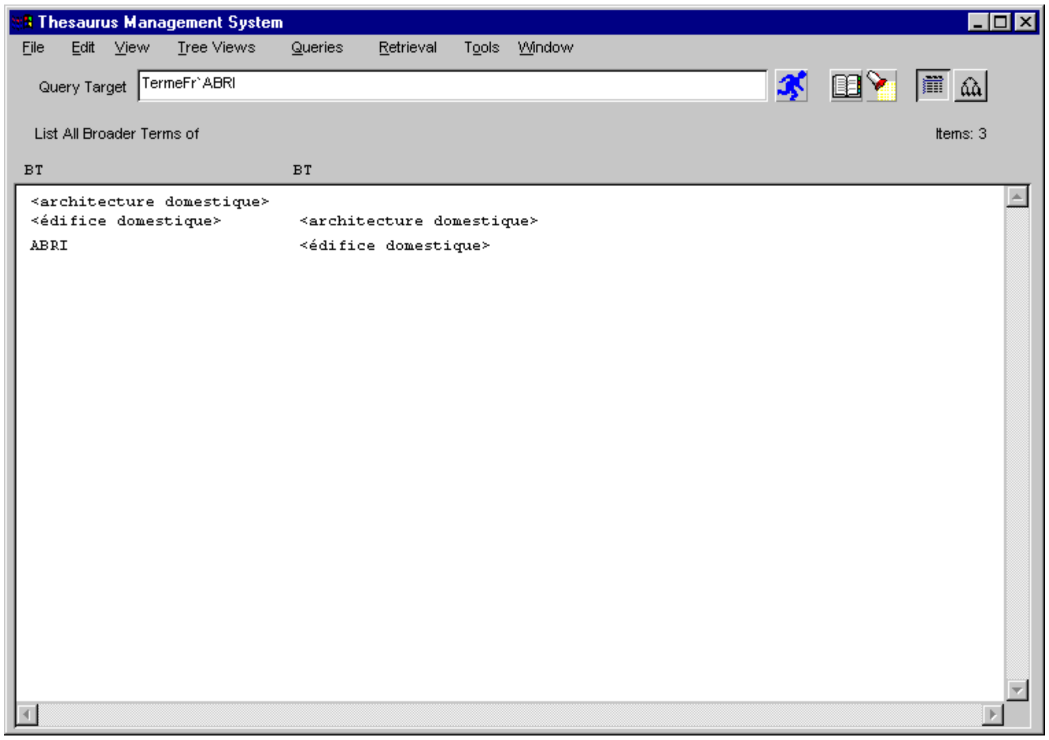
Figure 5 *Retrieval* menu displays textual results**.**

3.3.6 Query Result Presentation

The LIMBER-TMS provides various presentations for the information retrieved from the LIMBER base. Selecting a query from the *Queries* menu or the *Retrieval* menu the textual results are displayed in columns, while selecting a query from the *Tree View* menu the results are displayed as graphs. A specific *Tree View* query, called *Star View,* presents all the information associated with an object (designated in the *Query Target*) in the form of a tree-graph, while the same information is presented in textual form in a pop-up window, called *Object Card.*

3.3.7 Column Textual Display

The result of textual queries is the names of the objects existing in the answer set. This information may be presented in columns, as shown in Figure 6. Each column corresponds to an attribute of the objects in the answer set. The kind of each attribute appears as a label above the corresponding column.


Thesaurus Management System
File Edit View Tree Views Queries Retrieval Tools Window
Query Target

TermeFr`ABRI

List All Broader Terms of Items: 3

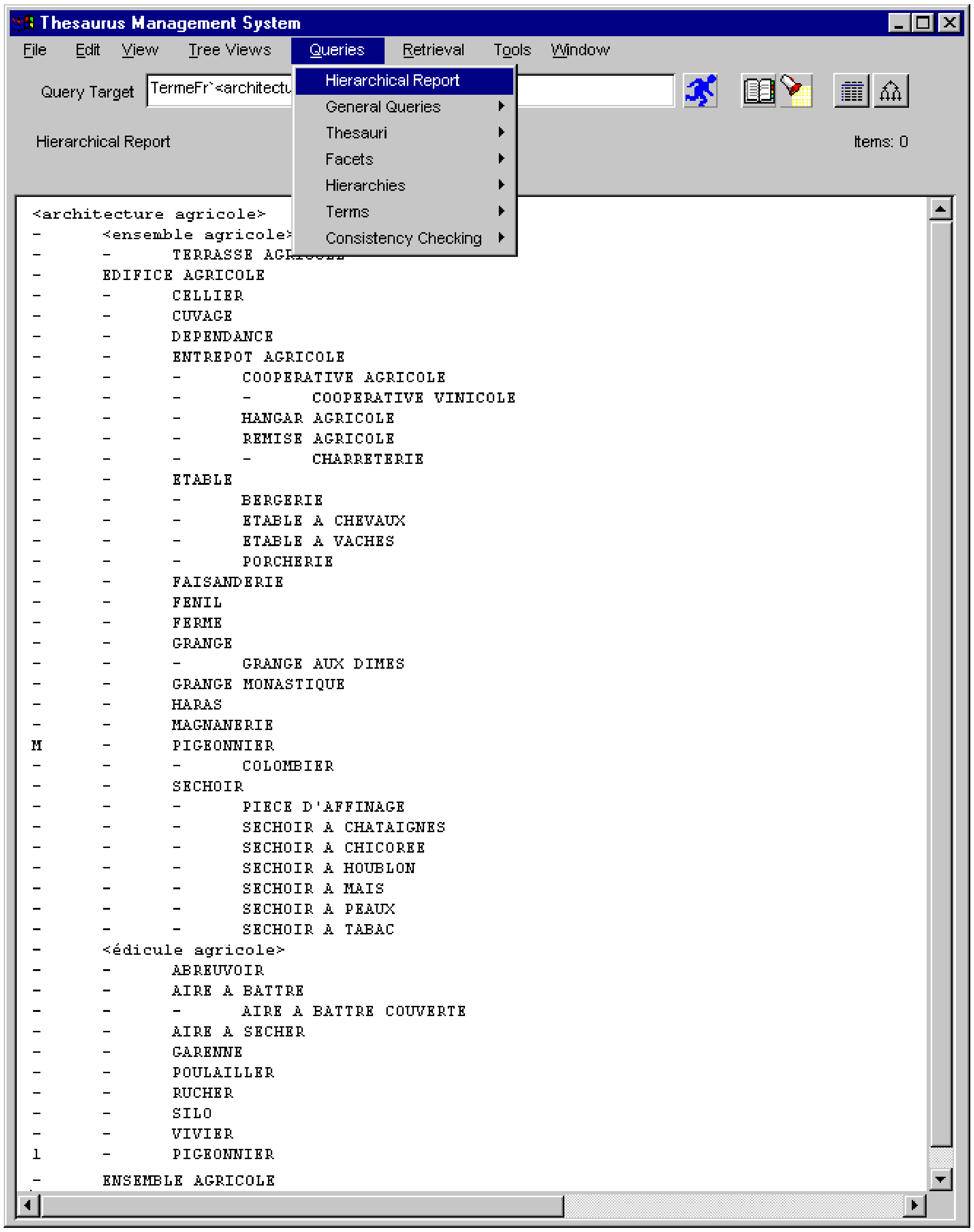
```
BT                          BT
<architecture domestique>
<édifice domestique>        <architecture domestique>
ABRI                        <édifice domestique>
```

Figure 6 *Text Area* displays the results in columns

3.3.8 Hierarchical Report Display

The result of a hierarchical report is the textual representation of a tree graph query (e.g. Narrower Term Tree) as shown in Figure 7.

When splitting has been done for a node the symbol M is added in front of this node and the number N (declaring the repetition) is also added in front of the repeated node (see PIGEONNIER node in Figure 7).



```
<architecture agricole>
-       <ensemble agricole>
-       -       TERRASSE AGR
-       EDIFICE AGRICOLE
-       -       CELLIER
-       -       CUVAGE
-       -       DEPENDANCE
-       -       ENTREPOT AGRICOLE
-       -       -       COOPERATIVE AGRICOLE
-       -       -       -       COOPERATIVE VINICOLE
-       -       -       HANGAR AGRICOLE
-       -       -       REMISE AGRICOLE
-       -       -       -       CHARRETERIE
-       -       ETABLE
-       -       -       BERGERIE
-       -       -       ETABLE A CHEVAUX
-       -       -       ETABLE A VACHES
-       -       -       PORCHERIE
-       -       FAISANDERIE
-       -       FENIL
-       -       FERME
-       -       GRANGE
-       -       -       GRANGE AUX DIMES
-       -       GRANGE MONASTIQUE
-       -       HARAS
-       -       MAGNANERIE
M       -       PIGEONNIER
-       -       -       COLOMBIER
-       -       SECHOIR
-       -       -       PIECE D'AFFINAGE
-       -       -       SECHOIR A CHATAIGNES
-       -       -       SECHOIR A CHICOREE
-       -       -       SECHOIR A HOUBLON
-       -       -       SECHOIR A MAIS
-       -       -       SECHOIR A PEAUX
-       -       -       SECHOIR A TABAC
-       <édicule agricole>
-       -       ABREUVOIR
-       -       AIRE A BATTRE
-       -       -       AIRE A BATTRE COUVERTE
-       -       AIRE A SECHER
-       -       GARENNE
-       -       POULAILLER
-       -       RUCHER
-       -       SILO
-       -       VIVIER
1       -       PIGEONNIER
-       ENSEMBLE AGRICOLE
```

Figure 7 *Text Area* displays the result of the hierarchy report

3.3.9 Tree Display

The choices of the *Tree Views* menu are recursive queries, displayed as graphs (see Figure 8). Some of them require a specified kind of target. A checking is performed when the menu is mapped on screen in order to verify that the given query target is of the kind that the queries require. All queries that require a different kind of target that the one presented in the *Query Target* area automatically become inactive.

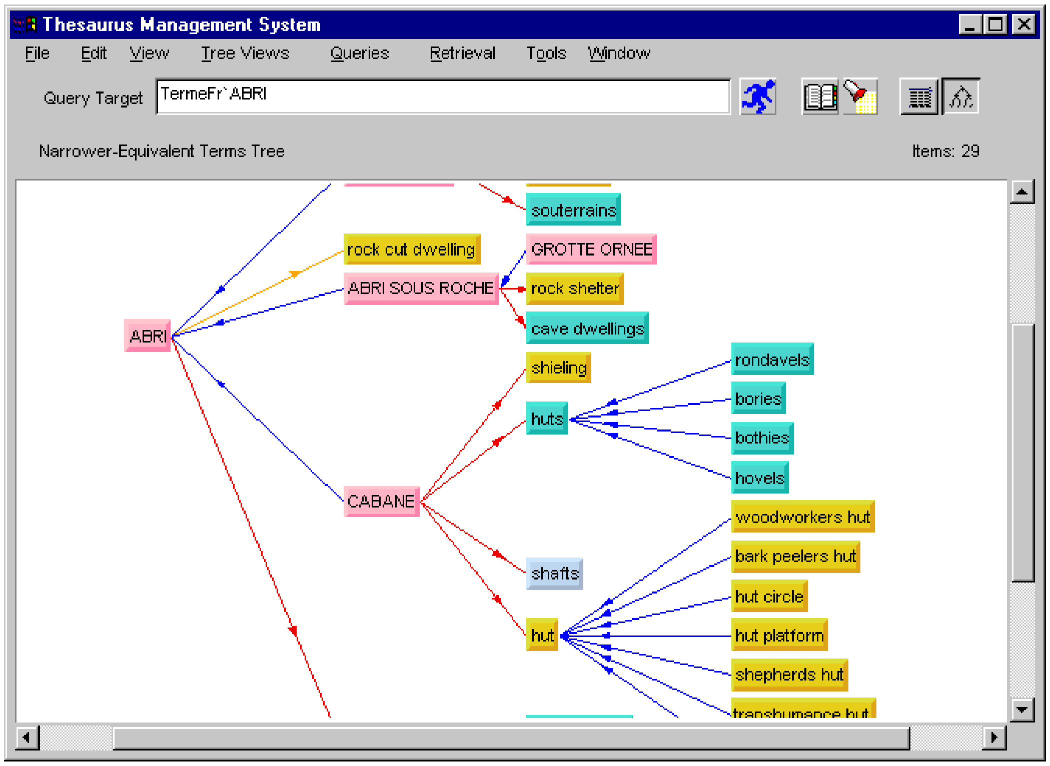


Figure 8 Tree graph results

3.3.10 Star View

By selecting *StarView* from the menu, a graphical query is executed which displays the query target as a central object (see Figure 9). The superclasses and subclasses of the central object are shown top-right and bottom-right respectively. The classes of which the central object is an instance of, are shown top-left, while if it has instances a box with the label ``INSTANCES" appears bottom-left.

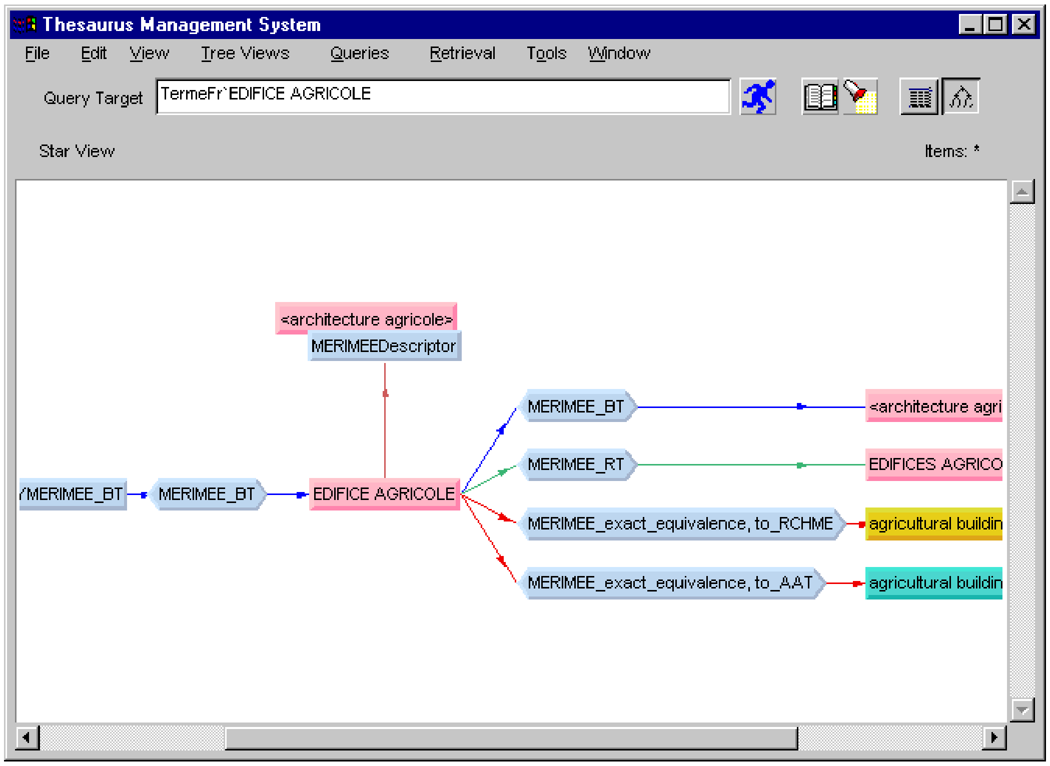
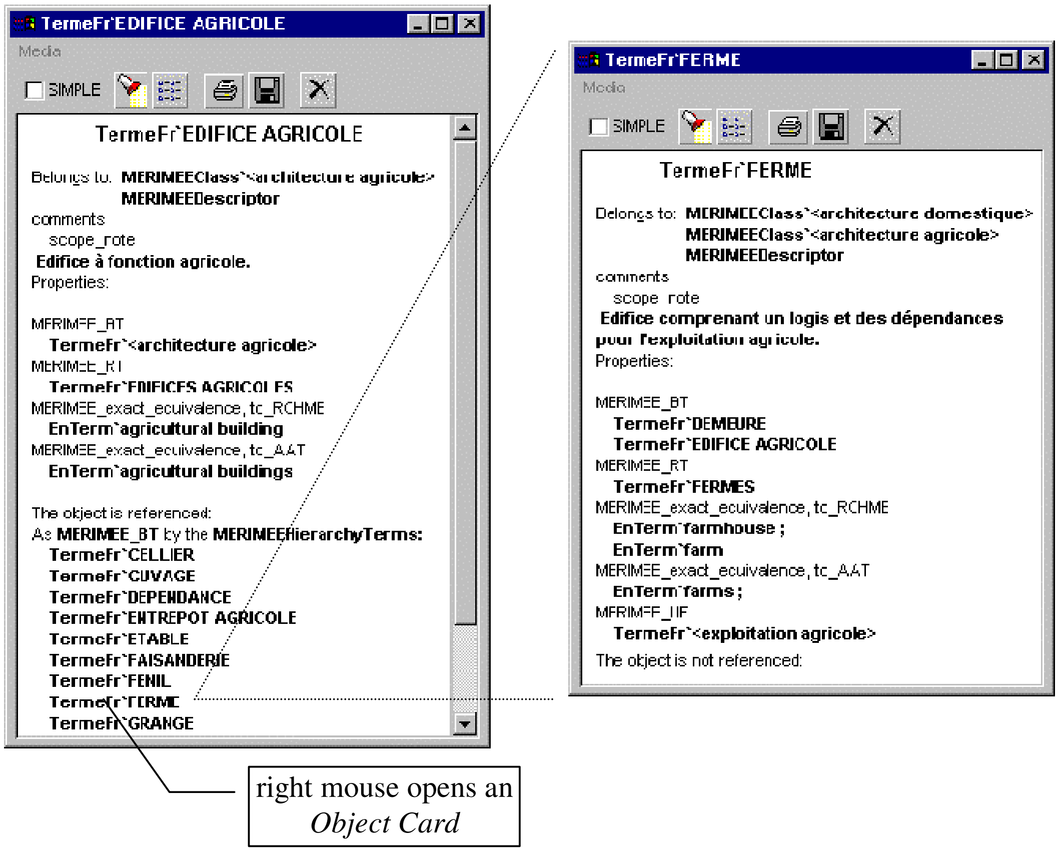


Figure 9 *Star View* result display

3.3.11 Object Card

The *Object Card* of an object contains the textual description for this object. The object card shows the complete information that is immediately related to this object (see Figure 10). The *Object Card* window is popped-up by clicking the right button on the object (its box in the graph or its name on the display, even on another *Object Card*).



```
TermeFr`EDIFICE AGRICOLE
Media
SIMPLE

TermeFr`EDIFICE AGRICOLE

Belongs to:  MERIMEEClass`<architecture agricole>
             MERIMEEDescriptor
comments
   scope_note
 Edifice à fonction agricole.
Properties:

MERIMEE_BT
   TermeFr`<architecture agricole>
MERIMEE_RT
   TermeFr`EDIFICES AGRICOLES
MERIMEE_exact_equivalence, tc_RCHME
   EnTerm`agricultural building
MERIMEE_exact_equivalence, tc_AAT
   EnTerm`agricultural buildings

The object is referenced:
As MERIMEE_BT by the MERIMEEHierarchyTerms:
   TermeFr`CELLIER
   TermeFr`CUVAGE
   TermeFr`DEPENDANCE
   TermeFr`ENTREPOT AGRICOLE
   TermeFr`ETABLE
   TermeFr`FAISANDERIE
   TermeFr`FENIL
   TermeFr`FERME
   TermeFr`GRANGE
```

```
TermeFr`FERME
Media
SIMPLE

TermeFr`FERME

Belongs to:  MERIMEEClass`<architecture domestique>
             MERIMEEClass`<architecture agricole>
             MERIMEEDescriptor
comments
   scope_note
 Edifice comprenant un logis et des dépendances
pour l'exploitation agricole.
Properties:

MERIMEE_BT
   TermeFr`DEMEURE
   TermeFr`EDIFICE AGRICOLE
MERIMEE_RT
   TermeFr`FERMES
MERIMEE_exact_equivalence, tc_RCHME
   EnTerm`farmhouse ;
   EnTerm`farm
MERIMEE_exact_equivalence, tc_AAT
   EnTerm`farms ;
MERIMEE_UF
   TermeFr`<exploitation agricole>
The object is not referenced:
```

Figure 10 *Object Card* windows

Note: If a user tries to open an already opened object card the previous one closes and the new one is built.

3.4 The web API to the TMS Server

The functionality of the LIMBER TMS Application Programming Interface (API) has been detailed in deliverable D8.1. In this chapter we will present the functionality and usage of the web API, i.e. the module that was developed in order to allow over-the-web access (HTML) to the TMS.

3.4.1 Installation

The installation of the web API is performed by copying the directory called "web API" from the CD to a directory of a web server. Since this may also require adjustments on the web server's directories, it is advised that only experienced personnel performs the installation.

After copying the files to the web server, the configuration file called "config.txt", which resides in the web API directory, has to be edited. More specifically, the first line of the configuration file has to include the IP address of the PC where th TMS is installed.

3.5 Using the web API

The web API functions as a medium between the LIMBER client and the TMS. Each time the client needs data from the TMS, it issues a request to the web API, passing the parameters of the request in the URL.
The web API parses the parameters and issues a request to the TMS. This request is issued using WINDOWS sockets (i.e. it is not over-the-web). As soon as the answer is received, the web API translates the results to a web page (HTML format) and forwards it to the client.
In the case of a command that alters the TMS base (i.e. deletion or addition of a term, etc.), the returned HTML page contains the results of the command.
The following table presents the various commands that may be issued by the client to the web API, as well as the parameters required.

Command	Description	Parameters
glst	Returns a list of all the hierarchies	None
trsl	Returns the translation of a term to a number of languages	Term, source language, list of target languages
hrch	Returns the hierarchy tree of a term	Term, source language, levels of the tree
scpn	Returns the scope note of a term	Term, source language
star	Returns a term's relations (BT, NT, RT, etc.)	Term, source language
newt	Creates a new term	Term, language, broader term
adlk	Creates a new relation between two terms	Term1, term2, source languages, type of relation
renm	Renames a term	Term, language, new name
dnod	Deletes a term	Term, language
dlnk	Deletes a relation between two terms	Term1, term2, type of relation

The results returned from the web API are converted to graphics by the client and then presented to the user. However, it is possible to view the HTML format of the results by using a web browser.
For example, let us assume that the web API resides on the web address: http://www.TMS.net/webAPI/webapi.exe. If we want to get the star view of the English term "WAGES", we would type the following URL on the browser: http://www.TMS.net/webAPI/webapi.exe?star&term=WAGES&source=En.
The following page is returned:

9957/EARNINGS/UF
11087/PAY (WAGES)/UF
11288/REMUNERATION/UF
11312/REWARDS (WAGES)/UF
11322/SALARIES/UF
5068/AVERAGE EARNINGS/RT
8545/FEES/RT
8549/FRINGE BENEFITS/RT
5168/INCOME DISTRIBUTION/RT
3993/LABOUR (RESOURCE)/RT

3994/LABOUR ECONOMICS/RT
5243/WAGES POLICY/RT
4076/SALAIRES/L1
4193/SALARIOS/L3

All information about the term “WAGES” is included in this page, but it is not in a graphical format.

The LIMBER Indexing Tool

4. Introduction

This chapter describes the design and implementation of the LIMBER indexing tool. This section briefly summarises, then continues from the section 3 of D8.1 covering the design of the LIMBER server tools.

This is a tool to generate keywords to be used to index metadata records. The tool is a generic text categorisation tool.

The indexing tool is used by the metadata indexer after an initial metadata record has been created for a new archive asset to produce keywords for it.

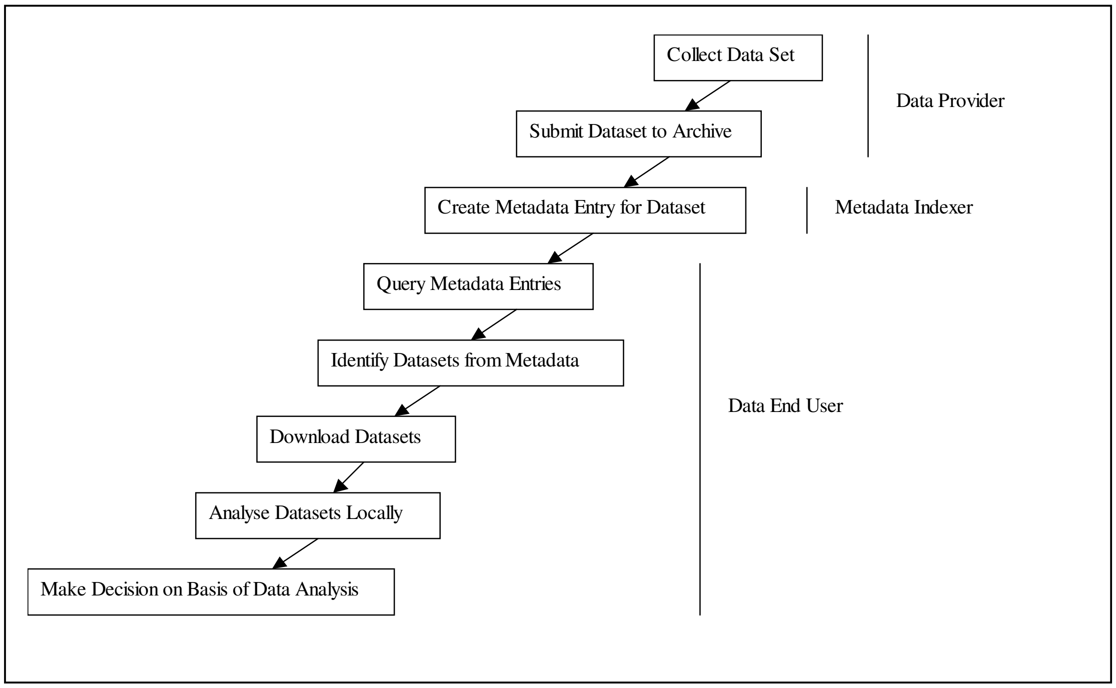

Collect Data Set
Submit Dataset to Archive
Create Metadata Entry for Dataset
Query Metadata Entries
Identify Datasets from Metadata
Download Datasets
Analyse Datasets Locally
Make Decision on Basis of Data Analysis
Data Provider
Metadata Indexer
Data End User

Figure 1. Lifecycle of Data and Metadata for Data archives

The indexing tool is a stand alone tool in the LIMBER architecture, taking a file as input and generating a file as output. The indexer requires to be trained on a training set of pre-existing, records already marked up wwith keywords. The tool will select the keywords for a new record on the basis of correlations with this training set. Therefore the automatic indexing cannot use any keywords that have not been used before.
To overcome this limitation, the tool allows the user to add extra keywords to a record that are not produced automatically. As new records are generated, they can join the training set, and be used to improve the indexing tool.

5. Architecture of the Indexer

The Indexer is not considered an end-user application, but is a separate application that can be used by metadata indexers to use a thesaurus to generate keywords for fields in the metadata entry for a data set. The consequence of providing this tool is to de-skill the job of metadata indexers which is currently a bottleneck in the use of metadata, and therefore cross domain data access. The following architecture is based on the current indexing procedure use case in section 1.2.1 of the User Requirements deliverable D1.

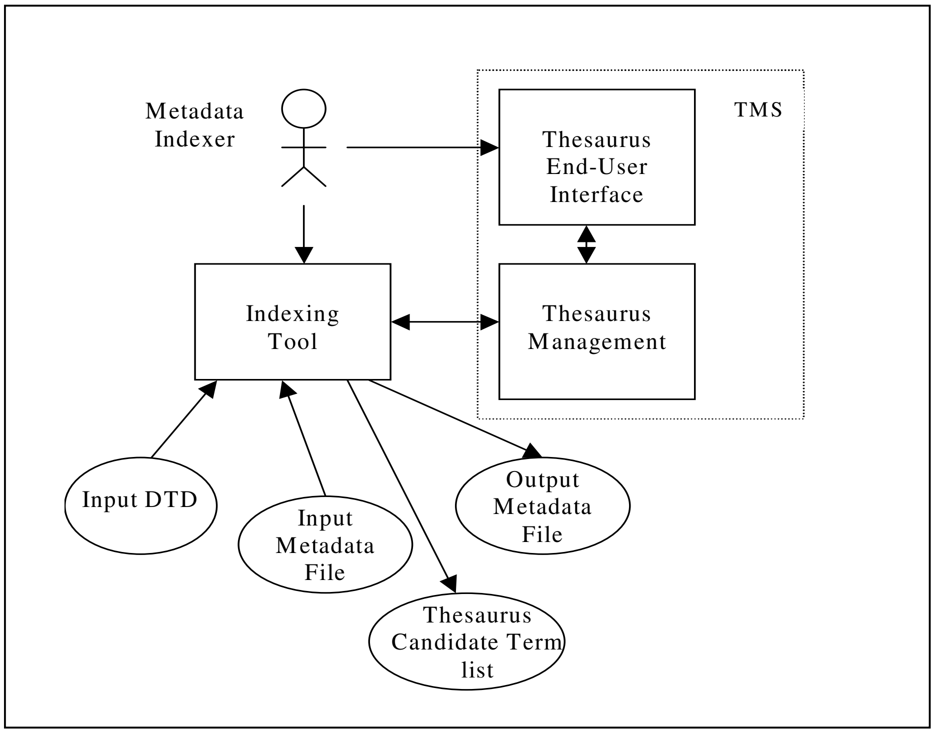


Figure 4. Architecture of the Indexing Tool

As it can be seen in Figure 4, the indexing tool takes as input and produces as output the following:

Input

1. A machine readable XML DTD of the metadata file (or XML schema definition or RDF schema definition as appropriate) – about 200 different fields that can contain text, about 20 of these can contain keywords, but some (e.g. variables) can be repeated many times in an actual metadata instance.

2. a machine-readable file containing a metadata representation in the DDI XML (or XML schema, or RDF) format for metadata. It will contain the text entries for fields in the natural language and fields to contain keywords. Such a metadata entry may contain 500 fields from which key words can be extracted.

3. The user will also state the natural language.

Output

1. The output will be the input-2 metadata document for a dataset, with the additional inclusion of keywords fields that is appropriate.

2. A machine readable file of a list of terms which are candidates for inclusion in a thesaurus

The indexing tool should work via the thesaurus server to any particular thesaurus.

Additionally, having used the indexer tool to generate appropriate keywords for a metadata entry, it would be desirable to present these results back to the indexing subject expert for inspection. This should also allow the deletion of any keywords from those suggested by the automatic indexer, and addition of any other appropriate keywords from the thesaurus, or any other keywords which if not in the thesaurus should be submitted as candidates for addition to the thesaurus. These changes would then be fed back to the indexer tool as part of its learning cycle to improve its future performance.

5.1 The General Machine Learning Approach

The machine learning approach uses a general *inductive process* to automatically build a *classifier* for a category c_i by observing the characteristics of a sample set of documents which have been classified manually by a domain expert. The task of building the classifier for all of C can then be seen m independent tasks. This method has the advantage of concentrating the engineering effort on the development of general methods for building the classifier, which can then be simply adapted to a new domain of application, or the extension of an existing domain, without the intervention of a knowledge engineer to construct the new rule set. The domain expert is only required to provide a set of classified examples for the algorithm to learn from.

Thus the machine learning approach requires the provision of an initial corpus of documents D_0 already categorised with the same categories C. This initial corpus is typically further subdivided into two:

- A *training* set T_r - the set of example documents used to construct the classifier.
- A *test* set T_e - the set of example documents used to evaluate the effectiveness of the classifier.

Clearly, for a sensible evaluation of the effectiveness of the classifier, there has to be a balance between the two sets, and also a good coverage of the categories in the corpus.

The machine learning approach to classification relies heavily on the techniques of the related field of *Information Retrieval, which* is the study of how information on a desired topic can be found from within a large corpus of (unstructured or semi-structured) documents.

Once the classifier has been constructed then there should be an evaluation phase to make an assessment of the effectiveness of the classifier, using the test set T_e.

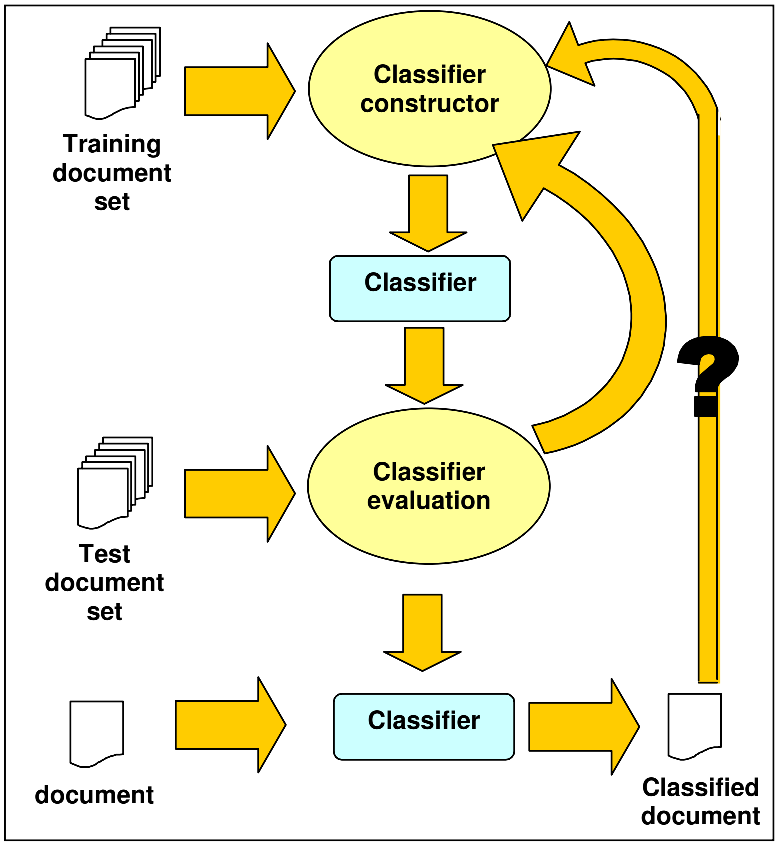


Figure 5: The Machine Learning approach to classification

The automatic classification tool for DDI-XML documents will undertake the following phases: pre-processing, indexing, classifier construction, classifier evaluation. Each of these is briefly addressed below.

5.2 Reading and preprocessing the document.

Step 1 is to read the set of training documents. This should be done as one batch job as the construction of the classifier typically is across all the training set at once.

Whilst reading in the document (whether training, test or production example), it would be desirable to do some processing on the document as follows.

5.2.1 Flattening the structure of the metadata record.

The metadata records are XML documents marked up according to the DDI. The XML structure (tags, tag names and attributes) should be stripped from the document, leaving a stream of text for the classifier to work upon.

5.2.2 Removing stop words.

Having reduced the metadata record to a stream of characters, we want to reduce it further by removing stop characters, dividing into a sequence of words and removing stop words.

- Remove white space and punctuation to divide character stream into words.
- Remove numeric and partial numeric "words".
  Issue: the date of the study is a keyword; this could be added by a separate analysis.
- Remove "stop" words – that is commonly occurring words, such as "the" "and", "then", which are not domain specific.
  Issue: gaining access to and recognising stop words in other languages (German, French, Spanish).
- Standardising on case.

5.2.3 Stemming (optional).

If possible, we could reduce the number of words, and record them as occurring more often via stemming, that is reducing grammatical variants of the same word to their root. E.g. educate, educates, educating, educated all reduce to the root word, educate.
Issue: obtaining access to stemming algorithms. Stemming algorithms are freely available in English (e.g. the *Porter algorithm*[Porter]); language specific searches may well turn up stemmers in other target languages.
Issue: there is some question in the literature whether stemming adds any better performance to categorisation methods. It is proposed that the utility of stemming is tested in the evaluation.

5.3 Indexing the document

The classifier cannot directly handle the documents, but first processes them into a more efficient internal representation; this process is known as *indexing* (somewhat confusingly as in this project we have been using the term indexing to refer to the whole classification process!). The indexing, which is applied to training documents, test documents and additional documents in the use of the classifier

The internal representation of documents most commonly used is a *vector of weights:*

$$\overrightarrow{d}_j = < w_{1j}, \ldots w_{|T|j} > \qquad (1).$$

Where T is the number of terms (typically, but not always, words[1]) that occur at least once in the training set, and w_{ij} is the weight of term t_i in document d_j, roughly speaking capturing how much the term t_i contributes to the document.

The weights w_{ij} are typically set to fall between 0 and 1, and most commonly set using (a variant of) the *tfidf* indexing function[2] from IR, defined as:

$$tfidf(t_k, d_j) = \#(t_k, d_j) \cdot \log \frac{|T_r|}{\#_{T_r}(t_k)} \qquad (2).$$

where $\#(t_k, d_j)$ is the number of times term t_k occurs in document d_j (*term frequency*) and $\#_{T_r}(t_k)$ is the number of documents in the training set T_r the term t_k occurs in (*document frequency*). This captures the notion that a term is more significant if it occurs many times in a document, but is less discriminating if it is frequently occurring throughout the training set.

This function is then usually *normalised* so that it the weight of the term in the document lies between 0 and 1:

$$w_{kj} = \frac{tfidf(t_k, d_j)}{\sqrt{\sum_{s=1}^{|T|}(tfidf(t_s, d_j))^2}} \qquad (3).$$

This means that documents of varying length are treated equally.

5.3.1 Term Reduction

A problem with classification algorithms reported in the literature is that the large number of terms T which need to be considered. In typical applications the size of T is very large and this causes problems in scaling the classification construction algorithm. Also, too many terms may lead to a problem of *overfitting* where contingent properties are used for classification purposes. For example, give then training examples of "yellow Porsche", "yellow Ford" and "yellow Renault" all leading to a classification under "CAR" may lead the example "yellow banana" to be classified under "CAR" because of the contingent term "yellow". Consequently, most classification methods recommend reducing the number of terms to a smaller set T' of the most relevant terms, where $|T'| \ll |T|$.

Many methods have been proposed for term reduction, including sophisticated statistical, probabilistic and information theoretic techniques. Also, term clustering and semantic indexing techniques attempt to synthesize an alternative term sets to

[1] Alternatively terms could be phrases from the document, or other semantic units found in the document. The literature reports that more sophisticated terms may not always have a beneficial affect, probably because as the term's complexity increases, its occurrence in documents is likely to decrease.
[2] *Term frequency – inverse document frequency* function. There are many variants on this function in the literature.

index against by analysing the document set to extract phrases, or semantically meaningful units.

For the purposes of the Limber project, we propose that trials should be undertaken to determine whether term reduction will be needed, and if so, we propose that the simplest possible approach is taken, that of *document frequency*. This is simply to take into account those terms which occur more than some threshold (typically 2, 3, or 4) number of times in the training set. While simple, and somewhat counter-intuitive (infrequently occurring terms can be highly indicative of a particular category) the literature reports that this method is nevertheless highly effective (though perhaps not as effective as other, computationally and conceptually more complex techniques).

5.4 Constructing a classifier

Two approaches to constructing a classifier have been included in the tool: the Knn and Bayesian methods.

5.4.1 Probabilistic or Bayesian Methods

Probabilistic methods usually attempt to estimate the probability that a given

$$P(c_i \mid \vec{d}_j) = \frac{P(c_i)P(\vec{d}_j \mid c_i)}{P(\vec{d}_j)}$$

document (vector) falls within a particular category c_i. These are usually based on the well known Bayes' theorem. These methods usually are usually *naïve* since they make the assumption that the terms in the documents are independent, so the simplifying equation can be established as follows:

$$P(\vec{d}_j \mid c_i) = \prod_{k=1}^{|T|} P(w_{kj} \mid c_i)$$

The methods discussed in the literature hinge on the various ways in which the values of the probabilities can be estimated from the training set.

5.4.2 kth Nearest Neighbour

The kth Nearest Neighbour (kNN) method is a *lazy-learner*. It does not explicitly build a classifier, but rather maintains a representation of the training document set and when it comes to classify a new document it dynamically compares the new document d_j with the training set and makes a judgement as to which training documents it has seen before are most similar. For some number *k* which has to be decided by experiment, the *k* most similar documents are chosen. If the number of these *k* documents which are classified under classification c_i exceeds a threshold, also empirically chosen, then the new document can be classified under c_i.

Formally, classifying a document d_j under the category c_i using the kNN method requires computing:

$$\sum_{d_z \in Tr_k(d_j)} \boldsymbol{RSV}\ (\boldsymbol{d}_j, \boldsymbol{d}_z) \cdot \breve{\Phi}(\boldsymbol{d}_z, \boldsymbol{c}_i)$$

where RSV is a measure function of the similarity of two documents, such as

$$\boldsymbol{RSV}\ (\boldsymbol{d}_j, \boldsymbol{d}_z) = \frac{\boldsymbol{d}_j \bullet \boldsymbol{d}_z}{|\boldsymbol{d}_j||\boldsymbol{d}_z|}$$

where $d_i \bullet d_j$ is the dot product of the vectors, $|d_i|$ the norm (length) of the vector, and

$$\breve{\Phi}(\boldsymbol{d}_z, \boldsymbol{c}_i) = \begin{cases} 1 & \text{if } \boldsymbol{d}_z \text{ is categorised under } \boldsymbol{c}_i \\ 0 & \text{if } \boldsymbol{d}_z \text{ is not categorised under } \boldsymbol{c}_i \end{cases}$$

The kNN method is considered in the literature to be a good performer, lacking some of the problems that linear classifiers have. However, it is more expensive in terms of data, as the vector representation of the training set has to be maintained, and computationally more expensive, as the whole training set has to be tested against the new document.   This problem of complexity can be overcome in part by clustering the training examples, building a linear classifier for each cluster, and then using those representative examples to decide the classification. While having good results, this requires the added complexity of a clustering algorithm.

5.4.3 Profiling the input files

In conventional text classification a complete input file would be used to build the classified, and a the complete file would have keywords associated with it. In the example of the metadata records used in LIMBER this is not the case, and only selected fields in the input require to be indexed, and only a range of fields in the metadata record should be used to construct the classifier. In effect, individual fields will be treated as though they were separate documents for conventional classification.

It is necessary to state which parts of an input file should be used to construct the classifier, and which to be classified. The solution chosen is to define a profile which operates over the input documents defining these. The profile should be written in one of the XML languages in order to maintain consistent minimal code implementation. The language chosen for this is XSLT since this itself will allow a standard processor to transform the input files.

An example XSLT profile for the DDI DTD is shown below:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

        <xsl:preserve-space elements="*"/>
        <xsl:output indent="yes"/>
```

```
	<xsl:template match="codeBook">

		<!-- N.B. the format of the transformed output XML *must* be flat.
i.e. no nested tags. -->

		<sourceDtd
src="http://www.icpsr.umich.edu/DDI/CODEBOOK.TXT"/>
		<indexAtomicity type="whole" />
		<targetElement elem="stdyDscr/stdyInfo/subject/keyword" />
			<!-- TODO eliminate the duplication of this string from above -
->
		<xsl:apply-templates select="stdyDscr/stdyInfo/subject/keyword"
mode="keyword"/>
		<useElement>
			<!-- These select attributes specify the XPaths of the nodes to
be parsed for textual data. -->
			<!-- by default, all enclosed tags are also parsed (unless
specified in the below sections).-->
			<!-- Those nodes whose children are also parsed are
commented as 'wildcarded' for clarity.  -->
			<!-- Paths without leading slashes are specified relative to
/codeBook, and those with a    -->
			<!-- leading // can be found anywhere in the codeBook. The
latter should be avoided, as it  -->
			<!-- encurs a serious performance hit - crossreferencing with
the codeBook DTD should for   -->
			<!-- instance show that "//varGrp/labl" would be better written
as "dataDscr/varGrp/label". -->

			<xsl:apply-templates select="stdyDscr/citation/titlStmt"/>
					<!-- wildcarded -->
			<xsl:apply-templates select="stdyDscr/citation/serStmt"/>
					<!-- wildcarded -->
			<xsl:apply-templates select="stdyDscr/citation/biblCit"/>
			<xsl:apply-templates select="//abstract"/>
			<xsl:apply-templates
select="stdyDscr/stdyInfo/sumDscr/universe"/>
			<xsl:apply-templates select="//nation" />
			<xsl:apply-templates select="stdyDscr/othrStdyMat/relMat"/>
			<xsl:apply-templates
select="stdyDscr/othrStdyMat/relMat/citation/titlStmt"/>    <!-- wildcarded -->
			<xsl:apply-templates
select="stdyDscr/othrStdyMat/relMat/citation/serStmt"/>    <!-- wildcarded -->
			<xsl:apply-templates
select="stdyDscr/othrStdyMat/relMat/citation/biblCit"/>     <!-- wildcarded -->
			<xsl:apply-templates select="//varGrp/labl"/>
			<xsl:apply-templates select="//varGrp/txt"/>
			<xsl:apply-templates select="//var/labl"/>
			<xsl:apply-templates select="//var/qstn/preQTxt"/>
```

```
                        <xsl:apply-templates select="//var/qstn/qstnLit"/>
                        <xsl:apply-templates select="//var/qstn/postQTxt"/>
                        <xsl:apply-templates select="//var/catgryGrp/labl"/>
                        <xsl:apply-templates select="//var/catgryGrp/txt"/>
                        <xsl:apply-templates select="//var/catgry/labl"/>
                        <xsl:apply-templates select="//var/catgry/txt"/>
                        <xsl:apply-templates select="//var/concept"/>
                        <xsl:apply-templates select="otherMat/citation/titlStmt"/>
                                    <!-- wildcarded -->
                        <xsl:apply-templates select="otherMat/citation/serStmt"/>
                                    <!-- wildcarded -->
                        <xsl:apply-templates select="otherMat/citation/biblCit"/>
                                    <!-- wildcarded -->
                </useElement>
                <xsl:apply-templates select="stdyDscr/citation/prodStmt/prodDate"
mode="mEK"/>
                <xsl:apply-templates
select="stdyDscr/citation/prodStmt/prodDate[@date]" mode="mEKdate"/>
                <xsl:apply-templates select="//nation" mode="mEK"/>
                <xsl:apply-templates select="//geogCover" mode="mEK"/>
        </xsl:template>

        <!-- Here, the match attribute list the possible XPaths of the nodes whose
children must not be parsed. -->
        <!-- Fully qualified paths (e.g. stdyDscr/citation/biblCit) can be used to
remove ambiguity - although  -->
        <!-- omitting the full path also can be used as a form of wildcarding. Each
section has to have its own -->
        <!-- unique "mode" identifier - set in the mode attribute of the appropriate
xsl:apply-templates and    -->
        <!-- xsl:template tags.

                -->

        <xsl:template
match="stdyDscr/citation/biblCit|abstract|universe|nation|relMat|labl|txt|preQTxt|qstn
Lit|postQTxt|concept">
                <xsl:value-of select="text()"/>.
        </xsl:template>

        <!-- and a second similar section for the <makeElementKeyword> nodes -->

        <xsl:template match="prodDate|nation|geogCover" mode="mEK">
                <makeElementKeyword>
                        <xsl:value-of select="text()"/>
                </makeElementKeyword>
        </xsl:template>

        <!-- and finally a special case template for extracting the 'date' attribute from
<prodDate> tags -->
```

```
        <xsl:template match="prodDate" mode="mEKdate">
                <makeElementKeyword>
                        <xsl:value-of select="@date"/>
                </makeElementKeyword>
        </xsl:template>

        <!-- the template for keyword tags already present in the source document -->

        <xsl:template match="*" mode="keyword">
                <keyword>
                        <xsl:value-of select="text()"/>
                </keyword>
        </xsl:template>

</xsl:stylesheet>
```

5.5 Classifier Evaluation

Once a classifier has been constructed it should be evaluated against the test set T_e to measure its effectiveness as a classifier against the classifications given to the test set by the human expert. Once the evaluation phase has been carried out, the parameters of the classification algorithm can be adjusted to generate a new classifier for evaluation. Thus by this iteration, an improved classifier can be developed.

The standard measures of effectiveness for text classification system are precision and recall.

precision: if a document *d* is classified under category *c,* then this decision is correct,
recall: if a document should be classified under category c, then this decision is made.

To make an estimate of these values, we test the classifier against the test set T_e and record:
- *FP_i – the false positives for category c_i* ,
- *TP_i – the true positives for category c_i* ,
- *FN_i – the false negatives for category c_i* , and
- *TN_i – the true negatives for category c_i* .

Then precision and recall for category c_i can be estimated as:

$$\hat{\pi}_i = \frac{TP_i}{TP_i + FP_i}$$

$$\hat{\rho}_i = \frac{TP_i}{TP_i + FN_i}$$

These can then be either *microaveraged* across all categories (summing over all individual decisions and then calculating precision and recall) , or *macroaveraged* (calculating the precision and recall for each category and then averaging those results.

Some trade-off between them – depends what you want to do! Is a low level of FN more important than a low number of FP? Trade off between missing relevant records and swamping

- short documents (e.g. questions) – may want a higher recall (i.e. false positives more acceptable) and lower precision
- when searching for whole metadata records, then higher precision may be preferable at the expense of recall: some entries may be missed so that the user is not swamped with results.

Again, evaluations on these measures vary and are subject to empirical testing.

5.6 Multilingual Indexing

In principle, both the indexing algorithm and particular methods can be used for classification over any language. Thus the same machine learning tool should be applicable to any set of metadata records.

Two components which are needed to support the algorithm in other languages:

Stop word lists (essential).
Stemming algorithms (desirable but not vital)

It is suggested that a Web search (probably in the appropriate language) may well uncover suitable candidates for both of these.

A more complex problem is the need for a set of pre-classified records for training and testing. Without such a set the machine learning method will not be able to be trained. This would require a set of experts in the appropriate language to provide keywords for records in the suitable language using the appropriate language version of ELSST. This is a time consuming and expert task.

However, the task may be made easier as it observed that there are a significant number of studies in common across archives. If these have been indexed in one language, then it is reasonable to take that set of indexing terms, suitably translated, into the other language's record. This could provide an initial set of indexed documents to start the training process off in the second language. However, case should be taken that this training set is sufficiently large to be useful, and the results are carefully monitored by language experts.

6. Using the Indexing Tool

6.1 Task Scenarios

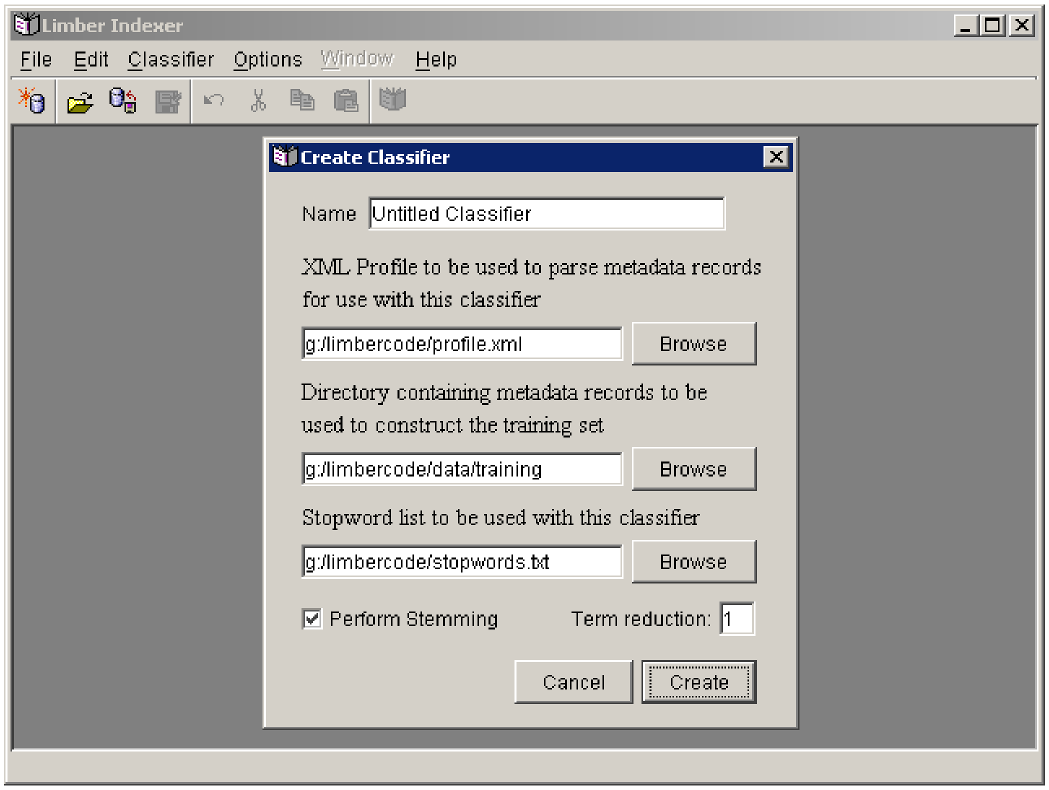
From the user's perspective two goals exist for which scenarios are described: creating a classifier, and classifying a metadata record with that classifier. Additionally, a variety of options are also available for the indexing tool. These are described in the next three sections.

6.1.1 Creating a classifier

The goal of creating a classifier is broken down into 7 interaction dialogue steps between user and system:

Goal - Create a new classifier
- File/New Classifier
- Dialogue Box - options
- Create Button
- Dialogue Box - report number of keywords, testset options
- Test Button
- Dialogue Box - report precision and recall statistics.

Selecting *Menu File/New Classifier* brings up a dialogue box shown below.



Name: Untitled Classifier

XML Profile to be used to parse metadata records for use with this classifier: g:/limbercode/profile.xml

Directory containing metadata records to be used to construct the training set: g:/limbercode/data/training

Stopword list to be used with this classifier: g:/limbercode/stopwords.txt

[✓] Perform Stemming   Term reduction: 1

This requires the user to enter the locations of three files:

XML profile – a script written in XSL defining which parts of the XML document should be used to construct indexes, and which parts should have keywords attached to them.

Directory of Metadata Records – the directory where all the records in the training set are to be found in separate files.

Stopword List – the file to be used as a list of stopwords not to be used in the classification.

Options include:

Perform Stemming – whether words should be stemmed or not.

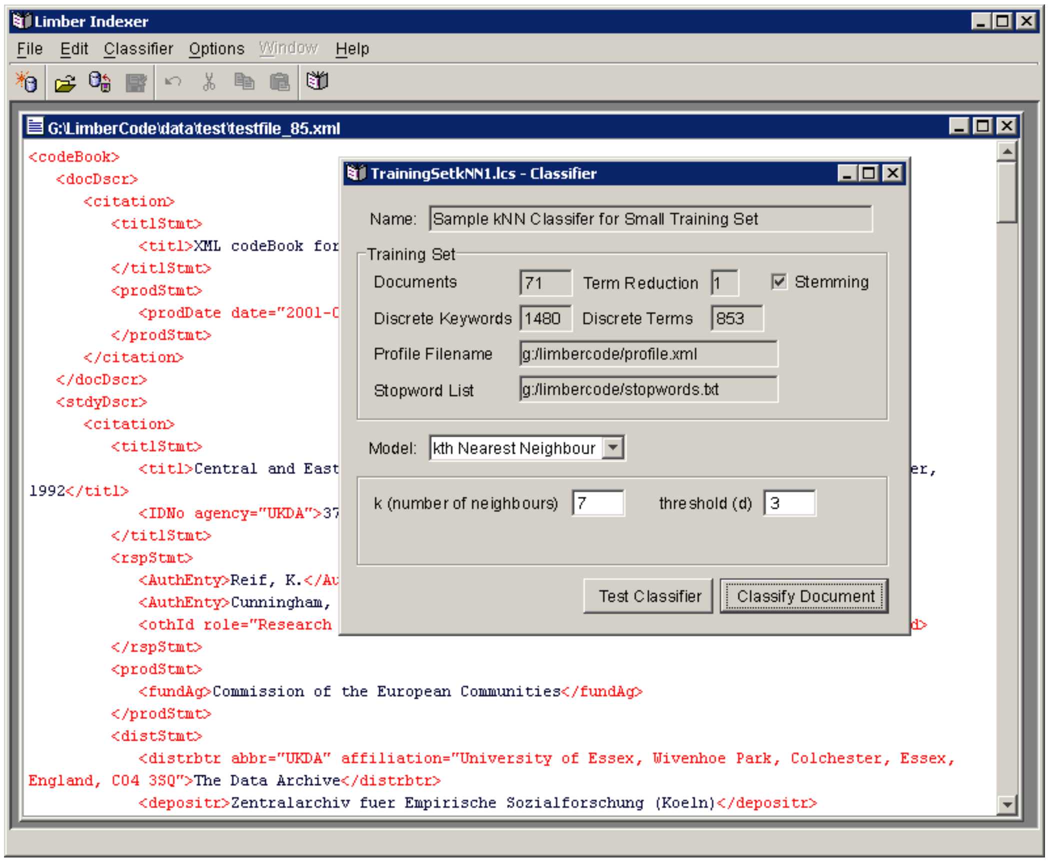
Term Reduction: The distance to be used for reducing terms.

Lastly the new classifier will require a name to be used later to call it.

When these items are completed, the user will select *create* button to create the classifier.

The classifier creation can take 30 minutes to an hour depending on how many files are stored in the directory of metadata records.

When the classifier has been constructed a classifier report dialogue box appears – shown below.


Limber Indexer
File Edit Classifier Options Window Help
G:\LimberCode\data\test\testfile_85.xml
TrainingSetkNN1.lcs - Classifier
Test Classifier
Classify Document

```
<codeBook>
   <docDscr>
      <citation>
         <titlStmt>
            <titl>XML codeBook for
         </titlStmt>
         <prodStmt>
            <prodDate date="2001-0
         </prodStmt>
      </citation>
   </docDscr>
   <stdyDscr>
      <citation>
         <titlStmt>
            <titl>Central and East                                                              er,
1992</titl>
            <IDNo agency="UKDA">37
         </titlStmt>
         <rspStmt>
            <AuthEnty>Reif, K.</Au
            <AuthEnty>Cunningham,
            <othId role="Research                                                              d>
         </rspStmt>
         <prodStmt>
            <fundAg>Commission of the European Communities</fundAg>
         </prodStmt>
         <distStmt>
            <distrbtr abbr="UKDA" affiliation="University of Essex, Wivenhoe Park, Colchester, Essex,
England, C04 3SQ">The Data Archive</distrbtr>
            <depositr>Zentralarchiv fuer Empirische Sozialforschung (Koeln)</depositr>
```

Name: Sample kNN Classifer for Small Training Set

Training Set

Documents	71	Term Reduction	1	✓ Stemming
Discrete Keywords	1480	Discrete Terms	853	
Profile Filename	g:/limbercode/profile.xml			
Stopword List	g:/limbercode/stopwords.txt			

Model: kth Nearest Neighbour

k (number of neighbours) 7 threshold (d) 3

The next step is to test the classifier, by selecting the test classifier button.

This produces a simple report of the performance of the classifier.

6.1.2 *Classifying a metadata record*

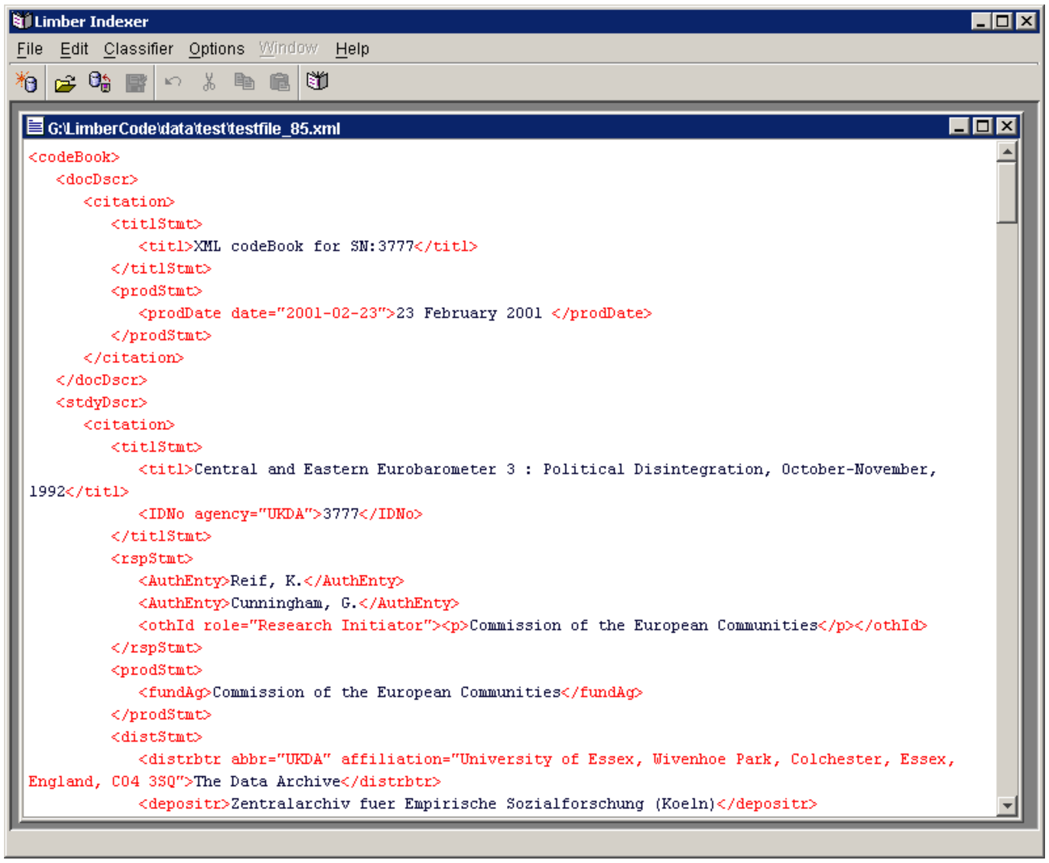
The goal of classifying a metadata record is broken down into 8 interaction dialogue steps between user and system:

Goal - Classify a document

File/ New Document
dialogue box - Browse, write filename, Open button
Classifier/*Name* - select the classifier

dialogue box - set k & n
button - Classify document
classified metadata report - Edit/ Add, Delete, Copy
File/Save document

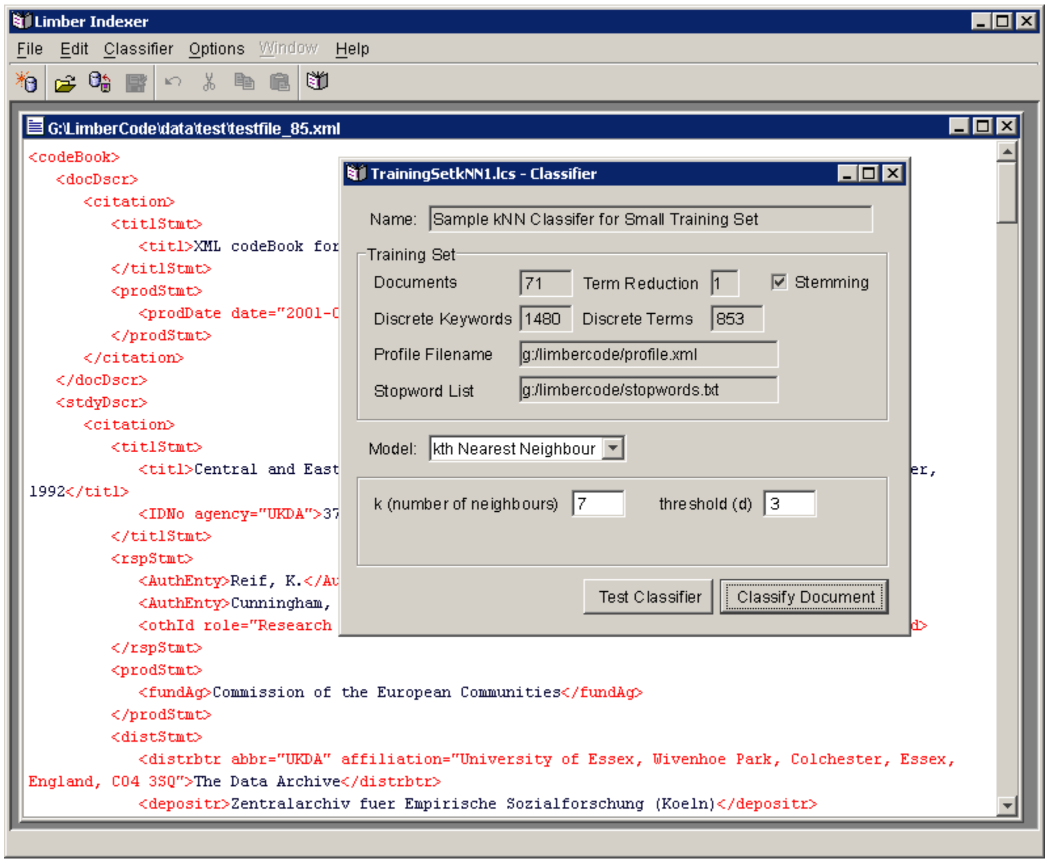
Firstly the document must be opened with the menu item *File/ New Document* which results in the file being presented as shown below:


Limber Indexer
File Edit Classifier Options Window Help
G:\LimberCode\data\test\testfile_85.xml

```
<codeBook>
   <docDscr>
      <citation>
         <titlStmt>
            <titl>XML codeBook for SN:3777</titl>
         </titlStmt>
         <prodStmt>
            <prodDate date="2001-02-23">23 February 2001 </prodDate>
         </prodStmt>
      </citation>
   </docDscr>
   <stdyDscr>
      <citation>
         <titlStmt>
            <titl>Central and Eastern Eurobarometer 3 : Political Disintegration, October-November,
1992</titl>
            <IDNo agency="UKDA">3777</IDNo>
         </titlStmt>
         <rspStmt>
            <AuthEnty>Reif, K.</AuthEnty>
            <AuthEnty>Cunningham, G.</AuthEnty>
            <othId role="Research Initiator"><p>Commission of the European Communities</p></othId>
         </rspStmt>
         <prodStmt>
            <fundAg>Commission of the European Communities</fundAg>
         </prodStmt>
         <distStmt>
            <distrbtr abbr="UKDA" affiliation="University of Essex, Wivenhoe Park, Colchester, Essex,
England, CO4 3SQ">The Data Archive</distrbtr>
            <depositr>Zentralarchiv fuer Empirische Sozialforschung (Koeln)</depositr>
```

In order to load a classifier, the classifier should be selection with the command: Classifier/*Name*

This produces a simple dialogue box stating the classifier to be used – the same dialogue box used when a classifier is produced. Peramtiers such as k & n can be set here before the *Classify Document* button is selected to classify the current document as shown below.


Limber Indexer
File Edit Classifier Options Window Help
G:\LimberCode\data\test\testfile_85.xml
TrainingSetkNN1.lcs - Classifier
Test Classifier
Classify Document

```
<codeBook>
   <docDscr>
      <citation>
         <titlStmt>
            <titl>XML codeBook for
         </titlStmt>
         <prodStmt>
            <prodDate date="2001-C
         </prodStmt>
      </citation>
   </docDscr>
   <stdyDscr>
      <citation>
         <titlStmt>
            <titl>Central and East                                          er,
1992</titl>
            <IDNo agency="UKDA">37
         </titlStmt>
         <rspStmt>
            <AuthEnty>Reif, K.</Au
            <AuthEnty>Cunningham,
            <othId role="Research                                          d>
         </rspStmt>
         <prodStmt>
            <fundAg>Commission of the European Communities</fundAg>
         </prodStmt>
         <distStmt>
            <distrbtr abbr="UKDA" affiliation="University of Essex, Wivenhoe Park, Colchester, Essex,
England, C04 3SQ">The Data Archive</distrbtr>
            <depositr>Zentralarchiv fuer Empirische Sozialforschung (Koeln)</depositr>
```

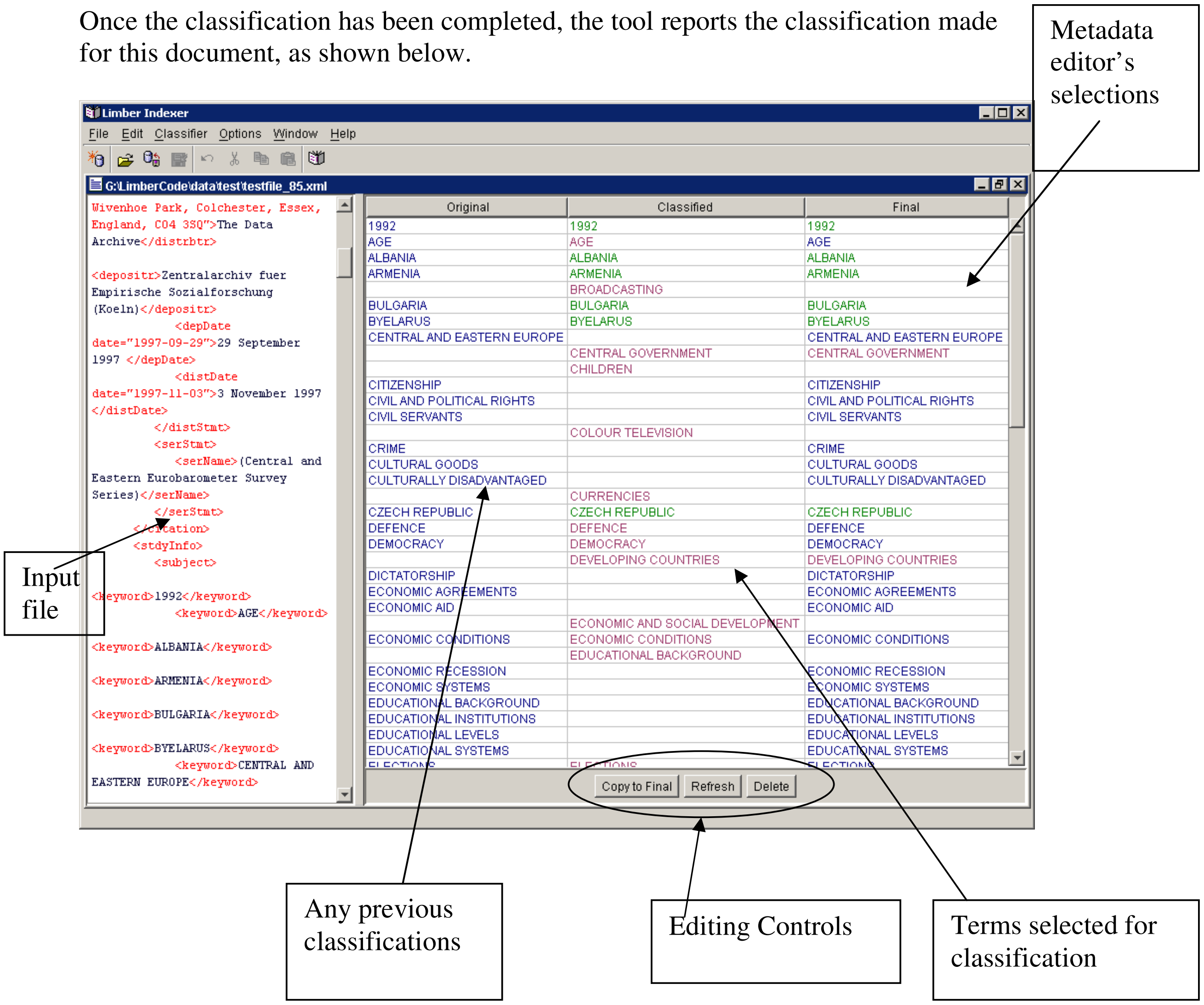
Name: Sample kNN Classifer for Small Training Set

Training Set
Documents: 71 | Term Reduction: 1 | [✓ Stemming]
Discrete Keywords: 1480 | Discrete Terms: 853
Profile Filename: g:/limbercode/profile.xml
Stopword List: g:/limbercode/stopwords.txt

Model: kth Nearest Neighbour

k (number of neighbours): 7 | threshold (d): 3

Once the classification has been completed, the tool reports the classification made for this document, as shown below.


Metadata editor's selections
Input file
Any previous classifications
Editing Controls
Terms selected for classification
Limber Indexer
File Edit Classifier Options Window Help
G:\LimberCode\data\test\testfile_85.xml
Copy to Final
Refresh
Delete

```
Wivenhoe Park, Colchester, Essex,
England, CO4 3SQ">The Data
Archive</distrbtr>

<depositr>Zentralarchiv fuer
Empirische Sozialforschung
(Koeln)</depositr>
            <depDate
date="1997-09-29">29 September
1997 </depDate>
            <distDate
date="1997-11-03">3 November 1997
</distDate>
         </distStmt>
         <serStmt>
            <serName>(Central and
Eastern Eurobarometer Survey
Series)</serName>
         </serStmt>
      </citation>
      <stdyInfo>
         <subject>

<keyword>1992</keyword>
            <keyword>AGE</keyword>

<keyword>ALBANIA</keyword>

<keyword>ARMENIA</keyword>

<keyword>BULGARIA</keyword>

<keyword>BYELARUS</keyword>
            <keyword>CENTRAL AND
EASTERN EUROPE</keyword>
```

Original	Classified	Final
1992	1992	1992
AGE	AGE	AGE
ALBANIA	ALBANIA	ALBANIA
ARMENIA	ARMENIA	ARMENIA
	BROADCASTING	
BULGARIA	BULGARIA	BULGARIA
BYELARUS	BYELARUS	BYELARUS
CENTRAL AND EASTERN EUROPE		CENTRAL AND EASTERN EUROPE
	CENTRAL GOVERNMENT	CENTRAL GOVERNMENT
	CHILDREN	
CITIZENSHIP		CITIZENSHIP
CIVIL AND POLITICAL RIGHTS		CIVIL AND POLITICAL RIGHTS
CIVIL SERVANTS		CIVIL SERVANTS
	COLOUR TELEVISION	
CRIME		CRIME
CULTURAL GOODS		CULTURAL GOODS
CULTURALLY DISADVANTAGED		CULTURALLY DISADVANTAGED
	CURRENCIES	
CZECH REPUBLIC	CZECH REPUBLIC	CZECH REPUBLIC
DEFENCE	DEFENCE	DEFENCE
DEMOCRACY	DEMOCRACY	DEMOCRACY
	DEVELOPING COUNTRIES	DEVELOPING COUNTRIES
DICTATORSHIP		DICTATORSHIP
ECONOMIC AGREEMENTS		ECONOMIC AGREEMENTS
ECONOMIC AID		ECONOMIC AID
	ECONOMIC AND SOCIAL DEVELOPMENT	
ECONOMIC CONDITIONS	ECONOMIC CONDITIONS	ECONOMIC CONDITIONS
	EDUCATIONAL BACKGROUND	
ECONOMIC RECESSION		ECONOMIC RECESSION
ECONOMIC SYSTEMS		ECONOMIC SYSTEMS
EDUCATIONAL BACKGROUND		EDUCATIONAL BACKGROUND
EDUCATIONAL INSTITUTIONS		EDUCATIONAL INSTITUTIONS
EDUCATIONAL LEVELS		EDUCATIONAL LEVELS
EDUCATIONAL SYSTEMS		EDUCATIONAL SYSTEMS

In this display the different colours of text represent:

Blue: keywords found in the original input document

Purple: keywords proposed by the indexing tool based on the model

Green: keywords added by rules (place and people names).

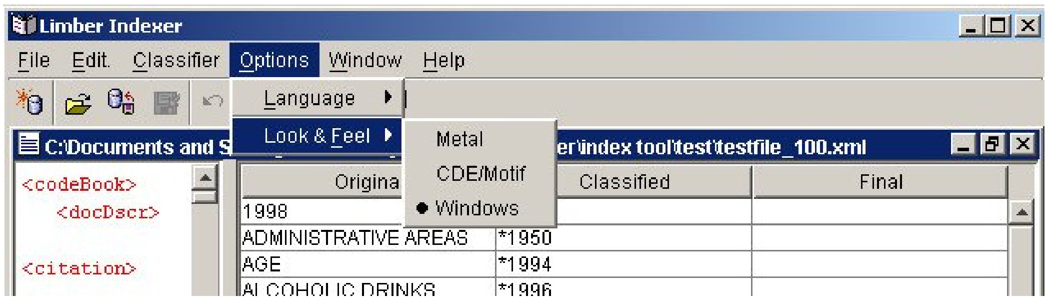
This stage allows the human editor to use their skill to override the suggestions of the tool, or to add to them. This shows that the process is not an automatic, but an assisted indexing of the document.

The human editor can now use the editing controls at the bottom of the panel to move the terms which are judged to be appropriate into the left hand, final, column; or add new terms into this column that were not suggested by the classifier.

6.1.3 Indexing Tool Options

6.1.3.1 Look and Feel Options

The top level Options menu allows the selection of the Look and Feel of the window system to be used for the Indexing Tool. This calls the standard Java Singset2 function to set the look and feel as one of Java, Motif, MS-Windows or Mackintosh. The last two are only available on the appropriate windowing system.


Limber Indexer
File Edit Classifier Options Window Help
Language
Look & Feel
Metal
CDE/Motif
● Windows
C:\Documents and S
er\index tool\test\testfile_100.xml

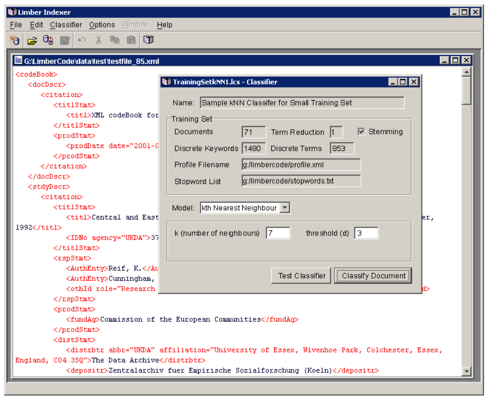
```
<codeBook>
   <docDscr>

<citation>
```

Origina	Classified	Final
1998		
ADMINISTRATIVE AREAS	*1950	
AGE	*1994	
ALCOHOLIC DRINKS	*1996	

The purpose of this option is to allow users to set the look and feel to be one that they are accustomed to; that is as compatible with other tools as possible, and as consistent with their previous experience as possible to reduce planning errors and execution slips when using the tool's user interface.

The two screen images below show the classification dialogue box, and the main window using two different look and feel settings to illustrate the changes set by the option.


Limber Indexer
File Edit Classifier Options Window Help
G:\LimberCode\data\test\testfile_85.xml
TrainingSetkNN1.lcs - Classifier
Test Classifier
Classify Document

```
<codeBook>
   <docDscr>
      <citation>
         <titlStmt>
            <titl>XML codeBook for
         </titlStmt>
         <prodStmt>
            <prodDate date="2001-0
         </prodStmt>
      </citation>
   </docDscr>
   <stdyDscr>
      <citation>
         <titlStmt>
            <titl>Central and East                                   er,
1992</titl>
            <IDNo agency="UKDA">37
         </titlStmt>
         <rspStmt>
            <AuthEnty>Reif, K.</Au
            <AuthEnty>Cunningham,
            <othId role="Research                                    d>
         </rspStmt>
         <prodStmt>
            <fundAg>Commission of the European Communities</fundAg>
         </prodStmt>
         <distStmt>
            <distrbtr abbr="UKDA" affiliation="University of Essex, Wivenhoe Park, Colchester, Essex,
England, CO4 3SQ">The Data Archive</distrbtr>
            <depositr>Zentralarchiv fuer Empirische Sozialforschung (Koeln)</depositr>
```

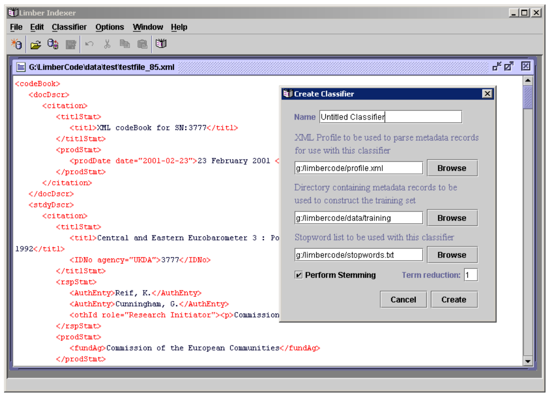
Name: Sample kNN Classifer for Small Training Set

Training Set

Documents	71	Term Reduction	1	✓ Stemming
Discrete Keywords	1480	Discrete Terms	853	
Profile Filename	g:/limbercode/profile.xml			
Stopword List	g:/limbercode/stopwords.txt			

Model: kth Nearest Neighbour

k (number of neighbours) 7 threshold (d) 3


Limber Indexer
File Edit Classifier Options Window Help
G:\LimberCode\data\test\testfile_85.xml
Create Classifier
Browse
Browse
Browse
Cancel
Create

```
<codeBook>
   <docDscr>
      <citation>
         <titlStmt>
            <titl>XML codeBook for SN:3777</titl>
         </titlStmt>
         <prodStmt>
            <prodDate date="2001-02-23">23 February 2001 <
         </prodStmt>
      </citation>
   </docDscr>
   <stdyDscr>
      <citation>
         <titlStmt>
            <titl>Central and Eastern Eurobarometer 3 : Po
1992</titl>
            <IDNo agency="UKDA">3777</IDNo>
         </titlStmt>
         <rspStmt>
            <AuthEnty>Reif, K.</AuthEnty>
            <AuthEnty>Cunningham, G.</AuthEnty>
            <othId role="Research Initiator"><p>Commission
         </rspStmt>
         <prodStmt>
            <fundAg>Commission of the European Communities</fundAg>
         </prodStmt>
```

Name: Untitled Classifier

XML Profile to be used to parse metadata records for use with this classifier

g:/limbercode/profile.xml

Directory containing metadata records to be used to construct the training set

g:/limbercode/data/training

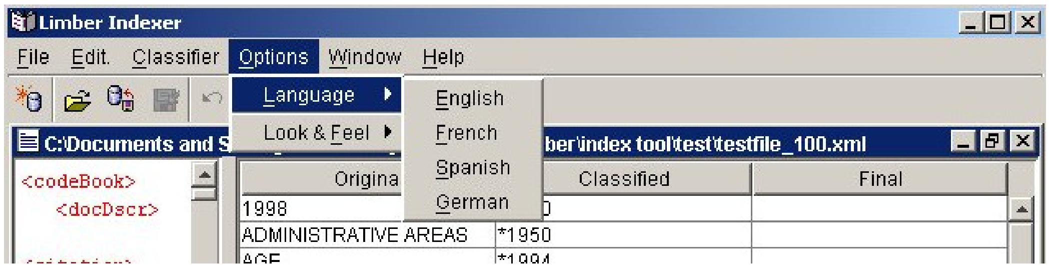
Stopword list to be used with this classifier

g:/limbercode/stopwords.txt

✓ Perform Stemming Term reduction: 1

6.1.3.2 Localisation Options

The menus and dialogue boxes have been designed to be internationalised using the string budling options in Java. Localisation options are available on the menu to set the language, number display, currency etc.. to one compatible with the end user.

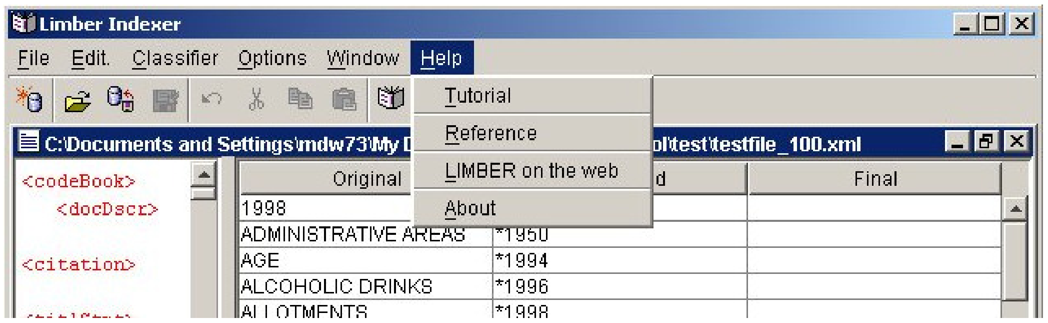


New localisation files can be added for novel languages/cultures as required.

6.1.3.3 Help

The help menu item on the main window provides four options:

1) About box stating copyright information
2) Pointer to the LIMBER project pages on the web for any latest information
3) A tutorial on using the Indexing tool – effectively the two preceeding sections of this document.
4) A reference document to the tool – effectively this section on LIMBER in this document.



6.2 Installation and Use of the Indexing Tool

The indexing tool is distributed as a Java Archive, in a Zip archive file.

The Java Archive should be unzipped into a directory. The dependency is on the Java API for XML parsing. This archive is included in many distributions of Java, but if not it can be obtained from: http://java.sun.com/xml/jaxp/index.html and included in the lib/ext directory of your JDK installation.

Double clicking on the archive will activate the programme.

6.3 Conclusion and Future Work for the Indexing Tool

The indexing tool currently does allow documents to be indexed but it requires considerable use before we can be confident about guidance on the values to use for K & N, and for the term reduction values.

Stop lists are provided in several languages (English, French, Spanish, German) but stemming algorithm is only included for English. Language simplification mechanisms for other languages should be included.

In theory the indexing tool can be used to index across languages given a common set of pre-marked up metadata files for the learning phase. This option needs further investigation and practical use before it can be applied by real users.

References.

Kjersti Aas, and Line Eikvil, **Text categorisation - A survey** *NR report no. 941 ISBN:* 82-539-0425-8 June, 1999, http://www.nr.no/research/samba/tm_survey.ps

C. Peters, (Ed.) Cross-Language Information Retrieval and Evaluation, LNCS 2069, Berlin:Springer, 2001.

M.F.Porter. **An algorithm for suffix stripping,** *Program*, 14 no. 3, pp 130-137, July 1980. http://www.omsee.com/developer/docs/porterstem.html

Fabrizio Sebastiani, **A Tutorial on Automated Text Categorisation.** In Analia Amandi and Alejandro Zunino (eds.), *Proceedings of ASAI-99, 1st Argentinian Symposium on Artificial Intelligence,* Buenos Aires, AR, pp. 7-35, 1999

Fabrizio Sebastiani **Machine Learning in Automated Text Categorisation.** Revised version of Technical Report IEI-B4-31-1999, Istituto di Elaborazione dell'Informazione, Consiglio Nazionale delle Ricerche, Pisa, IT, 2001. http://faure.iei.pi.cnr.it/~fabrizio/Publications/ACMCS01/ACMCS01.pdf Submitted for publication to *ACM Computing Surveys*.

Y. Yang, **An Evaluation of Statistical Approaches to Text Categorization**, Technical Report CMU-CS-97-127, Computer Science Department, Carnegie Mellon University, 1997.