Lexical Entry Tool (Software)
and
Accompanying Documentation

Deliverable D4.1
ISLE Computational Lexicon Working Group

Authors

Nuria Bel[1], *Marta Villegas*[1], *Montserrat Marimon*[1]

1: gilcUB, Barcelona, SPAIN

IST 1999-10647 ISLE
Deliverable Identification Sheet

Project ref. no.	**IST 1999-10647 ISLE**
Project acronym	**ISLE**
Project full title	**International Standards for Language Engineering**

Security (distribution level)	**Restricted**
Document name	*D4.1 Lexical Entry Tool (software) and Accompanying Documentation Prototype*
Type	*Deliverable*
Status & version	*Final Version*
Number of pages	*40*
WP contributing to the deliverable	
Task responsible	*gilcUB*
Other contributors	
Author(s)	*N. Bel, M. Villegas, M. Marimon - gilcUB*

DOCUMENT EVOLUTION (optional)

Version	Date	Status	Notes
0.2	18/02/2003	Final	

TABLE OF CONTENTS

0. EXECUTIVE SUMMARY

This document describes (i) the lexicographic station development platform used to automatically generate a prototype tool out off ISLE[1] guidelines formally implemented in a DTD, (ii) an actual implementation of the ISLE guidelines expressed in MILE, (iii) the use of the lexicographic station development platform for generating a prototype lexical tool for MILE/ISLE guidelines and the functionalities and (iv) the user manual of the resulting prototype lexicographical station.

The aim of ISLE is to develop, disseminate and promote de facto HLT standards and guidelines for language resources, tools and products within an international framework, in the context of the EU-US International Research Cooperation initiative.

In the 'multilingual computational lexicon' area, ISLE has extended EAGLES[2] work on lexical semantics to design standards for multilingual lexicons. The central outcome of ISLE is to define a general schema for multilingual lexical entry (MILE) that is to be the basis for a standard framework for multilingual computational lexicons. In addition, ISLE is to develop a prototype tool to assist the development of multilingual lexical resources in MILE format.

The aim of this prototype tool is to

(i) exemplify and disseminate the MILE entry using an actual model with already existing monolingual data,
(ii) assist the development of multilingual lexical resources following MILE schema,
(iii) make extensive use of already existing PAROLE and SIMPLE lexicons and
(iv) to eventually test the goodness of the guidelines by using a real scenario.

Three aspects crucially determined the definition of the lexicographic station development platform we are describing here:

(a) MILE is built as an additional layer on top of monolingual descriptions. In most cases, these monolingual layers already exist and need to be reused.
(b) MILE is a general schema liable to be customized according to in-house needs in real scenarios.
(c) the definition of the prototype tool an the definition of MILE itself were parallel tasks. This means we had not a final model while developing the tool.

This situation led us to define a lexicographic station development platform that guarantee the portability of the final prototype. The lexicographical station development platform, has been designed as a Tool builder which parses any DTD describing an Entity Relationship model in order to automatically (i) map the DTD into a relational dB and (ii) build up a user-friendly interface able to cover the most common requirements of a lexicographic station and (iii) to exemplify, test and validate the goodness of the MILE model in a real scenario, that is, reusing already existing monolingual resources such as PAROLE and SIMPLE lexicons.

[1] International Standards Lexical Encoding. See Ref??
[2] 'Expert Advisory Group in Lexical Standards' Ref?? .

This strategy will guarantee the portability of the final prototype to the final specifications, the portability of already existing resources (i.e. reusability of already existing monolingual resources) and, finally, the portability to specific applications thus allowing for customization.

This project lays on the idea that the information contained in a DTD which describes a conceptual model expressed in terms of Entity-Relationship Model can be used to automatically build up a relational dB. This ' DTD approach' allows easy customization. The user no longer has to accommodate to the structure and insights of the lexicographic tool but rather the tool accommodates to the requirements and idiosyncrasies of the user needs.

Since the resulting prototype tool is seen as an exemplification, it has been designed as a self-explaining tool and, therefore, the user can consult the mappings between the dB and the DTD and is provided with a set of browsing facilities that allow to understand the model behind.

From now on this document is organized as follows.

Section 1 describes the general architecture of the lexicographical station platform development.

Section 2 describes the generation module and the ‘load sgml data’ module. As we will see, the generation module is responsible to create a relational database out of a DTD. The ‘Load sgml data’ module guarantees that already existing lexical resources can be directly loaded into the database. Since, multilingual correspondences defined in MILE relay on monolingual descriptions which, in most cases, may be already existing resources, this ‘load sgml’ module becomes crucial.

Section 3 deals with the customization module of the prototype tool.

Section 4 describes the general architecture of core interface module of the prototype tool.

Section 5 describes an actual implementation of the MILE (Multilingual ISLE Lexical Entry) module.

Section 6 contains the software specifications for the resulting prototype tool.

Section 7 describes the main functionalities and user’s manual for the prototype tool.

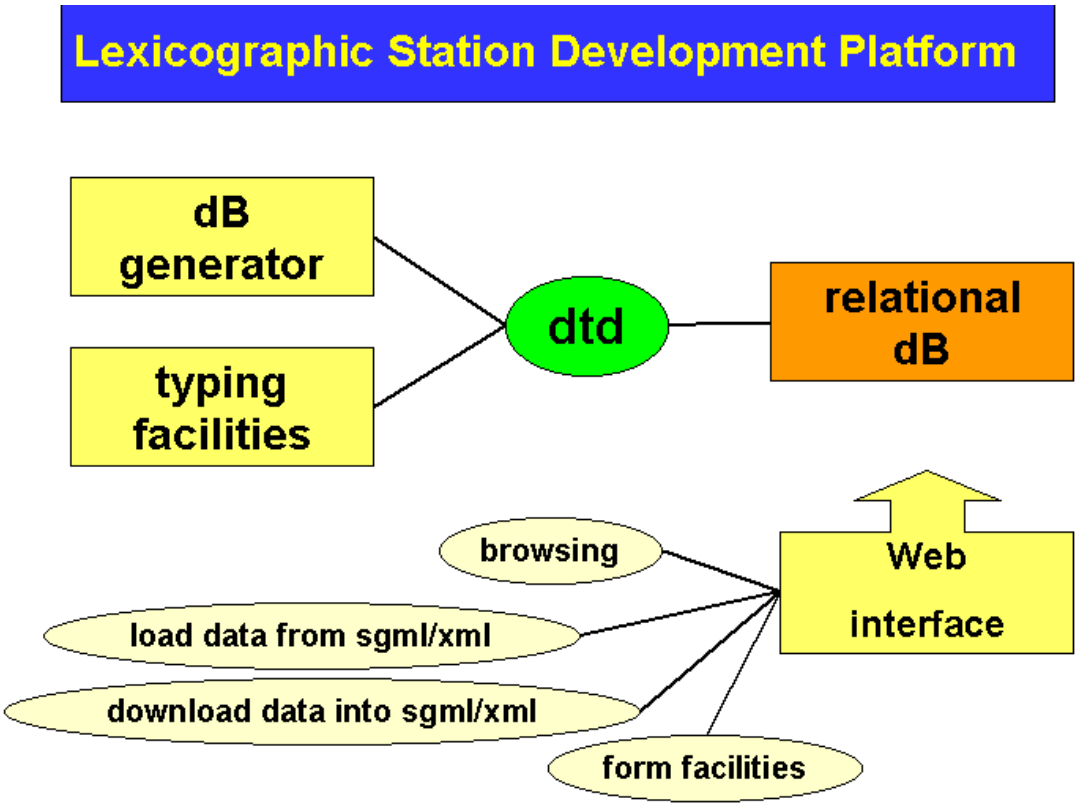
Section 8 reports our experience using the prototype tool. As reported there, we have used the prototype tool to load already existing monolingual sgml data and to encode new entries.

General architecture

The lexicographical station development platform is a prototype tool generator that reads and parses a sgml/xml DTD and generates a relational data base that can be managed with a core web dB interface.

The lexicographical station development platform guarantees that already existing monolingual resources expressed in sgml/xml can be easily reused and ported by and to MILE.

Basically, the lexicographic station development platform includes a generation module, a customization module and a core interface module as exemplified below:



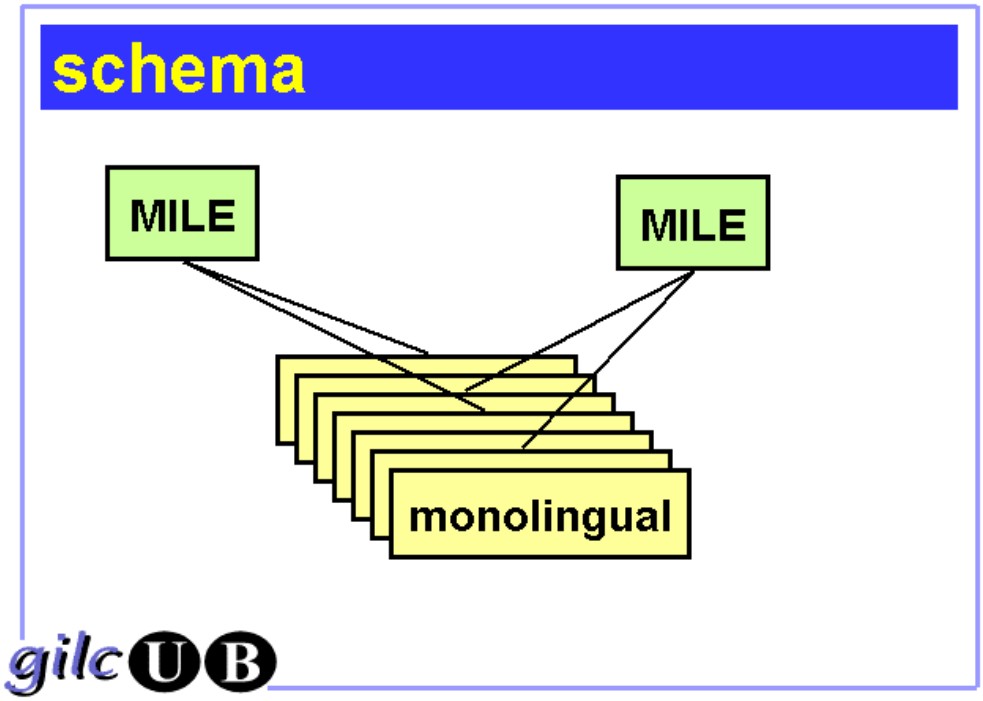
The **generation module** automatically generates a relational dB out off a DTD. The project benefits from the fact that a conceptual model expressed in terms of Entity-Relationship model can be easily mapped into a relational dB.

The **customisation module** allows to modify certain aspects of the dB at the time that allows overcoming some of the well known shortcomings of sgml DTDs such as typed references and type declaration.

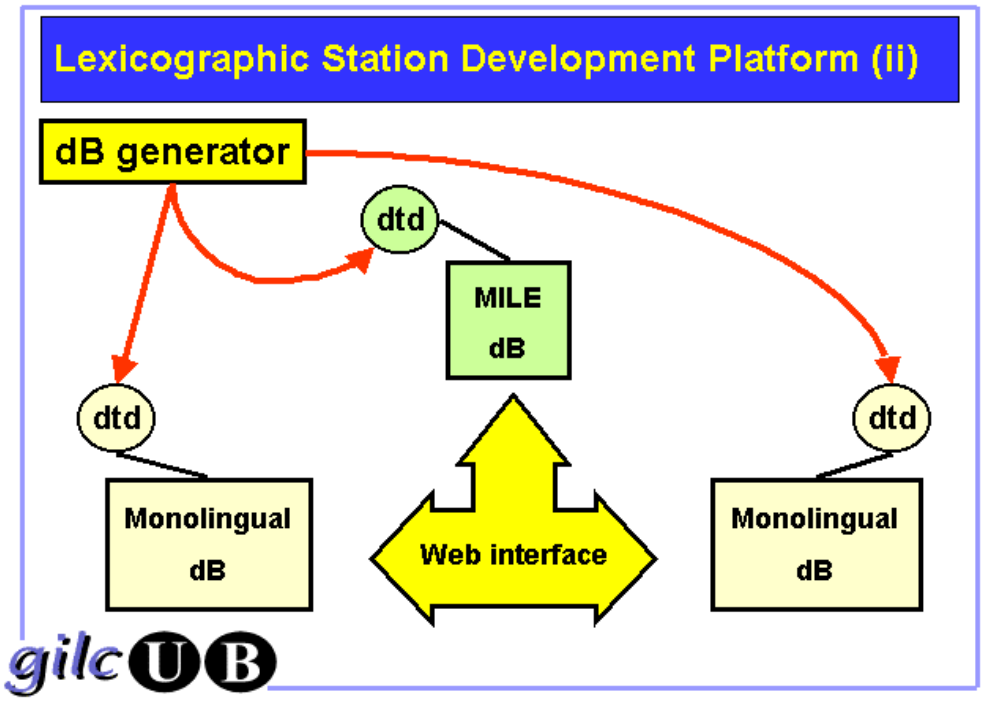
The **core interface module** consists of a series of scripts that allow managing the dB with a friendly interface. Although user requirements differ from site to site according to in-house needs, the tool comes equipped with a set of basic functionalities. Our experience in past lexicographic projects led us to define an accurate list of requirements which include (i) query and browsing facilities, (ii) import, export and migration of data, (iii) easy encoding of new data (iv) test and validation of both the data and the model, (v) customization facilities, and (vi) lexicographic tools such as type definition, class extraction and statistical facilities. As in the case of the

generation module, this core interface module operates upon the model expressed in the DTD in order to make the necessary calculations to access, manipulate and display data from relevant tables.

ISLE defines the multilingual layer as an additional layer on top of the monolingual ones. Thus, whereas monolingual layers collect morphological, syntactic and semantic information needed to describe monolingual lexicons, the multilingual layer defines correspondence objects that describe relations between monolingual representations. This approach guarantees the independence of monolingual descriptions at the time that allows the maximum degree of flexibility. This structure can be represented as follows:



As we can see from figure above, the dB generator needs to generate at least two monolingual databases and one bilingual database and the web interface needs to address three different databases. The overall architecture of the system can be represented as follows:



The generation module

The generation of the data base is done by means of a perl script that, making extensive use of the perlSGML module, reads the DTD and generates tow output scripts:

- **BuildDB**: is an output file containing the relevant 'CREATE TABLE' instructions. This output file can be edited to make the desired modifications (shorten or length the fields, delete tables,...) and can be executed by MySQL by typing 'mysql> *data_base_name* < *script.file*'

- **LoadSGMLData**: is a perl script that reads an sgml data file and distributes the data it contains into a series of tabular files which correspond to the tables in the dB. TabularDTD is sensitive to the hierarchic relations between sgml elements in order to keep track of the foreign keys involved in each content element (see section 3.2 for further details). This script also generates the relevant 'INSERT' statements and is responsible of loading the tabular files in the corresponding tables.

Tables definition

BuilDB reads the DTD looking for all elements and classifies them into ***main*** or ***content*** elements. ***Main elements*** are top elements having an ID-type attribute. ***Content elements*** are embedded elements without an ID-typed attribute. For each main element, *BuildDB* creates a corresponding ***main table***. Two additional type of tables can be also created. These are ***content tables*** and ***list tables***. Content tables are created whenever an element has a content element. List tables are created whenever an element includes an IDREFS-typed attribute (that is, an attribute valued as a list of IDs).

The name of the tables derive from the name of the elements thus, main tables have the same name as the corresponding main object, content tables names result of the concatenation of the parent and the content element and, finally, list table names result from the concatenation of the element and the IDREFS-type attribute name. This can be see in Table 1 below.

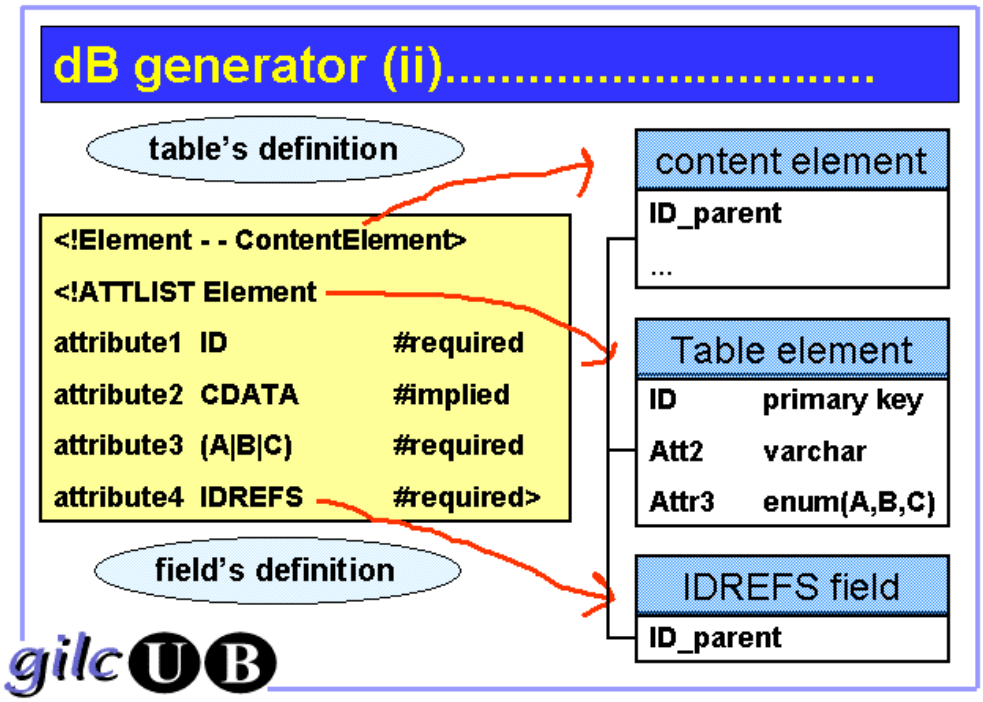


Table 1: Tables definitions

Fields definition

Attributes in the element' s ATTLIST description of elements are directly mapped into fields in the corresponding table definition according to the following criteria.

sgml declaration	sql definition	addiditional list table
ID	varchar(80)	
IDREFS	varchar(80)	list
NUMBER	integer(10)	
NUMBERS	integer(10)	
CDATA	varchar(100)	
NUTOKEN	varchar(80)	

NUTOKENS	varchar(80)	list
NAME	varchar(80)	
NAMES	varchar(80)	list

Content tables include two additional fields: one corresponds to the table ID and is defined as an auto increment primary key; the other serves to relate the content element with the relevant parent element and acts as foreign key.

List tables serve to encode list-typed attributes. They include two fields which are defined as primary keys. One is defined as 'id_parent' and serves to indicate the element containing the list-typed attribute. The other is defined as 'id_attibute' and serves to indicate the attribute itself. In the table above, we can see the kind of attributes that generates a list table.

In the following figures we exemplify the mapping of a given element Element as described in figure 1. In figure 2, we can see how the attributes of 'Element' are mapped into table's fields. Figure 3 describes the mapping of a content element, in this case the embedded ConentElement generates a corresponding table which includes a primary key attribute defined as autoincrement and a foreign key attribute that serves to relate the content element with its parent element. Finally, figure 4 describes the mapping of an IDREFS-typed attribute into a list table. In this case, the table consists only of two attributes defined as multiple primary keys. These keys, serve to relate the object with each of the values of its list-typed attribute:

```
<!ELEMENT Element –0- ContentElement>
<!ATTLIST
        id              ID          #REQUIRED
        attData         CDATA       #IMPLIED
        attEnum         (A|B)       A
        attIDref        IDREF       #IMPLIED
        attIDrefs       IDREFS      #IMPLIED>
```

Figure 1. sgml description for Element

Field	Type	Null	Key	Default	Extra
Id	Varchar(5)		PRI		
AttData	Varchar	YES		NULL	
AttEnum	Enum(A|B)			A	
AttIdRef	Varchar(5)	YES	MUL	NULL	

Figure 2. Main table definition for Element

Field	Type	Null	Key	Default	Extra
Id	Varchar(5)		PRI	0	Auto increment
Id-parent	Varchar(5)		MUL		
….	…	…	…	…	…

…	…	…	…	…	…

Figure 3. Content table definition for ContentElement

Field	Type	Null	Key	Default	Extra
Id	Varchar(5)		PRI		
Id-parent	Varchar(5)		PRI		

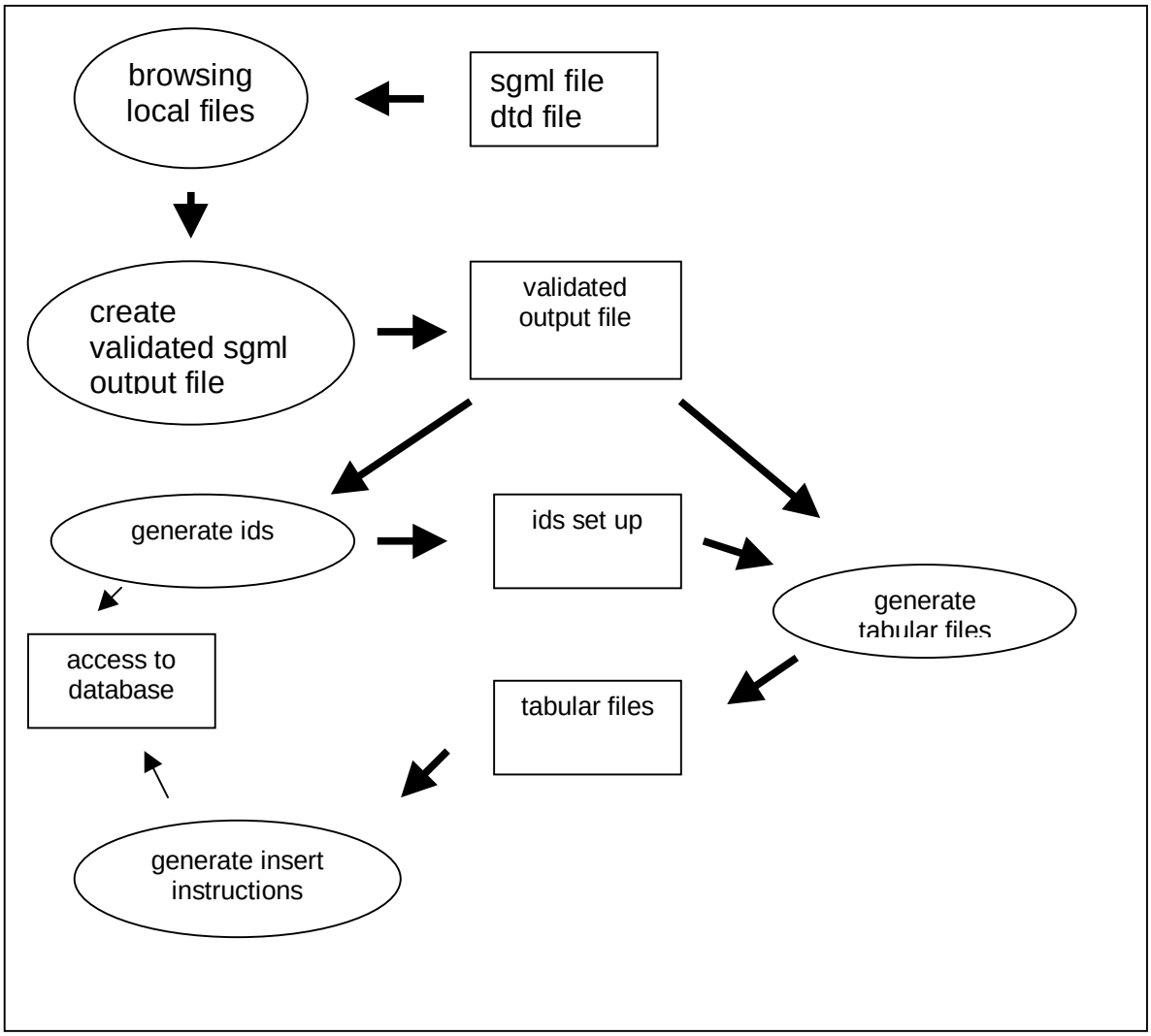
Figure 4. List table definition for AttIrefs attribute

loadSGMLData

Load SGMLdata is responsible for loading data into the database from external sgml data files. This module is crucial as it ensures that already existing resources can be easily ported to the database.

LoadSGMLdata is included into the prototype tool as an option. The system allows browsing local files so that the user can select the relevant DTD and data file to be loaded. The file is parsed using nsgmls parser in order to validate the data and generate the output file.

Once the data file is validated, the system works with the resulting output file. This allows overcoming format and layout aspects that may vary from file to file. Basically the system performs three main actions: (i) for each kind of element in data file, the system checks whether the database contains some data in order to set up the auto increment ids; (ii) all data in the output parsed file is allocated in temporal tabular files, each file contains data for a given type of element (iii) finally, the relevant insert instructions are generated taking into account the set of elements included in data file. The overall procedure can be represented as follows:



Customisation module

In order to overcome some of the well-known shortcomings of DTDs (typed references, type declaration, inheritance...) the prototype includes a customization module.

This customization module serves a double purpose. On the one hand, it allows expressing type constraints that cannot be expressed in sgml DTDs. These constraints will ensure the goodness of the model. On the other hand, it becomes crucial to define the 'domain' of a given element. In any DTD describing an entity relationship model, relations among elements can be established as 'vertical' or 'horizontal' relations. Vertical relations are the standard hierarchical relations between an element and its content elements. Horizontal relations are those established by IDREF or IDREFS typed attributes that serve to relate a given element with some other element of the model. Both, vertical and horizontal relations between elements define the domain or scope of an element. In the following example we describe the domain of a given element Element 2 containing one IDREF attribute typed as element 5. In this example, the domain for our Element 2 includes all nodes dominated by Element 2 plus Element 5 domain

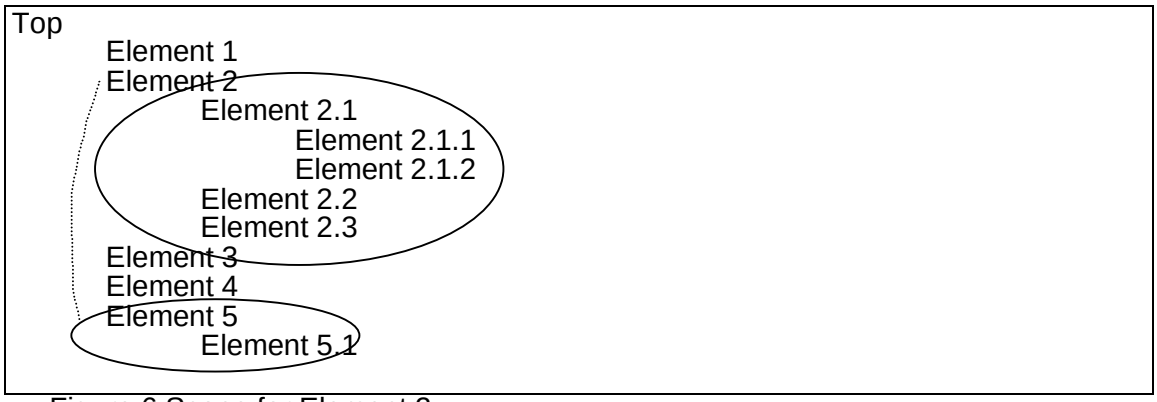


Figure 6 Scope for Element 2

As we will see in next section, the prototype tool comes equipped with some basic functionalities. These functionalities are better tuned if typed references are explicitly established in this customization module.

Typed references are stored in an additional table named 'relations' Essentially, this table collects the element/table containing an IDREF(S) attribute, the attribute itself and the allowed element/table:

table	**attribute**	**related table**
tableA	attributeA	elementB

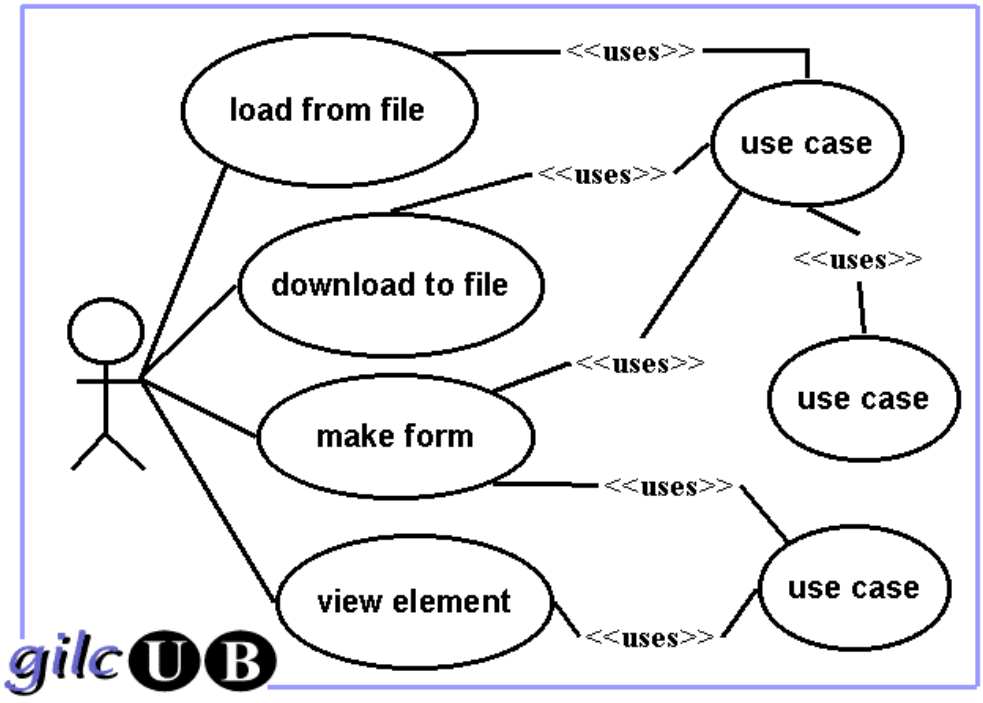
In this example the system knows that the attribute 'attriibuteA' of the element/table 'tableA' can only take as value Ids of elements that belong to 'elementB' type.

Core web dB interface

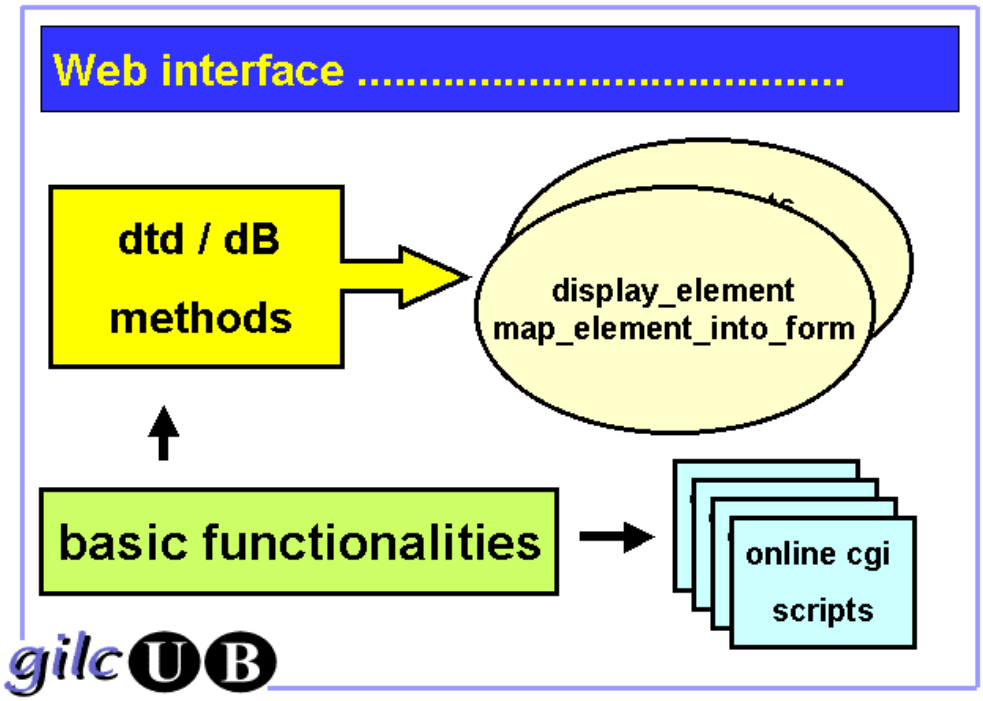
Besides tables definition described in previous section, the system provides with a user interface able to manage the dB in a friendly and explanatory fashion.

The aim of the tool is to provide with a minimum set of build in functionalities that cover the most common user's requirements of a prototypical lexicographic station. The explanatory and dissemination purpose of ISLE project, lead us to additionally include a number of functionalities that serve to know and understand the resulting prototype tool.

All this is to be achieved without facing lexicographers with the technicalities of a dB. Lexicographers are only be required to know the model expressed in the DTD and, therefore, they operate upon the elements defined in the DTD. It is the system that makes the necessary calculations to access and manipulate data from the relevant tables.
The prototype tool, therefore, is designed as DTD dependant rather than dB dependant. Thus, the system includes a good number scripts that taking the DTD structure as input make the necessary calculations to operate on the relevant tables in the dB. Essentially these facilities include (i) loading and downloading data from and into sgml files, (ii) making forms to manage the dB and (iii) learning about the model.



The schema above has been implemented as a library of dtd/dB basic methods that make extensive use of a set of basic functionalities.



download data in sgml/xml fashion. The user is given a tree representation of the DTD and selects one element. The system, then makes the necessary calculations to extract data in sgml/xml format for the desired element.

define forms to extract or load data. The system allows defining online forms to manage the database. The first step in this process is to define the domain of the form. This is the point where the customization process explained above becomes crucial. The user selects the top most elementhe/shewants to include in the form. The system calculates the domain of the selected element by taking into account the horizontal and vertical relations it participates in. Once this is done, the system displays a form with the relevant fields. Fields in the form are defined following attribute's definition in the DTD. Thus, CDATA attributes translate into text fields, ENUM attributes translate into pop-up fields, customized IDREF attributes translate into pop-up menus, and IDREFS attributes translate into multi valued scrolling list fields. Once the user has filled in the desired data, the tool makes the necessary calculations to build up the relevant sql query.

Browsing the data and the model. The tool contains a good number of facilities to browse both the data and the model and its mapping into the database. The prototype allows the user to see the data in a DTD fashion and benefits from the fact that it knows the relational component of the database since this is formally expressed in the DTD.

As already mentioned before, all functionalities above are DTD dependant. This means that html code is generated taking into account the information in the DTD. That is, the system reads the DTD in order to know (i) what kind of elements/tables are involved in each action the user wants to perform, (ii) how are these elements tables described and (ii) what sort of relations these elements/tables are involved in. This strategy guarantees that the web dB interface can be used independently of the dB behind.

MILE module

As already mentioned before, ISLE defines the multilingual layer as an additional dimension on top of the monolingual ones. Thus, whereas monolingual layers collect morphological, syntactic and semantic information needed to describe monolingual lexicons, the multilingual layer defines correspondence objects that describe relations between monolingual representations. These approach guarantees the independence of monolingual descriptions at the time that allows the maximum degree of flexibility and consistency in reusing existing monolingual resources to build new bilingual lexicons.

Bilingual correspondences between source and target unit elements can be rather complex and may involve different transfer conditions. In these cases, the bilingual layer allows to establish *tests* and/or *actions* upon monolingual descriptions in source and target lexicons respectively. Tests and actions are constraints or enrichments on monolingual descriptions needed only when moving from one language to another. More exactly: tests specify a condition in source language under which a given translation is valid; and, actions specify a condition in the target language under which a given translation is valid.

These transfer conditions include semantic transfer conditions and syntactic transfer conditions which can be quickly summarized as follows:

Semantic transfer conditions:

- Argument correspondences between source and target predicates.
- Addition of semantic feature(s) to source or target SemUs.
- Addition of semantic feature(s) to an argument of source or target predicate.

- **Syntactic transfer conditions:**

- Constrain the head of the syntactic description by adding syntactic or semantic features.
- Link source and target positions (i.e. syntactic arguments)
- Adding a syntactic positions to source or target syntactic descriptions.
- Changing the optionality status of a given syntactic position.
- Prohibit the realization of a given syntagma in a given syntactic position.
- Adding semantic or syntactic features to syntagmas filling a given syntactic position.
- Lexicalizing the syntagma filling a given syntactic position.

The kind of tests and actions involved in each correspondence depends on the words involved in each case and on the kind of information included in both source and target lexicons. More crucially, the set of transfer conditions involved in a given bilingual correspondence operates upon descriptive elements that, in most cases, vary from unit to unit.

This scenario makes it impossible to define static fixed forms (or templates) to encode bilingual correspondences. Notice that the number and kind of transfer conditions, and the number and kind of objects these transfer conditions apply on, will change from correspondence to correspondence depending on the kind of monolingual descriptions we are trying to link.

The complex nature of bilingual correspondences led us to define MILE module as an object and the list of admissible transfer conditions as a set of methods that enlarge the initial MILE object in order to collect the desired information.

In the simplest case, MILE object establishes binary relation between a source semantic unit and a target semantic unit. In the most complex case, this object is enriched with a set of relevant constraints. In the following lines we describe in much more details the MILE object

Preliminaries

- the bilingual layer is a set correspondences between semus of different monolingual lexicons.
- the syntactic descriptions of involved semus belong to monolingual lexicons. Thus, whenever a correspondence between two semus is established at the bilingual layer, by default, all syntactic information from monolingual layers is ' inherited' :
- The bilingual layer allows establishing **TEST & ACTIONS** upon these inherited syntactic descriptions in source and target lexicon (respectively).

  TESTS & ACTIONS are constraints/enrichements on monolingual descriptions needed only when moving from one specific language to another. More exactly:

  TESTS specify a condition in the source language under which a given translationis valid.

  ACTIONs specify a condition in the target language under which a given translation is valid.

Formal aspects

Bilingual correspondences can be typed as follows:

- simple correspondences: when no transfer conditions are involved. They can be further specified or classified as fully equivalent or partial equivalent,
- complex correspondences: when transfer conditions are involved. Transfer conditions include: Syntactic transfer conditions and Semantic transfer conditions.

In the following we describe how the model formally deals with simple and complex correspondences.

Simple correspondences

Simple correspondences relate one source semu with one target semu without implying transfer conditions, additional information can be included in order to further specify the correspondence.

Simple correspondences are formally expressed as CorrespMultUsem elements defined as follows:

CorrespMultUsem	• commentaire • id • pointdevue • statutcorr • usemlangue1

	• usemlangue2

Complex correspondences

Complex correspondences are formally expressed as CorrespMultUsem or CorrespMultUsynUsem elements according to the kind of transfer conditions involved.

Essentially, transfer conditions include semantic transfer conditions and syntactic transfer conditions defined as follows

Semantic Transfer conditions include:

- argument correspondence (see argument matching)
- addition of semantic features (see adding semantic features)
- addition of semantic features to arguments (see adding sem feat to arguments)

The resulting CorrespMultUsem element will derive from the combination of relevant semantic transfer conditions involved in each semu-semu correspondence.

CorrespMultUsem	• commentaire • id • pointdevue • statutcorr • usemlangue1 • usemlangue2 • ajoutetraitsemlangue1 • ajoutetraitsemlangue2 • modifieurlangue1l • modifieurlangue2l
	correspmultargarg • cheminarglangue1 • cheminarglangue2 • commentaire • informearglangue1l • informearglangue2l

syntactic transfer conditions include:

- constraint the head (see constraint head)
- matching positions (see match positions)
- addition of a syntactic position (see add position)
- constraint a position by
    - changing is optionality status' (see constraint optionality)
    - prohibit the realization of a syntagma (see prohibit syntagma)
    - lexicalizing its realization (see lexicalize position)
    - adding semantic feature(s) to a syntagma (see add sem feat syntagma)
    - adding syntactic feature(s) to syntagma (see add synt feat syntagma)

The resulting CorrespMultUsyn element will result from the combination of relevant syntactic transfer conditions involved:

CorrespMultUsyn	• commentaire • id • pointdevue • statutcorr • usynlangue1 • usynlangue2 • constrdescrusyn1 • constrdescrusyn2 • reecritlexicalisesyntl1 • reecritlexicalisesyntl2 • ajoutepositionusynl1 • ajoutepositionusynl1
	correspMultPosPos **cheminPosLangue1** **cheminPosLangue2**

Bilingual correspondences may involve semantic and syntactic transfer conditions. In this case, these are expressed via CorrespUsynUsem objects which include semantic transfer conditions and syntactic transfer conditions defined as follows:

correspmultUsynUsem	• commentaire • id • pointdevue • statutcorr • correspmultusyn (see above) • correspmultusem (see above)

Semantic Transfer Conditions

argument matching

This allows establishing correspondences between source/target arguments and to restrict/enrich the semantic description of involved arguments. Two methods are foreseen:

- match source/target arguments
- match & specify source/target arguments

CorrespArgArg	• sourceaccesspath • targetaccesspath

CorrespArgArg	• sourceaccesspath • targetaccesspath • sourcearginforml • targetarginforml

addition of semantic feature(s) to argument

This allows to select and specify an argument by adding semantic features.

&add_sem_feature_argument($predicate,$argument_number,@sem_features)

Modifieur	• commentaire • id • pred = $predicate • cheminargmodifie = $argument_number
	SelectAndSpecifyArg • accesspath = $argument_number • informargl = @sem_features

addition of semantic feature(s)

This allows adding semantic features to source/target semus.

&add_sem_feature_semu($semu,@sem_features)

Syntactic Transfer Conditions

position matching

Allows establishing correspondences between syntactic positions when there is no isomorphic mapping. (a simplified version of the model is given here: matching between added positions are not included)

possible examples: like(eng) vs. gustar(sp)

correspMultPosPos	**cheminPosLangue1** **WayToPosition** • targetposition **cheminPosLangue2** **WayToPosition** • targetposition

constraint head

Allows to constraint the Self element of the syntactic description assigned to semu. As in most cases the Self element corresponds to the head of the construction we call this ' constraint head' .

DescriptionConstraint	• id • comment • example **IntervConsConstraint** addsyntfeaturel = @semfeat

addition of syntactic position

This allows adding a syntactic position to either the source or target syntactic description. Since positions include adjuncts, modifiers can be also added if required.

These new position elements may already exist in the monolingual lexicon or not. In the former case, the user merely needs to select the relevant one. In the later case, the user has to defined a new one. In any case, all position elements belong to monolingual lexicons, as they are the inventory of admissible positions for a given language. The difference is that some positions are only ' relevant' for translation purposes.

constraint position optionality status

This allows changing the optionality status of a given position:

&constraint_optionality($description,$construction,$position,$status)

DescriptionConstraint	• id • comment • example **ConstructionConstraint** **PositionConstraintl** • optionalitymodification

prohibit a syntagma

This allows to prohibit the realization of a certain syntagma in a given position.

DescriptionConstraint	• id • comment • example **ConstructionConstraint** **PositionConstraintl** **SyntagmaConstraint**

constraint position lexicalization

This allows to constraint the lexicalization of a given position. In the model, lexicalixations are defined in terms of LexFeatures. LexFeatures allow specifying the lexicalization of a syntactic leaf (or of the head of a phrase). The lexicographer is free to refer to a morphological unit or to a string corresponding to the graphical form of the lexical unit.

Lexicalization of position only applies for existing positions. Added positions (see above) can be defined as lexicalized if required.

&lexicalize_position($description,$construction,$position,$syntagma,$lexicalize)

DescriptionConstraint	• id • comment • example **ConstructionConstraint** **PositionConstraintl** **SyntagmaConstraint** addsyntfeaturel = $lexicalization

add semantic feature to syntagma

This allows to add semantic features to syntagma filling a certain position. It also allows distinguishing between ' enrichment' and ' force' operations. In the first case the resulting syntagma is enriched with certain semantic feature. In the later case, the resulting syntagma is forced to ' bear' the added semantic feature.

&add_sem_feat_syntagma($description,$construction,$position,$syntagma,@semfeat)

DescriptionConstraint	• id • comment • example **ConstructionConstraint** **PositionConstraintl** **SyntagmaConstraint** addsyntfeaturel = @semfeat

add syntactic feature to syntagma

This allows to add syntactic features to syntagma filling a certain position. Syntactic features may be Closed or Open. Closed syntactic features are those already provided by the model. Open syntactic features are language dependent and, therefore, defined

by lexicons for a certain need not covered (or foreseen) by closed features. In any case, bilingual lexicons may need no new open syntactic features

&add_synt_feat_syntagma($description,$construction,$position,$syntagma,@syntfeat)

DescriptionConstraint	• id • comment • example **ConstructionConstraint** **PositionConstraintl** **SyntagmaConstraint** addsyntfeaturel = @syntfeat

Prototype Tool: Main functionalities & User manual

In this section we briefly describe the functionalities already implemented in the prototype tool at the time we provide explanations to be used a user manual.

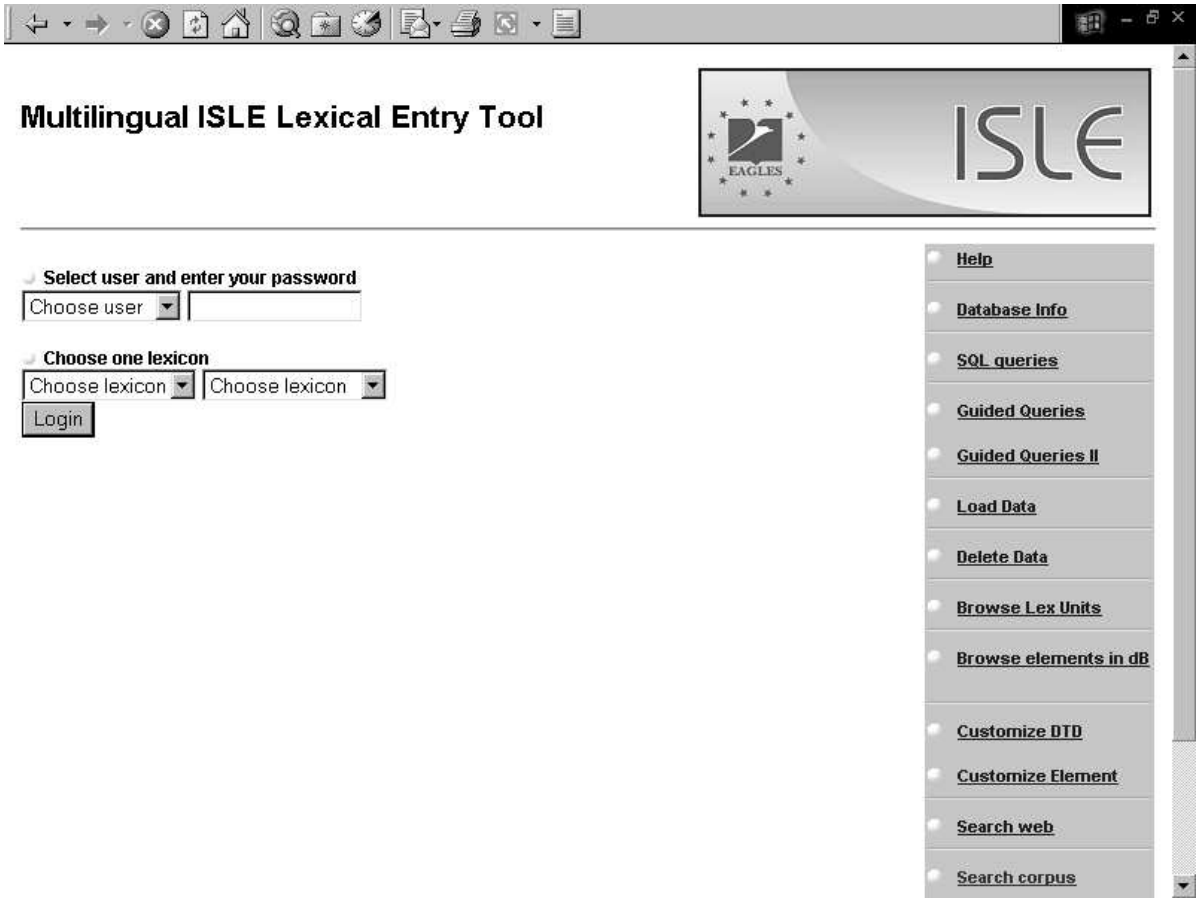
Unless explicitly stated, all functionalities bellow are DTD-dependent, that is, they involved on-line html pages that are created according to DTD specifications.

accessing the prototype tool

The access to the prototype tool is done via the temporal address http://gilc.ub.es/perl/login.pl Most functionalities can be executed remotely, however to get a full working tool it is recommended to install the prototype in a local server.

The system foresees different kind of users with different kind of privileges. Users are requested to log in. Visitors of the tool can choose **user=visitor and password=v1s1t0r**. This allows them to access all databases (i.e. lexicons) as visitors. As a general rule, visitor users cannot perform insert and delete instructions.

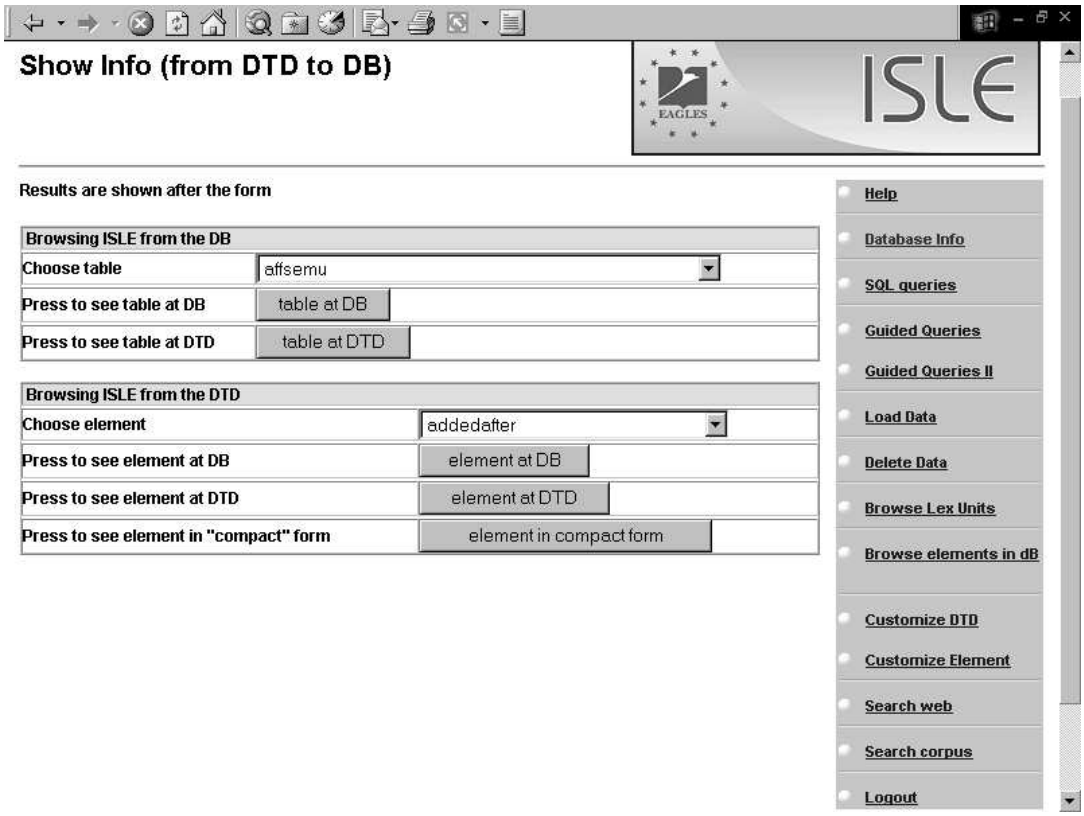
Once the user name and password are checked, the user can access any of the main options at the right.



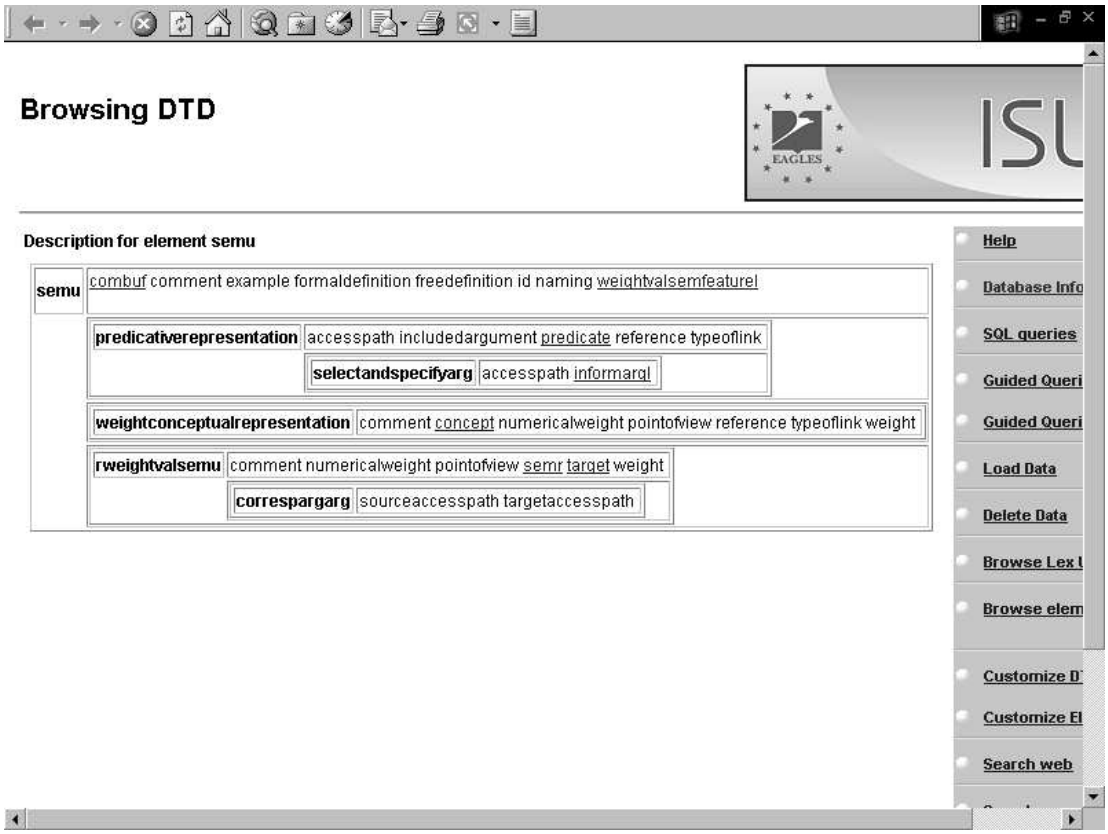
database info

This option allows browsing the structure of the database and the mappings between elements in the DTD and tables in the database.

As illustrated in the screen below, two main browsing options are available: (i) browsing MILE from the dB and (ii) browsing MILE from the DTD. In the first case, the user selects one table in order to see its description or the corresponding elements in the DTD. In the second case, the user selects one element from the DTD in order to see (i) its definition, (ii) the corresponding mappings in the database or (iii) to get a 'compact view' of the element.



The 'compact view' option allows to get complete description of elements in DTD. This description includes the attributes of the selected element, its content structure and the involved typed references if any. In this case, typed attributes are displayed as links so that the user can see the complete domain (i.e. vertical relations and horizontal relations) of the selected element.

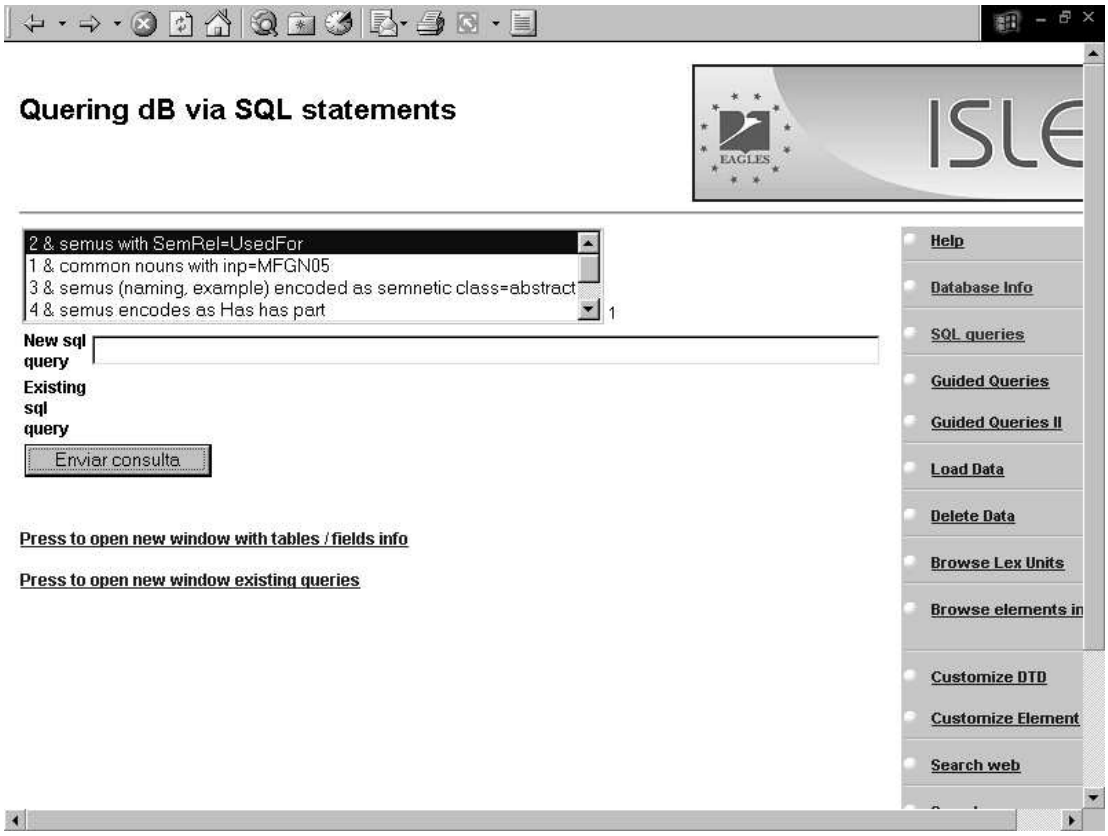


Compact view of element SemU

SQL query

This option allows quering the database using sql queries. The user can enter a new sql statement or choose one of the already existing statements. The system also offers the possibility to get a complete list of tables' descriptions in the database.

Once, the sql statement has been executed, the user is allowed to store the query for further uses.



Guided Queries

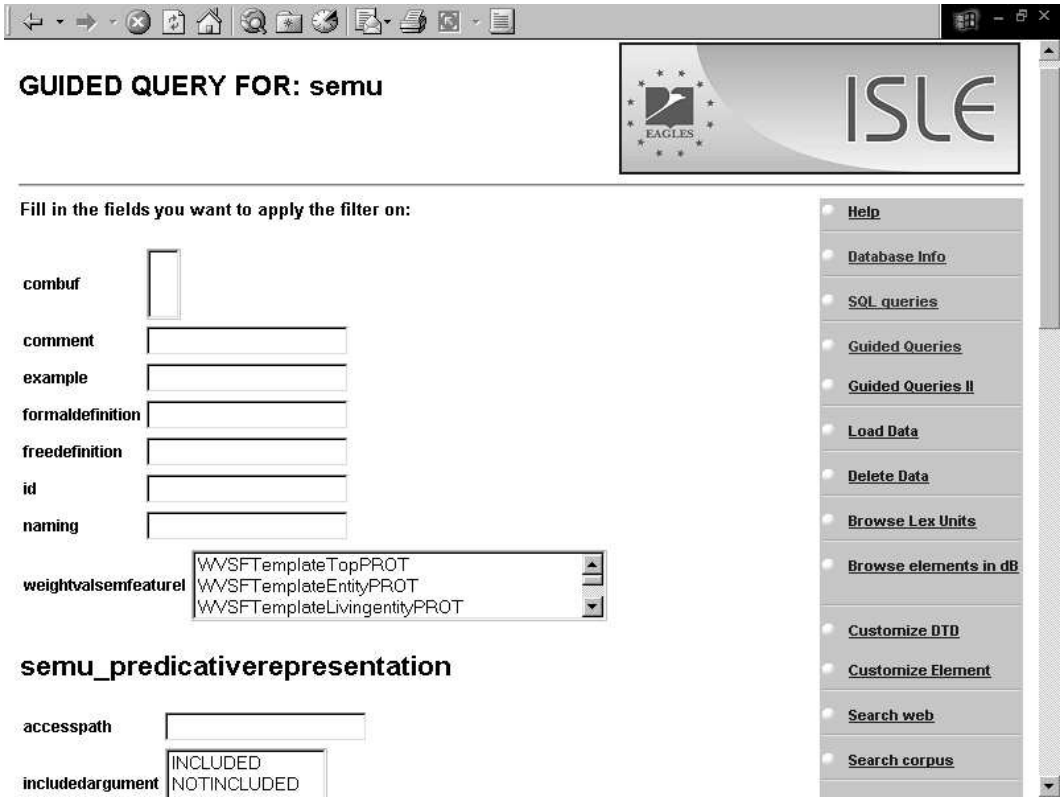
This option allows queering the database in a guided fashion. This allows the user to get information out off the database without knowing any thing about the database itself or sql language.

The process of building up a final sql query, includes different steps.

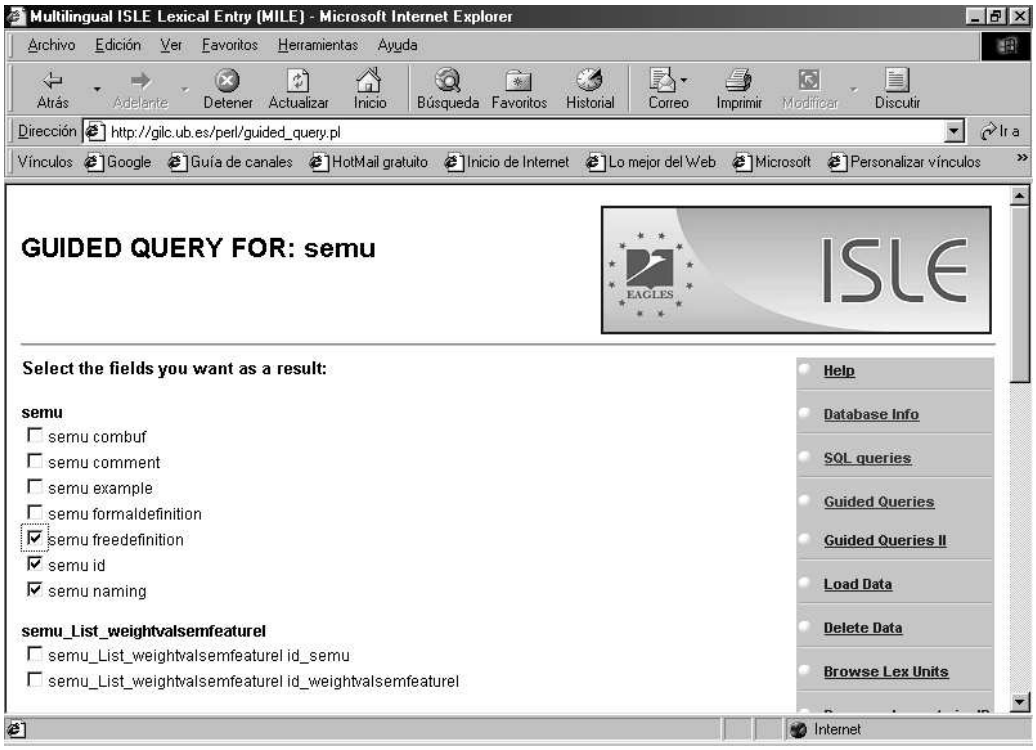
Firstly, the system displays all elements in DTD in a tree format. The user can select the relevant element he/she wants to include in the query. This allows the system establishing the domain of the final sql query. This domain corresponds to the 'FROM' part of the sql query and includes the selected element plus the base children elements.

Secondly, the system displays a full form for all attributes and content elements (i.e. base children) of the selected element. The resulting form is sensitive to the description of attributes in DTD. Thus, ENUM attributes are displayed as scrolling lists fields containing allowed values while IDREF(s) attributes are displayed as scrolling lists fields including the list of existing Ids of the relevant element. This is possible provided the user has previously declared the typed reference using the customization option.

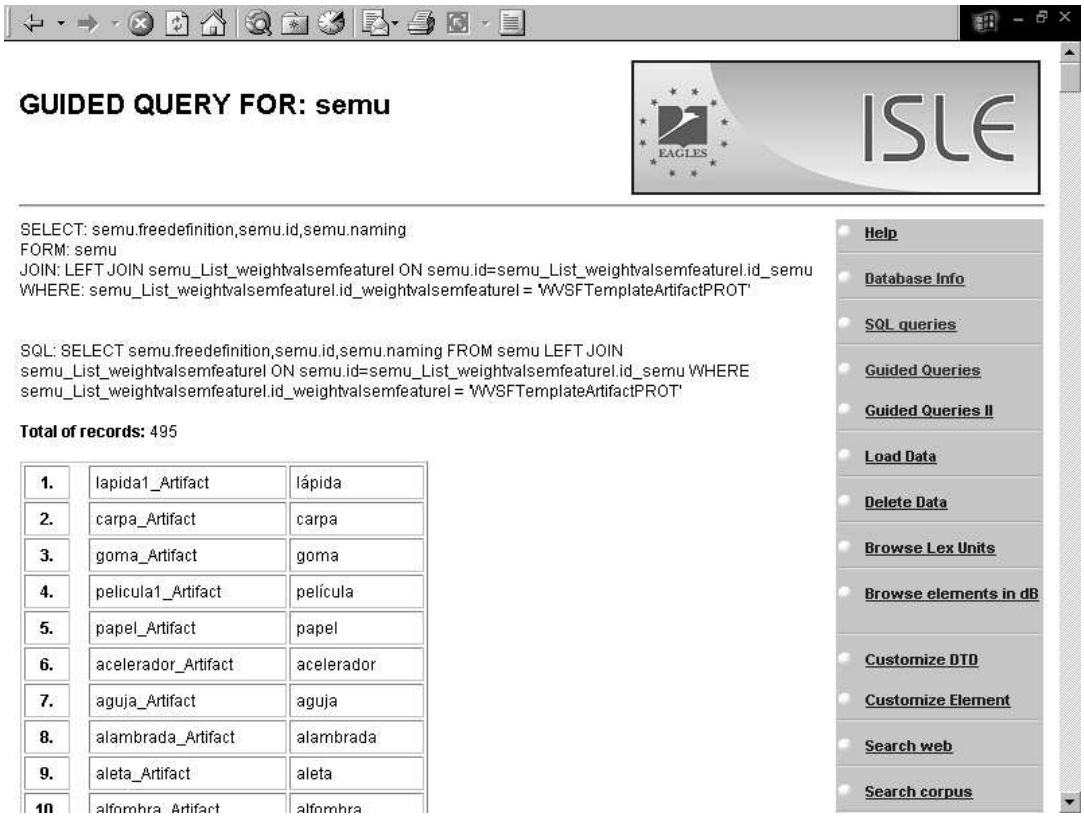
The user has to fill in the form with the desired values. The non-empty fields will actuate as filters in the resulting sql query. That is, non-empty fields are included in the WHERE part of the final sql statement.



Thirdly, the user is required to select the fields he/she wants to get the information from. This will become the SELECT part of the final sql query:



Finally, the user gets the resulting SQL statement and the corresponding results in a table format:



Load data

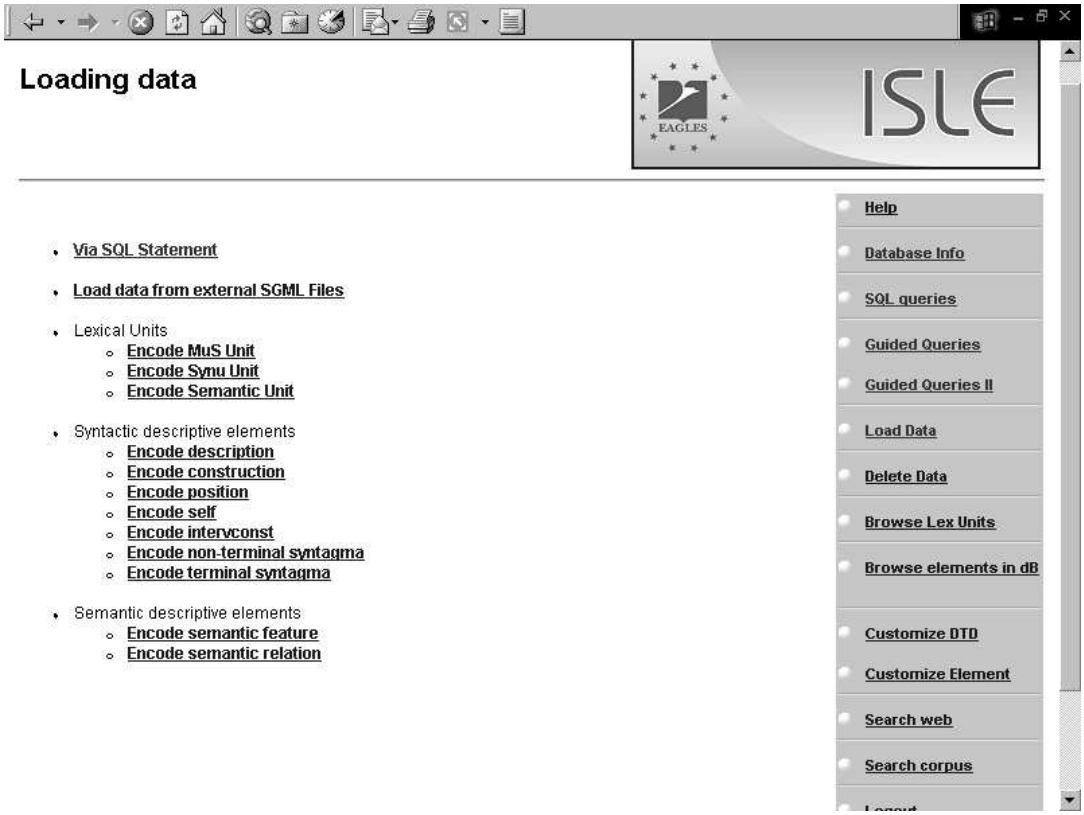
The prototype includes three different ways to load data into the database: (i) via sql statements, (ii) from external sgml files and (iii) using already defined forms. In any case the user needs to have privileges to insert data into database. 'Visitor' users do not have 'insert' privileges, thus they cannot load data into the tool.

The first two options are general and DTD dependent. This means they are part of the core db interface module that is able to work properly independently of the DTD we are using. The last option is not DTD-dependent but rather consists of a set of frozen scripts that allow encoding certain data.

In the first case, the user is redirected to the 'SQL query' option previously described.

The second option can only be executed from a local client. In this case, the user can browse his files system in order to select the DTD and the sgml data file he/she wants to load. As described before, the system parses the data using ngmls parser in order to generate the output result. All data is loaded into the database following the procedure described in section 2.3.

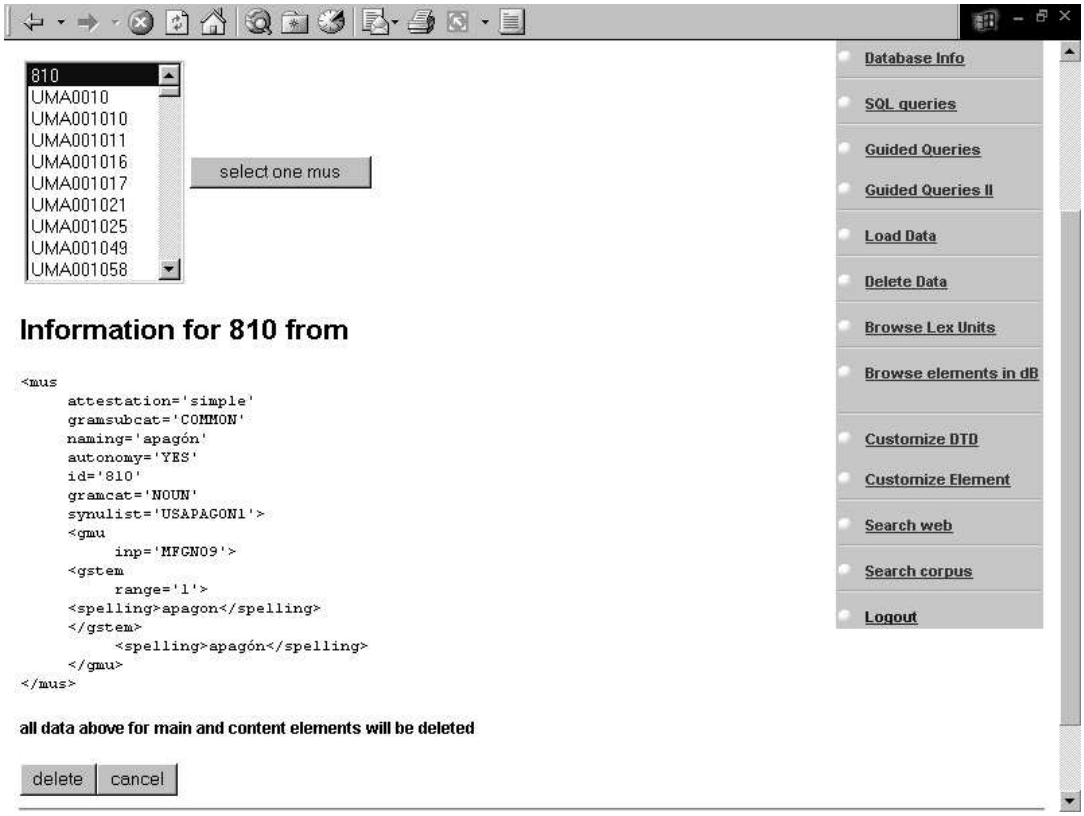
Finally, the third option includes a number of fixed scripts that allow users to load data using friendly html forms. This means that these scripts are not 'on-line' scripts that will work for any DTD. In this case, the scripts are fixed and specially defined for the particular purpose of loading specific data (lexical units). The tool, therefore, only includes a limited number of loading forms that, in the case of monolingual lexicons, correspond to morphological, syntactic and semantic lexical units and relevant descriptive elements.



Delete data

This option allows to delete data from database. The user needs to select the type of element he/she wants to delete. Then, the system accesses the database in order to retrieve all instances of that element so that the user can select the desired instance. Once the user has chosen the instance he/she wants to delete, the system displays the element with all its children in sgml format and is required to confirm the deletion.

If the user confirms the deletion, the element and all its children elements (this includes base children and long-distance children) will be deleted from database:



Remember that Visitor users do not have delete privileges.

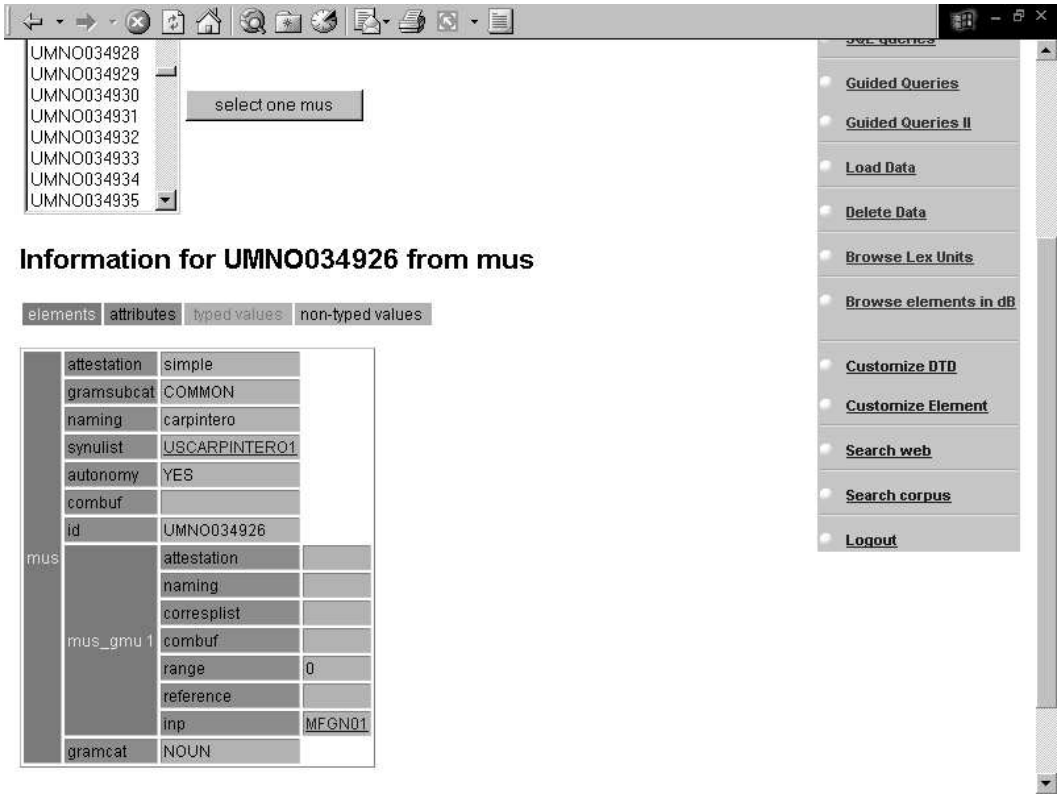
Browse lexical units

This option allows getting a complete view of the information encoded in morphological, syntactic and semantic layers for a given word. The user is requested to enter a word. The system, then, generates full sgml code for MuS, SynU and SemU lexical units. This allows having a complete view of lexical information concerning words in database

Again, this option is not DTD-dependent

Browse elements in dB

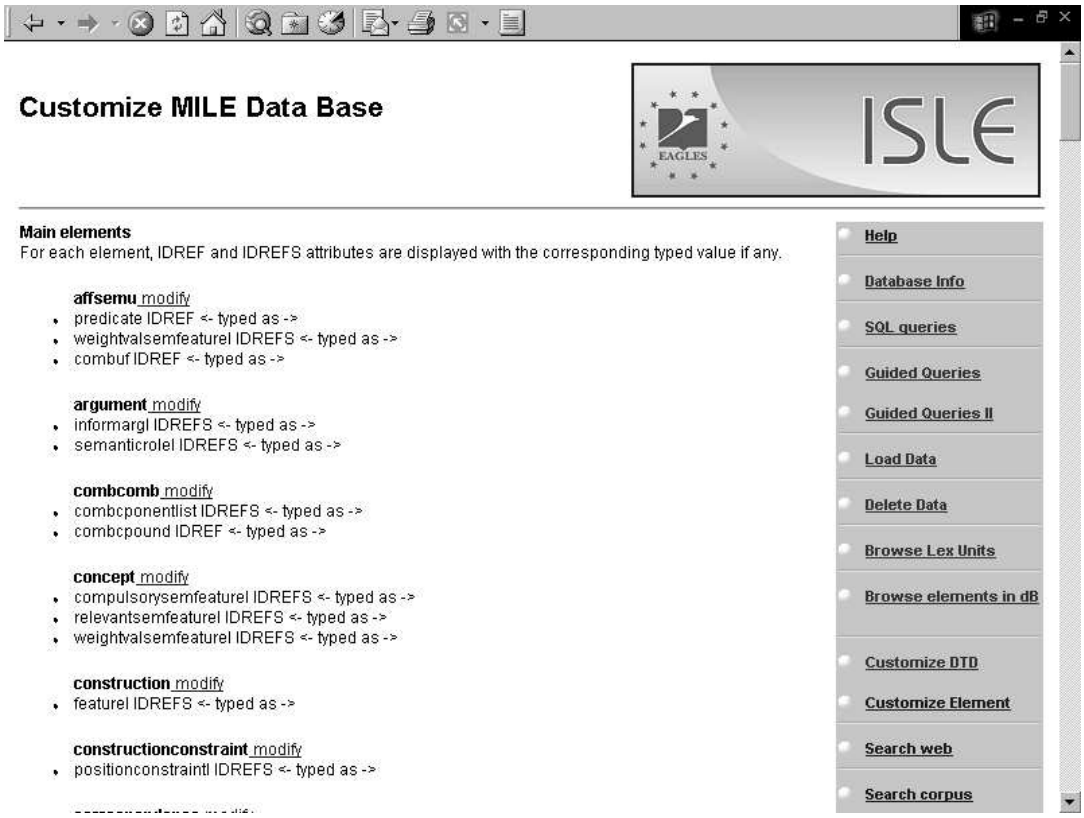
This option allows browsing information from all elements in DTD. The user needs to select one element from the DTD. Then, the system retrieves all instances of that element stored in the database and displays a new scrolling list with all Ids found. The user selects the desired instance element and gets a complete description in table format.



The information retrieved includes all attributes of the selected element plus its children description. As can be seen from the screen above, typed valued attributes are displayed as links. This allows the user browsing the involved element in order to get a complete description.

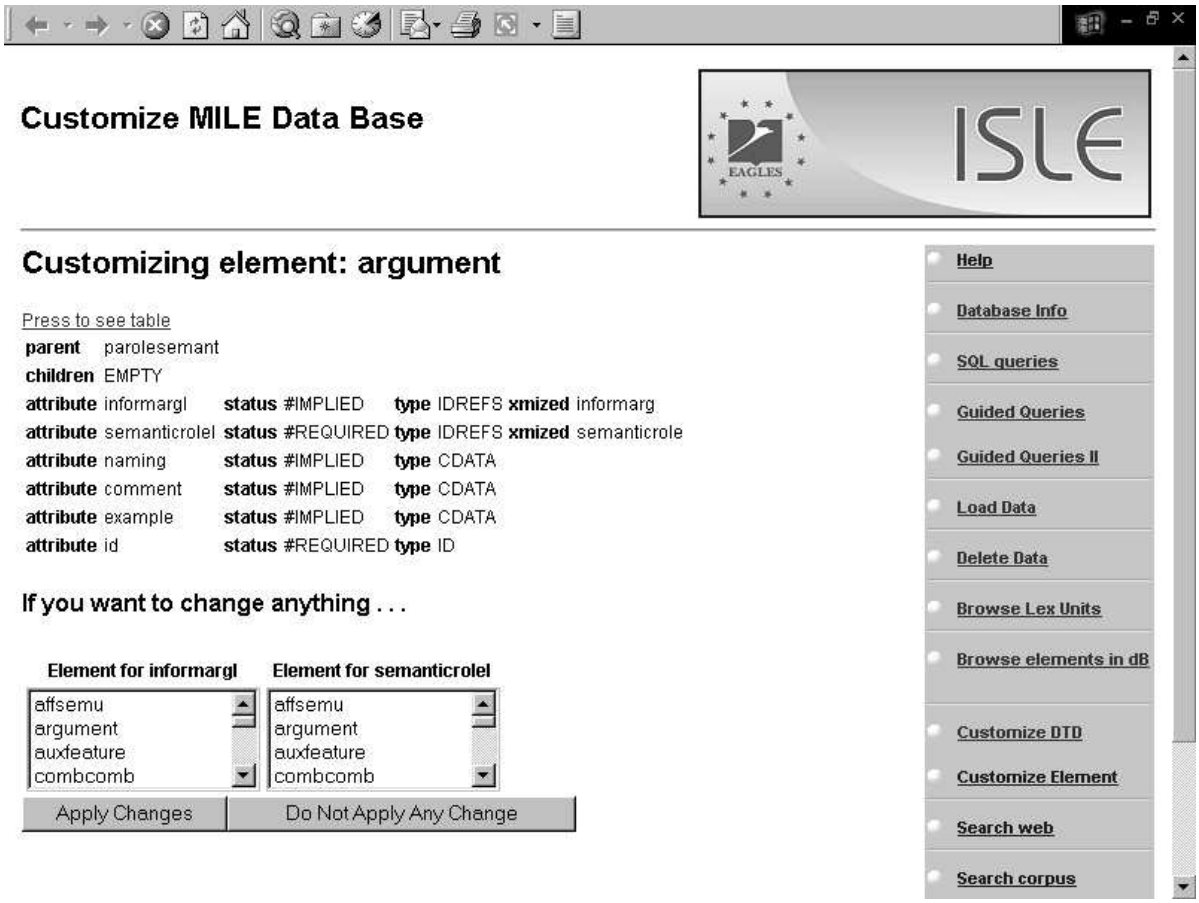
Customize DTD

Customize DTD allows to declare typed references. The system lists all elements in DTD containing an IDREF(s) attribute. If IDREF(s) attributes have been already typed, the system gives the typed value. The user can select desired elements to modify typed references.



Once the user selects an element, the system displays a complete description of the element including parent /content information and attributes description. The user is also allowed to see the mapping of the element into the corresponding table(s) in the database.

For each IDREF(s) attribute, the system allows defining the type of value (i.e. element) the implied attribute is required to take. This information will be stored in the additional table ‘relations’. As already mentioned, this information is crucial to get an effective performance of the prototype tool.



Search web

The tool comes equipped with a simple concordance generator that pushes the search string to the www. The system uses the Yahoo search engine to retrieve occurrences of the input string in the www. The retrieved lines are displayed as concordances to easy the task of lexicographers.

Search corpus

The prototype tool includes a corpus search engine that allows extracting concordances and statistical information.

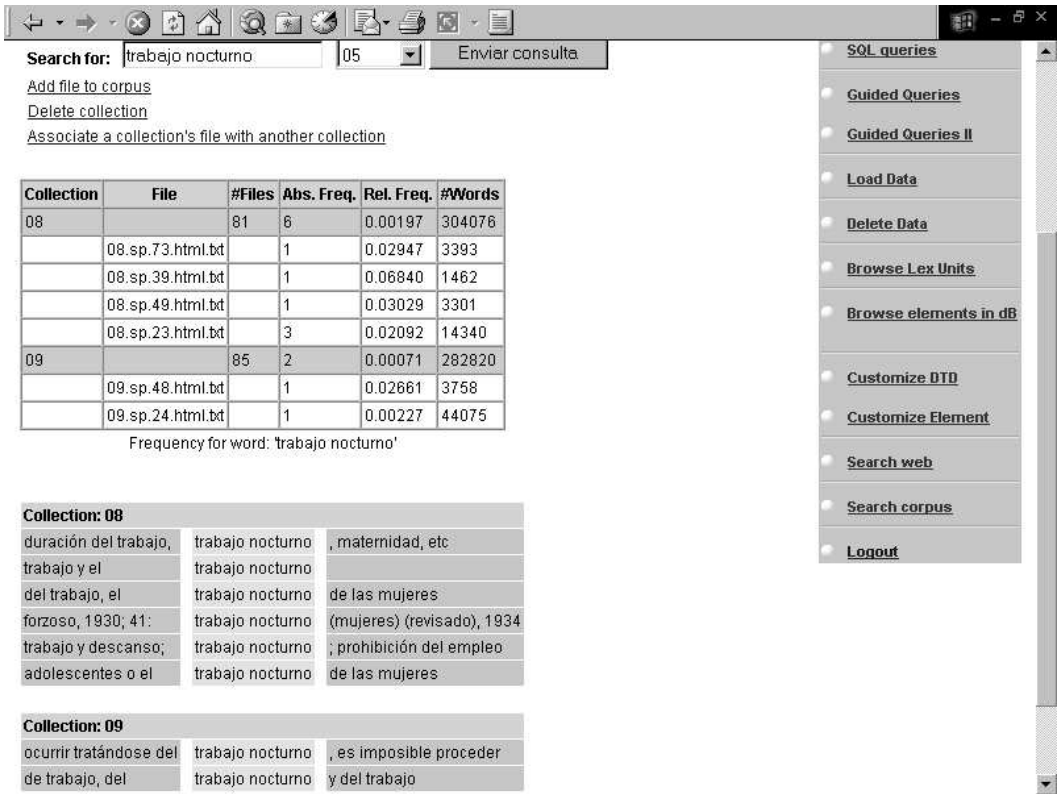
The corpus is stored in the database, using reverse index techniques. The system allows three main functionalities:

- adding new files to the corpus
- deleting a collection.
- searching facility

The corpus is organized as a series of collections (or sub corpus). Each file belongs to one or more collections. This allows the user restricting the scope of the search to a limited number of collections if desired.

As a result of the search option, the system gives some figures about the distribution of the input string through out the selected collections and displays occurrences in concordances shape.

For each selected collection, statistical figures include: information about the number of files and words included in the collection and the absolute frequency and relative frequency of the occurrence per file.
Concordances are organized into collections and speciy the file they are included in.



Each database (i.e. lexicon) is assigned one corpus. Thus, if the current database is the Spanish one, the current corpus is the Spanish one. Up to now and for validation purposes, the tool includes three different corpus: Spanish, English and Italian. Spanish and English corpus have been downloaded from ILO (International Labour Party) textual database. These corpus include more that 4 million words organized into 12 different classes (or collections). The Italian corpus merely contains a version of the Pinochio story downloaded from the web.

Software Specifications

The prototype is implemented using well supported open source resources which can be easily portable. Essentially these include MySQL data base server and Apache server:

Database server: 3.23.16-alpha version of MySQL which can be downloaded from www.mysql.com.

Perl support for MySQL: Perl support for MySQL is provided by means of the n'DBI/DBD' client interface. The Perl 'DBI/DBD' client code requires Perl5.004 or later. Perl DBI/DBD modules can be downloaded from:
www.symbolstone.org/technology/perl/DBI/index.html or
www.perl.com/CPAN/modules/by-module/DBIx/i among others.
You should get the Data-Dumper ,DBI and Msql-Mysql-modules distributions and install them in that order.

Web Server: Version 1.3 of the Apache web server which can be downloaded from www.apache.org/httpd.

Perl support for the Apache server: mod_perl is the Apache/Perl integration project. mod_perl can be downloaded from a CPAN site under modules/by-module/Apache

Validation and Assessment

The Lexicographical Station Development Platform described in this document has been successfully used to generate the corresponding relational database out of Parole/Genelex monolingual and bilingual DTDs.

The core web interface has allowed us loading all Spanish Parole/Simple sgml data and the Parole/Simple sample data from Italian, English and Catalan. The system has worked properly and has allowed loading in a very efficient way a great amount of data stored in sgml data files. This has proved crucial as it allows reusing any already existing monolingual resources for the ISLE project.

At this moment, our Spanish database contains about 60900 morphological units, 29927 syntactic units and 9924 semantic units. The tool has been used to encode new entries with no problems. As far as the monolingual part is concerned, the prototype has proved to perform correctly and efficiently. The complexity of the Parole/Genelex DTD (with more that 100 elements and 200 tables) and the amount of data loaded led us to conclude that the generation module, the load sgml module and the web interface have been largely validated.

Since reusability and monolingual lexicons are essential requirements for MILE, the efforts devoted to this part of the project are considered well spent.

The Prototype Tool is being used to load English and Italian data for MILE demonstration. Thanks to our American college Ruth Reeves, some small bugs have been detected with respect to MILE (i.e. the bilingual part of the system). We are now working to fix these bugs. Crucially, bugs reported have nothing to do with the 'DTD-dependent' part of the system but rather on the 'frozen' part of it. This allows us to guarantee, that once we get a final version of the bilingual DTD, the system will perform correctly.

The Platform has been also used to generate a relational database out off a small and simplified DTD for the bilingual lexicons in Peking project (IST 2000 25338). At present, we are using the web interface to encode Spanish/English bilingual entries for the Peking project. Our Peking bilingual database contains 11000 bilingual units and we expect having 15000 by the end of this year.

References

Bel, N., Busa, F., Calzolari, N., Ogonowski, A., Peters, W., Ruimy, N., Villegas, M., Zampolli, A. 2000. SIMPLE: a general Framework for the Development of Multilingual Lexicons. In *LREC Proceedings*, Athens.

Busa, F., Calzolari, N., lenci, A., and Pustejovsky, J. 1999, Building a Semantic lexicon: structuring and Generating Concepts. In *The Third International Workshop on Computational Semantics,* Tilburg, The Netherlands.
Calzolari, N. 1991. *Acquiring and Representing Information in a Lexical Knowledge Base,* ILC-CNR, Pisa ESPRIT BRA-3030/ACQUILEX – WP No. 16.

Calzolari, Nicoletta, Lenci, Alessandro, Zampolli, Antonio, Bel, Nuria, Villegas, Marta, Thurmaier, Gregor., 2001. The ISLE in the Ocean Transatlantic Standards for Multilingual Lexicons (with an Eye to Machine Translation). In *Proceedings of MTSummiy VIII,* Santiago de Compostela, Spain.

Calzolari, N., Mc Naught, J., Zampolli, A. 1996. EAGLES Final Report. Pisa.
Fellbaum, C. 8ed.). 1998. *Wordnet: An Electronic Lexical database,* Cambridge, MA: The MIT Press.

Fontenelle, T. 1997. *Turning a bilingual dictionary into a lexical-semantic database.* Tübingen: Max Niemeyer, (Lexicographica. Series maior; 79).
GENELEX Consortium. 1994. Report on the Semantic Layer, Project EUREKA GENELEX, Version 2.2.
Underwood, N., Navarretta, C. 1997. A Draft Manual for the Validation of Lexica. *Final ELRA Report,* Copenhagen.