The ELAN Architecture

Christophe de Saint-Rat, Samuel Cruz-Lara, Patrice Bonhomme, Laurent Romary

ELAN Deliverables

WP3

DECEMBER 1999

The ELAN Architecture

Deliverables D3.1-1 and D3.2-1

Project Number	**MLIS-121**
Project Title	**European Language Activity Network (ELAN)**
Deliverable Number	D3.1-1 and D3.2-1
Work Package ID	WP3
Contractual date of delivery to EU	31 December 1999
Actual date of delivery to EU	17 December 1999
Deliverable Title	**The ELAN Architecture**
Authors	*Christophe de Saint-Rat*
	Samuel Cruz-Lara
	Patrice Bonhomme
	Laurent Romary

Abstract

The ELAN project is a distributed language resources system, offering access to existing resources to their potential users throughout Europe. In order to serve the electronic multilingual resource market, our task is to specify and elaborate a network of inter-connected resource servers. This document defines the technical specifications of each nodes that form the ELAN network: the user interface of the ELAN client; the resource servers and the network management unit.

Keywords

Language resources, Distributed systems, Network, User interface

1 Introduction

1.1 Scope

One of the objectives of the ELAN project is to define a proper software environment through which it might be possible to access and/or distribute linguistic resources, which would be spread among different servers.

There are different reasons why to go about in this fashion.

First, it should be considered that there already exist several sites around the world (most of the European ones being represented in ELAN) which actually act as brokers for different types of linguistic resources. The idea emerging behind the sole period of the ELAN project is actually to be able to bring together the corresponding resources, in order to provide any user with a global access to these. In some cases this will lead to more coherence between these databases since redundancies might be detected or complementary resources (e.g. parallel texts) put into correspondence.

Second, the classical view of a centralized database containing all the information in a given domain is far from applicable to linguistic resources where, because of their intrinsic diversity (prose, theatre, poetry, newspaper articles, dictionaries, historical documents etc.), there is a need for them to be created and above all maintained at a place where there is the competence to do so. In particular, the encoding background adopted within ELAN, that is SGML (and its subset XML), allows one to continuously enrich documents with specific linguistic annotations like part of speech (POS) tags or proper names (to quote only two possibilities).

Finally, there can be specific constraints that can preclude some given resources to be deported to another site than the site which has originally created them. In particular, some of the partners within ELAN have specific agreements with publishers, which express strong conditions on the actual distribution of electronic files. It is thus more sensible not to take the risk of hampering the agreement by overly spreading the corresponding contents. In the case of the ELAN network, each resource is only accessible through specific queries which can thus be controlled as to their actual applicability.

1.2 User in mind

Still, from the user's point of view, there should not be much change in the way the resources are to be accessed, which means that whether there are one or several servers should be rather transparent to him. Of course, taking into account the short duration of our project, we will not supply, in the prototype to be delivered, the whole set of functions that an existing server such as, say, the one developed by Leiden, is currently able to provide its own users with. Transparency is thus to be understood with regard to existing different servers, not from the point of view of benefiting from all the possibilities of the different sites at the same time (at least for the duration of the project).

As we will see, adopting a distributed framework, as opposed to the classical view of a centralized database, induces several specific problems for which this deliverable is trying to provide some plausible answers. Among those, we will have to deal specifically with the problem of broadcasting queries to different servers and conversely combining the corresponding result sets. As an example, statistics can only be dealt with in our distributed architecture if part of the computation is kept on the remote servers' sides and part is carried out locally (on the access server).

1.3 Objectives

In the context presented above, the objective of Work Package 3 (Software and Network) can be seen as an attempt to validate the feasibility of a networked environment for accessing linguistic resources and put forward the main aspects on which it would be necessary to focus in the future.

Beyond that, and coherently with the general approach of the standards of the project, we have tried to base any proposal and/or implementation on existing (and even emerging) technologies (i.e. Java, Swing, CORBA, XML), so that to ensure some kind of durability to our work.

Precisely, the aim of this work package is to ensure a good synchronization between the different implementation tasks by providing a precise description of:

- the interaction scenario between the user and the network (through the local access point);
- the interaction between the different elements of the network;
- the interaction between the system components of Network Access as well as the interaction of Network Access with related work packages.

In particular, the task will have to put forward the actions to be taken to ensure the consistency of the different databases held by LR servers (users, meta-data, network configuration …), stating precisely when and in which condition the corresponding data are accessible and modifiable.

This deliverable (t0+3) is a report defining:

- the precise organisation of the user interface;
- the organisation of the network proper;
- the flow of information within the network;
- the organisation of maintenance and updating of Network Access components.
- The deliverable at t0+9 will be a software corresponding to a platform independent user client.

2 General network organisation

The general purpose is to develop a client/server system which allows a user to contact one or more servers and select sub-corpora for querying, taking into account various access conditions.

2.1 A decentralized network - rationale

All tasks dealing with the user interface should be concentrated on the client side, while searching and other computationally intensive operations should be accomplished by the server.

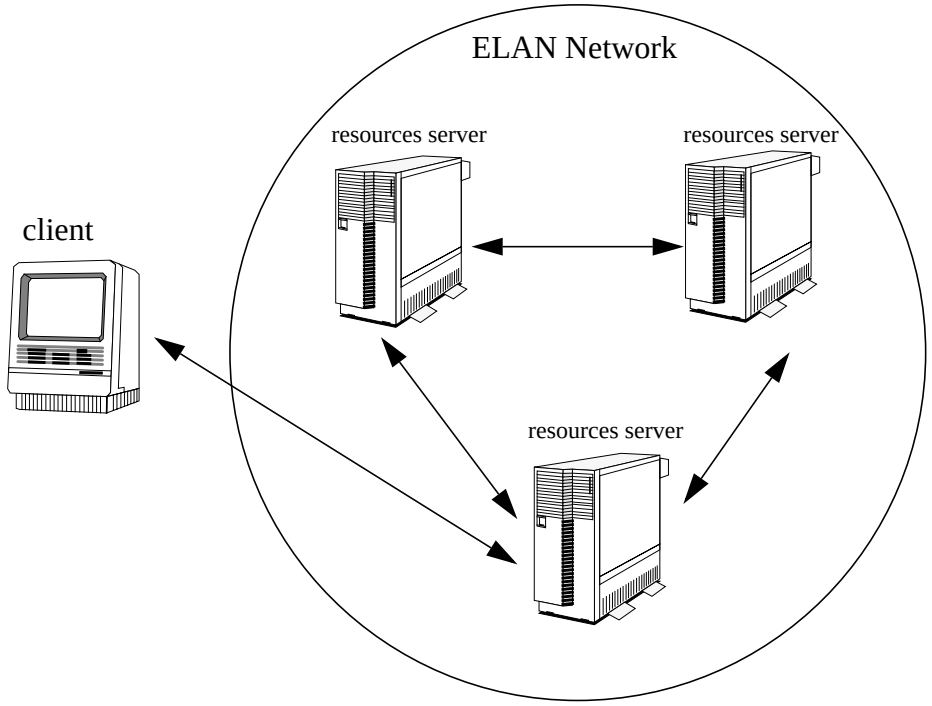


Figure 1. **General structure of the ELAN network**

The network (see Figure 1, “General structure of the ELAN network,” on page 9) has the following characteristics:

- each server is an autonomous unit containing its own linguistic data;
- when a server receives a request he acts as a “broker” and transmit the request to other servers in the network which are known to it;
- one server is accessible to registered users through a general purpose Java-compatible web browser.

2.2 A user interface - friendly

This interface should be designed with a non-technical user in mind. Technically advanced features should be available in an intuitive way. Although all system components will have their own interface, due to different functions, they must have the same “look and feel” (i.e. surface and behavior). This reduces the time the user needs to become acquainted with the network and contributes to the aspect of simplicity.

This user interface will be implemented as a client at the level of which little, not to say no, linguistic resource processing is to take place (notion of thin client). Basically, the interface will allow a user to make his different queries and will display result sets according to some specific style-sheets associated with these.

The following design criteria should be adopted for the interface:

- it should be GUI-based ;
- its implementation should allow the user to choose its interaction language (in particular, any PAROLE or TELRI language has to be taken into consideration).

In case of an error the user should be supplied with clear information about the error and what he can do to correct it. If the error could not be corrected by the user (e.g., due to a software bug) the helpdesk is automatically informed.

2.3 And the users

To be in line with the idea of a decentralized network, we have consider that a given user should only have to be registered at one given site and that no central user database should have to be set up. From the network point of view this has two consequences:

- each server should manage its own user databases, containing both the general user identification information and dynamic information related to session parameters (see section “NMU / SERVERS communication” on page 26) ;
- each time a query is broadcasted from an access server to a remote server, an identification tag (user id and authorization level) should be transmitted in order to evaluate the applicability of the query.

3 User client (Deliverable 3.1.1)

3.1 Introduction

Given the preceding architecture, any authorized user will be provided with an environment which will lead him along the following steps:

1. connection to a local server;
2. user identification;
3. choice of working servers. Given the list of available LR servers - available through the local server - in the network, together with their respective server profiles, the user will select those servers which may provide the proper resources or the proper services (tools) he wants to access ;
4. selection of a subset of resources. Through an iterative process of requests to the selected servers, the user will build up a virtual sub-corpus (i.e. By way of pointers to individual resources) upon which he will actually work ;
5. access to document content proper. The user specifies a query which is broadcast to the appropriate servers holding the data selected in the user's sub-corpus. The queries are transmitted after being translated into a Common Query Language - CQL - shared by all the servers in the network.

As a consequence, the ELAN project makes a distinction between two types of queries:

1. **Resource Selection Query (RSQ).** This type of query corresponds to step 4 above. Its interpretation is optimized as it accesses the index database.
2. **Content Access Query (CAQ).** This type of query corresponds to step 5 above. It represents a direct access to the document content.

3.2 ELAN main services

3.2.1 Connection to a local server

Within ELAN each individual server is responsible for the management of its own database of registered users. A common code of practice is to be shared between the servers belonging to the network to ensure a coherent verification scheme among them.

As a result, the first step in the user-client scenario is to connect to one’s own access server by means of a web browser which then downloads the user client as an applet window. As shown in Figure 3, “ELAN user's reception panel,” on page 13. the applet gives access to the different ELAN services as described below.

As an alternative, it is also possible to download a client application and use it to connect to the network instead of using a browser. This is useful in two cases :

- the browser is not compatible with the ELAN system (very last versions of the usual browsers should be used.)
- the internet connection is slow and the time to download the client applet at every connection is very long.

During the process of sub-corpus selection, the access information of the documents is matched with the user's entry in the user registration. Access to the documents is provided accordingly.

3.2.2 User identification

In order for a user to connect to a server, he must follow an identification process. Identification is the process that a server executes to verify if a user is a registered ELAN user.

Each registered user will be identify with a unique login name and a secret password.

Guest access can also be granted depending of each server policy.

3.2.3 User registration

Registration is the process that all new users will have to follow in order to use an ELAN server as a registered user (see Figure 2, “Users registration,” on page 12). Obviously, registration is done by filling in a form. A login name and a secret password must be provided by the user. The server verifies that the selected login name is unique. The registration will be validated by the administrator of the local server. For this purpose, an administration interface will be also provided with each server of the ELAN network.

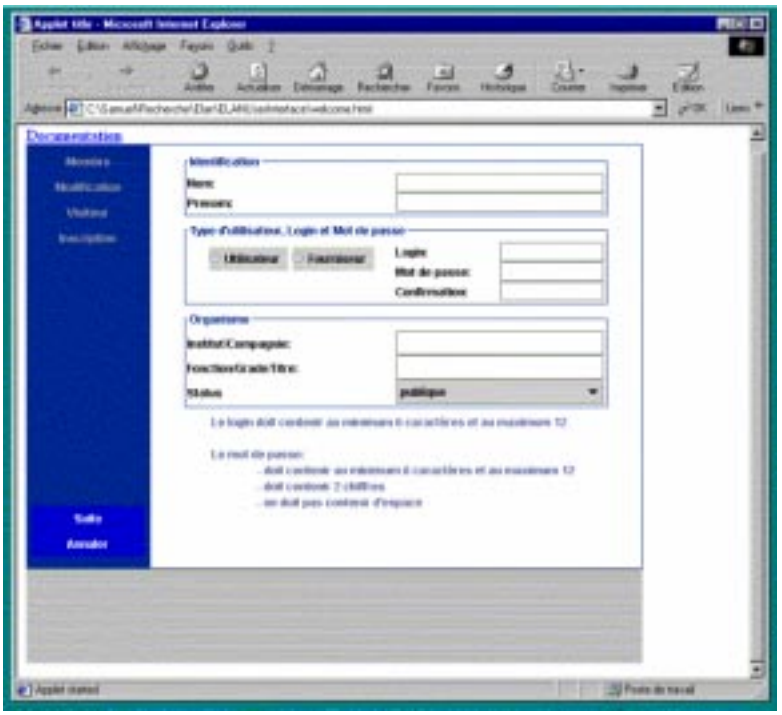


Figure 2. **Users registration**

3.2.4 Client environment and interface

The ELAN client environment is made of a tabbed window where each tab grant access to a specific functionality.

The ELAN interface will offer the following functions :

- Selection of the working servers;
- Selection of working resources;
- Selection of tools which may be applied to resources in the shopping basket;
- Manipulation of the shopping baskets;

- Definition of the user preferences;

The client window is build as a “reception panel”. Inside this panel, we have several other panels which may be selected by the way of a tabbed component. Each panel offers one of the functions enumerated above (see Figure 3, “ELAN user's reception panel,” on page 13)

The “Servers” tab allows a user to select the servers he wants to access. This panel is composed by a list of “referenced servers”. For each referenced server, we indicate its name, its URL and its state (i.e., on-line or off-line). Thus, a user can dynamically select all servers corresponding to his needs.

The “Resources” tab allows a user to dynamically select all the resources he wants to work with using the “shopping basket” paradigm.

The “Tools” tab presents a set of tools that the user could use with his list of selected resources. These tools should allow “concordances queries”, “word list queries” as well as “statistical queries”. All tools are independent from one another, and a new tool can be added easily.

Finally, the two last tabs “Preferences” and “I/O” allow a user to respectively personalize his own environment (saving options, interface language etc.) and manage his shopping baskets (creation, deletion).

Figure 3. **ELAN user's reception panel**

a) The Workspace

In the ELAN system, the user should be able to save different kind of information or data for later use. A registered user probably does not want to select again his client interface language every time he connects to the network, or the servers or resources he usually works with.

As we see, there is a need for a private environment associated to each registered user. This is what is called the Workspace.

The Workspace is saved on the user's login server, and reloaded every time he connects to the network, so his private environment is automatically restored from one session to the other.

A Workspace usually contains :

- the environment preferences (see "the preferences panel" below)

- the list of selected servers (see "selection of servers" below)
- the user's saved resources basket (see "selection of resources" below)

Note that only registered users can have a private workspace, and can save it on the server. Guest users will use a "default" workspace, and won't be able to save it on the server.

b) Servers selection

Before any query session (resource selection), a user has the possibility to select among a set of on-line servers (see Figure 4, “Servers selection panel,” on page 14), the server(s) he wants to work with. At any moment of the process, the user can edit his list of working servers and modify it.

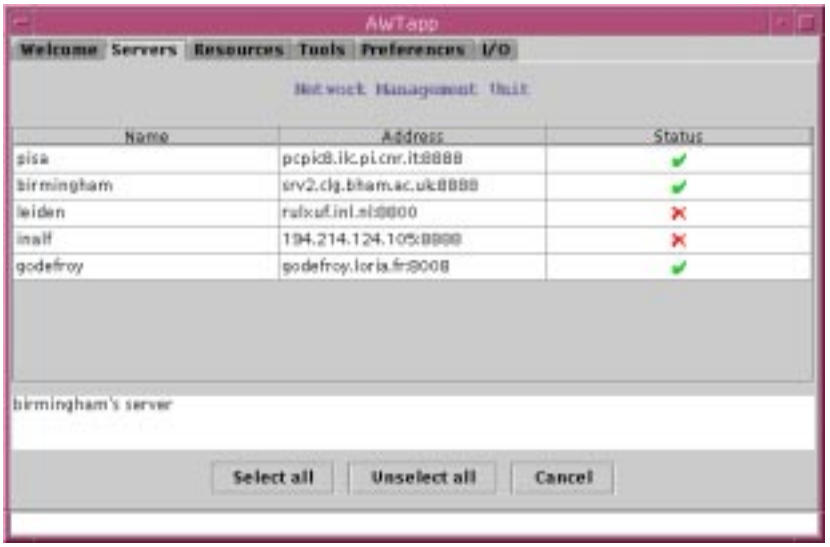


Figure 4. **Servers selection panel**

c) Resources selection

The principle is to select resources from the various databases on the different servers, and to combine them in “shopping basket”. The user has the possibility to query either the whole ELAN network or a subset of selected resources servers. The resource selection is based on the corpus documentation (meta-data i.e. the TEI headers) and enables the user to make some simple queries (on a single field) or multiple queries (combination of fields with Boolean expressions). For that purpose, the user will have access to a friendly query interface for editing, modifying and sending his queries to his local server.

Another way for the user to select his working corpora is to browse through the different sub-corpora available on the different servers.

d) Tools and content query

As the main goal of this architecture is to give a user an access to a large set of resources, it is mostly important to provide an environment dedicated to the tools and content access. In these tools the user will be able to edit CQL queries and send them to the various servers referenced in his “shopping basket”.

The interface will obviously depends on the type of query and results that are expected. To edit the CQL queries, two possibilities should be provided:

- A simple or basic querying environment for novice users based on a GUI interface, that will allow simple queries to be made with no special knowledge of the CQL language.
- More skilled users should have the possibility to enter a query by just typing it in the CQL language. Much more sophisticated queries exploiting the full power of the CQL could be made this way.

Various tools will be provided, ranging for mono or multiLingual concordances queries to statistical queries.

e) Shopping baskets

Shopping baskets keeps the resources that the user has selected. A user can have several baskets with different resources, but only one basket can be active at a time.

The resource selection is made by way of the “shopping basket” paradigm : while the user consult the resources he wish to access, he puts them in his current “shopping basket”.

Obviously, the shopping basket can interactively be modified and kept from one session to the other. All on-line operations are effectively realized on the shopping basket, that is, download operations – when possible -, but also, all operations related to the tools which may be applied to the resources, for example, mono and multilingual concordances. These tools can be selected by means of the “Tools” tab.

3.3 Technical requirements and implementation

3.3.1 XML for data and message encoding

In the framework of ELAN, all flowing data, (i.e. requests, results, messages, …), as well as all information about users and user’s working spaces, will be encoded using XML (see W3C REC-xml-19980210).

It is right now usual to use the SGML (see ISO 8879:1986) standard for encoding language resources (see PAROLE, TELRI, Silfide, … projects). However, it has been observed, especially within the World Wide Web Consortium (W3C), that SGML was somehow too complicated to be – as such – a possible exchange format on the Internet. As a result, a subset of SGML – XML (which stands for eXtended Markup Language) – has been proposed (see W3C REC-xml-19980210). It is suggested to design the network infrastructure for linguistic resource distribution upon this new standard for the following reasons:

- XML is a proper subset of SGML and as such will be compatible with most SGML applications implemented so far;
- XML has been fully adopted by major software providers in the framework of next generation web browsers (in particular, Microsoft’s Internet Explorer 5, Netscape’s Communicator 5.0 and Sun’s Project X), which will ensure platform independence for the ELAN network;
- XML provides an advanced linking mechanism (inspired by the TEI extended pointer mechanism) – the XML Pointer Language (see W3C WD-xptr-19980303) and the XML Linking Language (see W3C WD-xlink-19980303) -, which can be the basis for the identification of resources across the ELAN network.

a) Resource meta-data encoding

PAROLE recommendations for linguistic resource representation are based on the TEI guidelines, which is a document representation meta-language widely used both in the academic and the industrial fields. It may thus seem natural to base any definition of a document exchange infrastructure upon the same very language. Each resource available on the ELAN network will have an associ-

ated documentation, also called meta-data. This information will be encoding using the PAROLE recommendations.

As the XML recommendation is a subset of the SGML standard, it should be noted that both CES and TEI encoded data could be readily transformed into XML data.

b) ELAN Workspace encoding

The ELAN workspace is in fact an XML document associated with a given user and which is stored on the local server between two given sessions in order to keep any information related to the user identification or activity. It basically comprises the following elements:

- Identification information about the user;
- the current server selection (only one selection is being considered and saved);
- the different sub-corpora (or “shopping baskets”) selected by the user;
- a history of user queries to the server;
- possibly some other parameters such as personal word lists that the user would like to keep.

Appendix B. “The ELAN Workspace DTD” on page 99 presents a proposal of DTD for the ELAN workspace and Appendix C. “A simple workspace instance” on page 78 shows an elementary instance of such a workspace.

c) Queries and results sets encoding

In ELAN, results sets as well as queries are valid XML document. As such, they can be considered as instances of DTDs. The various DTDs define the protocol used to build queries and results sets. All documents or messages exchanged within the ELAN Network are made of two parts :

- a “header” compliant to the SIL DTD and providing general information (user id, server id, etc...)
- a specific part compliant to a DTD depending on the document or message type (query, resultset, etc.)

Specific APIs are provided to parse, build and more generally deal with these documents. The API documentation is available at the ELAN Software site: `http://www.loria.fr/projets/MLIS/ELAN`

d) The SIL DTD

This is the general header associated with every document or message within ELAN. The information stored include:

- The document’s type
- The id of the server that issued the document
- The version of the DTD this document follow
- A user identification section that includes :

  - the login and password (encrypted) of the user

  - the group(s) the user belongs to

e) The ELAN Query DTD

All ELAN queries are embedded in an <elanQuery> XML element, which content differ depending on the query type. A specific DTD has been defined for all query types.

Here is an example of a concordance query document:

```
<!DOCTYPE sil SYSTEM "sil.dtd">
<sil sid="leiden" type="elanQuery" version="0.5">
   <uid type="user">
      <login id="cruz-lara"/>
      <passwd><![CDATA[ !@#$%^&&% ]]></passwd>
      <access>
         <default group="guest"/>
      </access>
   </uid>
   <elanQuery>
      <concQuery>
         <textQuery>A CQL Expression</textQuery>
      </concQuery>
   </elanQuery>
</sil>
```

f) The Result sets DTD

In ELAN all results sets are basically made of two parts:

1. a meta part that contains general information about the results set. This includes:

   - the total number of documents matching the query

   - the actual number of documents returned in this results set

   - eventually error messages issued by the servers

2. a results part that contains the differents results. Each result is basically made of:

   - one or more lines of raw text

   - a set of attributes

The attributes names depends on the type of results. For example, in case of a concordance query results set, each result is made of 3 attributes:

- node contains the node word
- leftContext contains the leftContext
- rightContext contain s the right context

Here is an example of a concordance query result set document :

```
<!DOCTYPE sil SYSTEM "sil.dtd">
<sil sid="leiden" type="resultset" version="0.5">
   <uid type="user">
      <login id="cruz-lara"/>
      <passwd><![CDATA[ !@#$%^&&% ]]></passwd>
      <access>
         <default group="guest"/>
      </access>
   </uid>
```

```
    <rs>
      <meta>
        <attr name="size">2</attr>
        <attr name="totalSize">5</attr>
      </meta>
      <result type="concordance">
        <record idno="FRW1">
          <attr name="leftContext">one ring to rule them</attr>
          <attr name="node">all</attr>
          <attr name="rightContext">, one ring to find them,</attr>
        </record>
        <record idno="FRW1">
          <attr name="leftContext">one ring to bring them</attr>
          <attr name="node">all</attr>
          <attr name="rightContext">and in the darkness bind them</attr>
        </record>
        <record idno="FRW2">word matched by the CQL expression </record>
      </result>
    </rs>
  </sil>
```

3.3.2 Software requirements

For the same reasons of platform independence, any new piece of software developed within the ELAN framework will be implemented in Java, a programming language which is getting the status of a de facto standard for Internet oriented applications. Among other aspects, it is possible to mention the following arguments for adopting Java:

- Java based Applets (i.e. Web based applications) will run on most existing Web browsers, thus keeping the software requirements as limited as possible;
- the “Servlet” API. A servlet is a server-side component, which in the context of HTTP, is the Java equivalent of CGI programs;
- Swing, which is a GUI component kit that simplifies and streamlines the development of windowing components. The Swing component set is part of a class library called the Java Foundation Classes (JFC);
- several XML parsers developed in Java (but not only!) are already available which will ease specific development dealing with PAROLE compatible data;
- internationalization, which is the process of designing an application so that it can be adapted to various languages and regions without engineering changes;
- Java comprises a treatment of Unicode [see ISO 10646] compatible data, making any software language independent;
- Java IDL, which adds CORBA (Common Object Request Broker Architecture) capability to Java, providing standards-based interoperability and connectivity. Java IDL enables distributed Web-enabled Java applications to transparently invoke operations on remote network services using the industry standard OMG IDL (Object Management Group Interface Definition Language) and IIOP (Internet Inter-ORB Protocol) defined by the Object Management Group;
- Java can be interfaced, in particular, with C/C++ programming languages, thus ensuring compatibility with the software available at the different partner’s sites.

In the framework of ELAN's user interface, several Java's API should be used, for example:

Using Swing may allow us to build high-level and high-quality GUI. As Swing is a 100% Java API we keep the benefits of portability, platform independence, etc. Swing components are said to be light-weight because they don't rely on user-interface code that's native to whatever operating system they're running on. In addition, with Swing´s "pluggable look and feel" (PL&F) capabilities, we can make the windowing components take on whatever appearance and behavior we like. So, we can make them look and feel just like native components of the user's computer system, or we can give them a uniform cross-platform look and feel – that is, a look and feel that always has the same appearance and behavior, no matter what kind of system is being used.

We should note that using Swing implies that, web browsers used in the framework of ELAN, must be "Swing-compatible". At present, only Sun's HotJava is 100% Swing-compatible. In order for Netscape's Communicator and Microsoft's Internet Explorer to be Swing-compatible a "Java Plug-in" must be installed.

Also, it should be easy to see that Java's "internationalization" will allow us to offer to ELAN users the possibility of working with several PAROLE or TELRI languages: English, French, Italian, Spanish, German, Dutch, Swedish, Finnish, Danish, Greek, and Portuguese. It should be noted that, several Java classes can convert between Unicode and the set of character encoding presented on Appendix A. "JDK Supported Encoding" on page 73.

3.4 Statistical tests

3.4.1 Introduction and trade-offs

Working on textual resources does not only mean browsing through contexts centered on a given node word of expression. The ELAN network also has to provide users with the possibility to operate on the corresponding contexts, in particular through statistical tests that may designate for instance possible collocations associated with the node word. Still, there is a hard trade-off when considering where exactly in the general ELAN architecture such a computation should take place, because of the following constraints associated with the different elements relating a user with the resource he wants to work with.

First let us consider what information is needed for the statistical tests we contemplate here:

- Reference frequencies — the number of occurrences of a given word in the "whole" textual database. Depending on what is actually needed (by the user), these frequencies can be the combination of the occurrences of the word in all texts for all databases, or restricted to the texts which he has selected;
- Contextual frequencies — the number of occurrences in the concordance lines selected by the user.
- Positional information — in specific cases, it is important to know the contextual frequencies associated with the different positions one given word occupies around the node word.
- Total size of the reference database — the total number of tokens in the reference corpus as chosen to compute reference frequencies;
- Total size of the concordance lines — the total number of tokens within the set of selected concordance lines (from which contextual frequencies have been computed)

It should be noted that these different figures can be combined (summed up) when produced by different servers whereas it makes no sense to combine statistical scores such as t-score or mi-score.

Let us now consider the computational constraints inherent to the general organisation of the ELAN network and the associated conclusions that can be driven with regards to the implementation of statistical scores.

3.4.2 From the client side

- The user does not have to know necessarily where the resources he has selected come from. He should thus be able to choose a statistical test transparently on his selection;
- for the sake of efficiency, the concordance lines may not have been fully loaded on the user client and thus the contextual frequencies cannot be computed here.

Conclusion: the user client will solely visualize results which will be provided to it by its local server. It may only sort and threshold the corresponding tables (e.g. word x decreasing score x position) in an interactive mode.

3.4.3 From the local server side

- The local server does not have all the data needed to compute a given score and it would be nonsense to download the whole concordance lines from remote servers to the local servers;
- on the contrary, it has the best position to combine information issued by remote servers to compute statistical scores;

Conclusion: the local server combines frequency tables which he queries to remote servers and provides the client with the proper statistical results.

3.4.4 From the remote server side

- The remote server can easily provide reference frequencies and reference corpus size for its own databases (which could even have been computed beforehand). For more specific reference frequencies, he should be provided with the list of resources from which these are to be computed;
- the remote server can also compute contextual frequencies and context size if it is provided with the proper references associated with the concordance lines (element identifiers or node word position + distance);

Conclusion: the remote server computes all the basic frequencies which are needed for a given statistical score according to the queries it receives from the local server.

3.4.5 The protocol retained within ELAN

1. Query :

```
<!DOCTYPE sil SYSTEM “sil.dtd”>
<sil sid=”leiden” type=”elanQuery” version=”0.5”>
   <uid type=”user”>
   ...
   </uid>
   <elanQuery>
      <wordFreqQuery>
         <word>example</word>
      </wordFreqQuery>
   </elanQuery>
</sil>
```

2. Results set:

```
<!DOCTYPE sil SYSTEM "sil.dtd">
<sil sid="pisa" type="resultset" version="0.5">
   <uid type="user">
```

```
…
</uid>
<rs>
   <meta>
      <attr name="size">3</attr>
      <attr name="totalSize">10</attr>
      </meta>
   <result type="wordFrequency">
      <record idno="FRW1">
         <attr name="word">example</attr>
         <attr name="frequency">10</attr>
      </record>
      <record idno="FRW25">
         <attr name="word">example</attr>
         <attr name="frequency">24</attr>
      </record>
      <record idno="FRW37">
         <attr name="word">example</attr>
         <attr name="frequency">41</attr>
      </record>
   </result>
</rs>
</sil>
```

3.4.6 Implementation

The preceding protocol for statistical information exchange will be comprised in the Java interface provided to the different linguistic resource server. The API for queries and results sets is available at the ELAN software site : http://www.loria.fr/projets/MLIS/ELAN

4 Organisation on the ELAN network (Deliverable 3.2.1)

(flow of information and management)

4.1 Organisation

The network architecture of ELAN is based on three major actors:

- The Users (or Clients) which have been described in the preceding sections.
- The Network Management Unit.

  The NMU should be considered as the heart of the network, that is, the NMU allows to link all the servers which are connected to ELAN. The NMU also maintains a database of information related to these servers (i.e., names, addresses, description, etc.).

  A simple Network Management Unit (NMU) must be implemented (see Figure 5, “Overall ELAN network architecture,” on page 23).
- The Linguistic Resources servers.

  As mentioned before, this is a set of independent but associated LR servers. In particular, each server is an autonomous unit bearing its own linguistic data. These servers can act as :

  - linguistic servers and answer queries sent to them.

  - brokers which broadcast the queries to all referenced and selected servers of the Network. A server act as a broker for the users connected directly to him, that is, when he is in the position of the login server.

Given that each single server has to:

- know about the list of servers affiliated to the network, and
- be certified in some respect before being connected to the network.

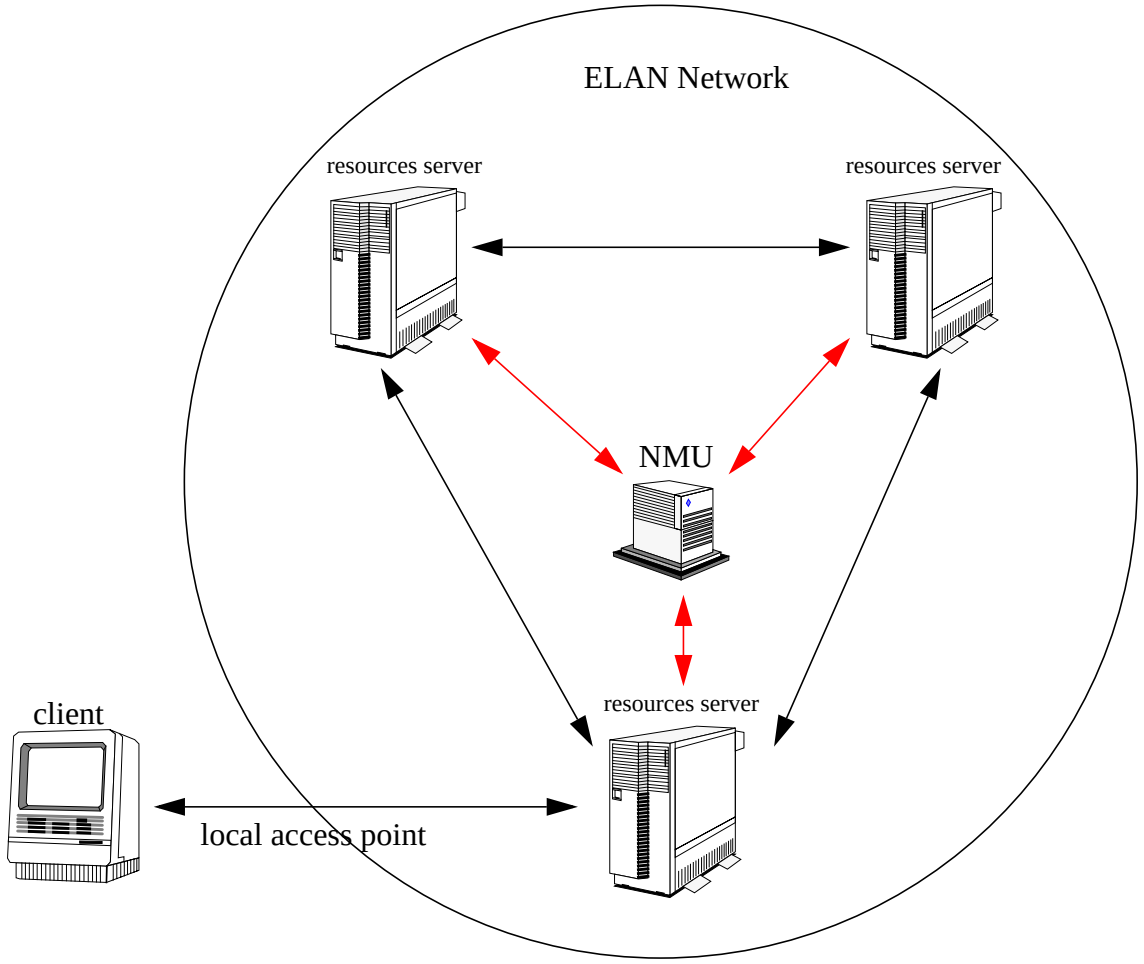


Figure 5. **Overall ELAN network architecture**

4.2 Clients, Servers and NMU communication

Two main networking protocols are used within the ELAN network:

- HTTP for client/server and server/server communications
- CORBA for NMU/servers communications
- For server/server communication, Java server-sided components known as servlets will be used (see Figure 6, “working of Java Servlets,” on page 24), while the communication between the LR servers and the NMU will implement CORBA, a distributed object paradigm (see Figure 7, “general CORBA client-server architecture,” on page 25)

4.2.1 Servlets

Servlets provide a Java solution to the problems usually associated to the development of server-side components to extend the functionalities of an HTTP server. Thus, it is a technology that can be used to build network systems based on the usual client/server paradigm.

Servlets are a good alternative to the CGI technology (*Common Gateway Interface*), and are a good answer to the issues associated to them. Among other problems, we can say that CGIs are not evolutive, they suffer of a lack of performances, etc. These problems are often linked to the fact that CGI are usually implemented in scripts languages like PERL... Servlets are implemented in Java, with all the advantages associated with this language, in particular portability.

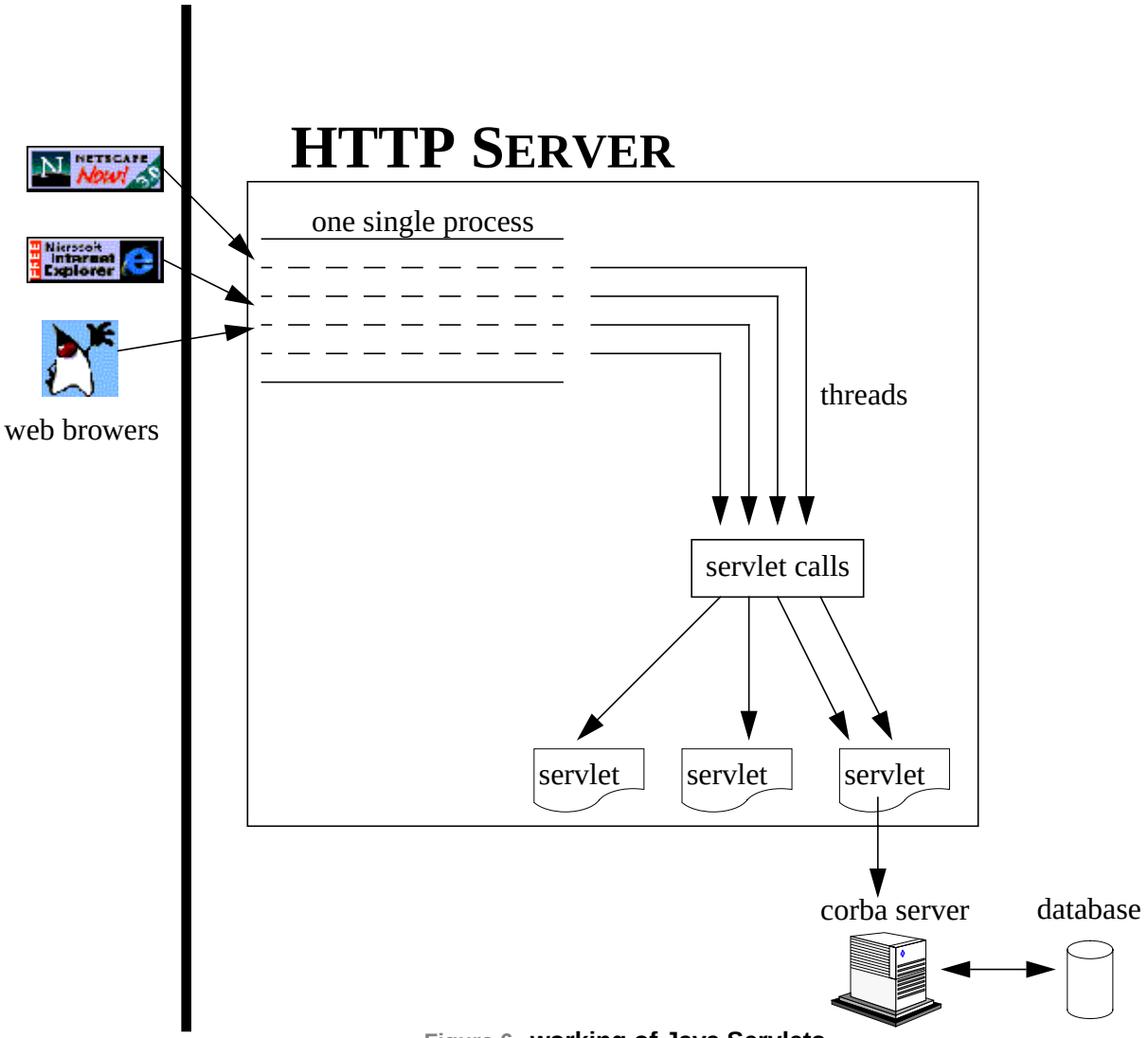


Figure 6. **working of Java Servlets**

4.2.2 CORBA

CORBA is a technology from the OMG (Object Management Group) that can be used to specify and develop client-server applications using distributed objects.

In CORBA, the server implements objects that can be made available to clients wherever they are located on the network. The server “publish” its objects using a specific protocol, so these objects become “distributed objects”. A client can then get references on these objects and use them as if they were local.

Three main components can be identified in a CORBA system :

- the IDL (Interface Definition Language)
- the ORB (Object Request Broquer)
- the Services (Naming, Security, Events, etc...)

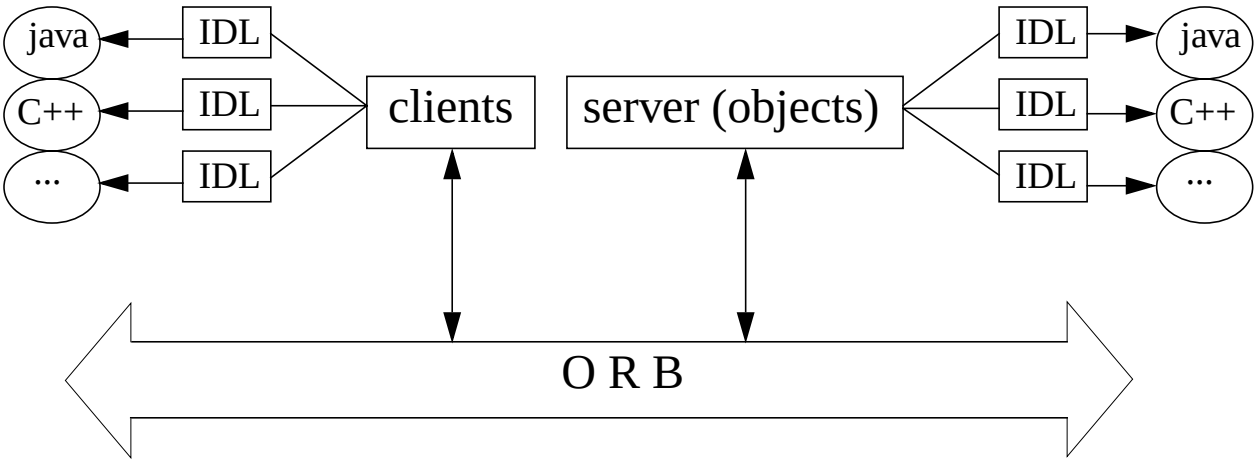


Figure 7. **general CORBA client-server architecture**

a) The ORB (Object Request Broker)

The ORB is the main component of the CORBA architecture. It can be seen as the “bus” through which the objects are being distributed. To make the communication as transparent as possible to the application, the ORB must take care of several issues that include:

- find and instantiate objects on distant computers
- invoke the methods of a distant object
- transmit parameters when invoking methods, and be sure the data types are coherent between clients and servers.
- publish meta-information for the various objects available in an object repository
- etc.

b) IDL (Interface Definition languages)

CORBA can be used with several differents programming languages. This is one of its most interesting characteristic. Now the representation of data on these languages can be very different, so we need a mechanism to specify objects interfaces independently from their implementation. This is the purpose of the IDL language.

IDL is an object-oriented language to specify distributed objects interfaces, independently from the language they are going to be implemented in. For example, in a CORBA architecture, this allows a Java client to use an object implemented in C++ on a server.

c) CORBA services

Depending on the implementation of the CORBA specifications, several services are available to the developer. The main services that should be available with all implementations are:

- the Naming service : This service is used by client to get references on objects. Servers associate a name in a specific name-space to all objects they publish, and client can retrieve objects references using the objects names that have been referenced.
- the Security service : Like in every distributed architecture, security in CORBA is a big issue. This service purpose is to control access to objects or methods of objects and be sure only authorized client can get references on distributed objects and call their methods.

- the Events service : This service specify and implements an alternative to the classic client-server model. It is build on the publisher-consumer model. In this model, a server generates an event with its associated data and clients consume the event when they get it.

4.3 The Network Management Unit (NMU)

4.3.1 Role of the NMU

As stated in section 2 (General Network Organisation), the Network Management Unit (NMU) is the only persistent link between the different servers affiliated to the ELAN network. It has thus the following functions:

- it maintains the list of servers affiliated to the network, together with a general profile for each of them (containing the languages dealt with by the server, the categories of document it may provide (tagged/untagged texts, lexicons, ...). To this end, each server is described by its name, its address (URL), and a flag describing its current status;
- it should be able to answer any query from each server belonging to the network when they have to know the list of available servers, the information attached to them and their current status;
- it updates the local database of the LR servers (SDB) through the network, each time a change has been made to the network description or when something has changed on the part of a given server;
- it is accessible through a simple form-based GUI, to allow the person responsible for the network (the "network master") to add a new server, modify the characteristics of a given server and/or disconnect a server from the network. At this point, no automatic processing will be made form LR servers to NMU to ensure full control of the quality and coherence of the information ;

Because of its central position in the ELAN architecture, the NMU is subject to several constraints, which are to condition its implementation. In particular, it has to be reliable as to its content since it is in charge of providing each server with the proper information to make it communicate with the others.

4.3.2 NMU / SERVERS communication

As indicated above, communication between the NMU and the LR servers will be implemented by using CORBA. As in all CORBA implementations, client/server communications are described by using the “Interface Definition Language” – IDL -.

Several implementations of the CORBA specification exists. Because it is free and completely implemented in the Java language, we decided to use the JacORB broker for the implementation of the ELAN NMU. Informations about JacORB are available at :

http://www.inf.fu-berlin.de/~brose/jacorb

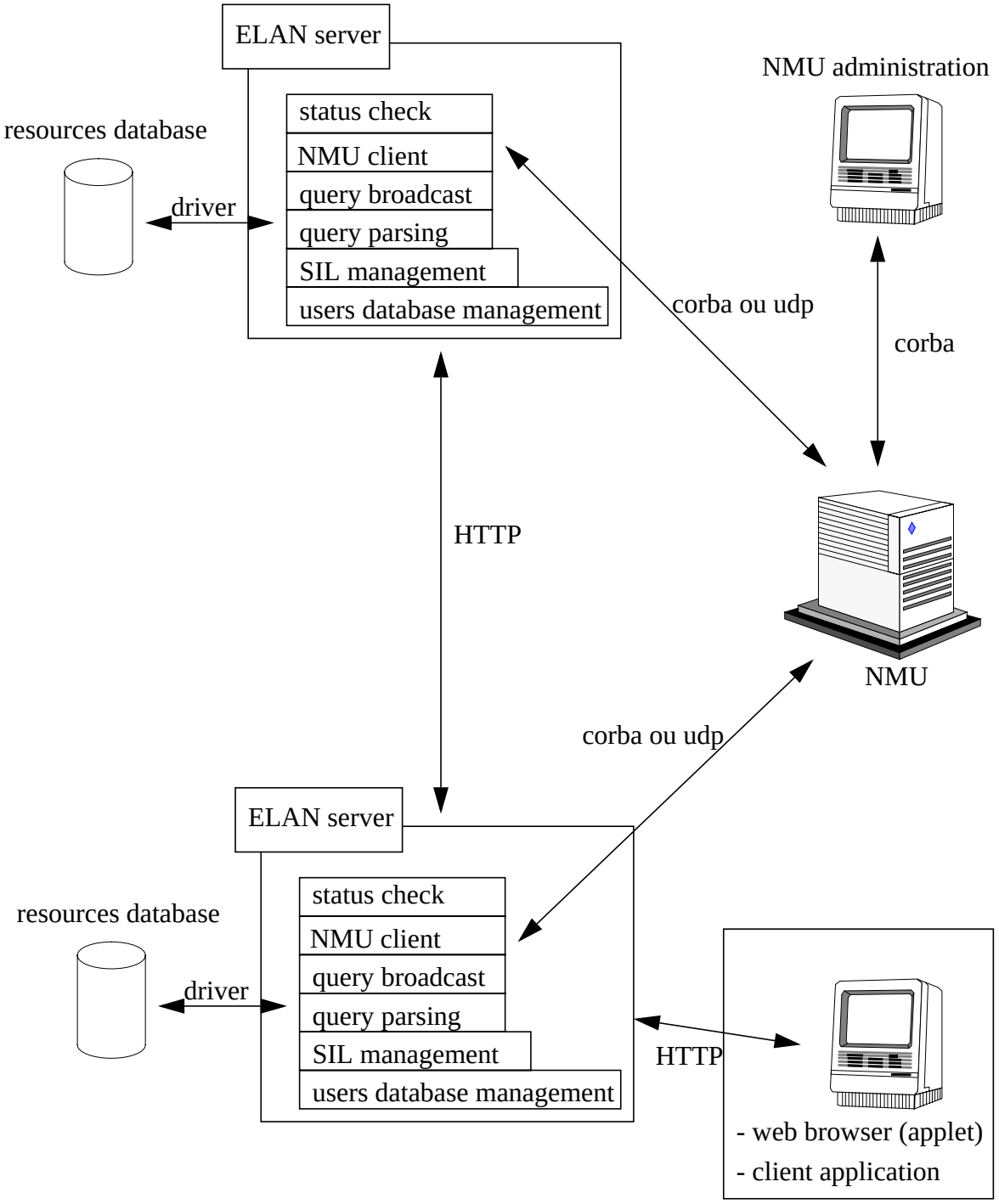


Figure 8. **Detailed architecture of the ELAN network**

4.3.3 IDL Description

IDL is a language used to describe client/server interaction. On the server side, IDL allows to describe the set of objects used by the server, as well as, their associated methods. On the client side, IDL is used as a guide which indicates what is the procedure that a client should follow to communicate with a server.

In ELAN, we suggest to use a single CORBA object – NMUServer -, which allows NMU/Servers communications. This object will be published on the Corba bus using the JacORB Naming Service.

It's name is "NMUServer" within the "ELAN" naming context. Every resource servers will have a retrieve a reference on this object to get information about the network or a specific server (addresses, status, etc.)

Below is the IDL description of the NMU server object.

```
module nmu
{
   enum Status {online, offline};
   typedef sequence<string> serverList;

   interface NmuServer
   {
      boolean add(in string name, in string address, in string info);
      boolean remove(in string name);
      boolean isUpdate(in string name);
      serverList getList();
      string getAddress(in string name);
      string getInfo(in string name);
      Status getStatus(in string name);
   };
};
```

4.3.4 Firewalls

As a "typical" TCP/IP implementation, JacORB must reserve some ports, on the one hand to implement the ORB and, on the other one hand, to allow client / server communication.

Unfortunately, when using JacORB one can not select the ports to be used. When JacORB is started, it searches a free set of ports which are selected randomly. If ELAN would work by using an Intranet framework, this would not be a problem. Obviously, ELAN works on an Internet basis, so several LR servers may be part of a sub-network which is access-protected, because of evident security reasons, from the Internet. This protection is the firewall. The firewall may forbid the use of some ports, so in most cases client / server communication may not be possible. That is, JacORB may select a set of ports which may be accessed inside the sub-network, but not from the outside.

An evident solution would be to authorize all accesses to the computer where the NMU resides. Nevertheless, this solution should be quite hard to implement, because of security access restrictions that must be maintained in a sub-network.

So, in order to test a first prototype we have decided to develop an intermediate tool whose function is to allow all clients to communicate with the NMU, while respecting all security restrictions used in the sub-network. This tool is currently being implemented as a "demon" which waits for an UDP request issued by the NMU. When the request arrives, the demon connects to the NMU. When the connection between the NMU and the demon has been done, the demon has a reference on the distributed object NmuServer, that is, the demon becomes a client of the NMU server.

By using the UDP protocol, we can select one single fixed port to receive all requests. Port number 1130 has been selected (see Figure 9, "Solving a firewall problem," on page 29). Obviously, the demon must be executed inside the NMU's server sub-network.

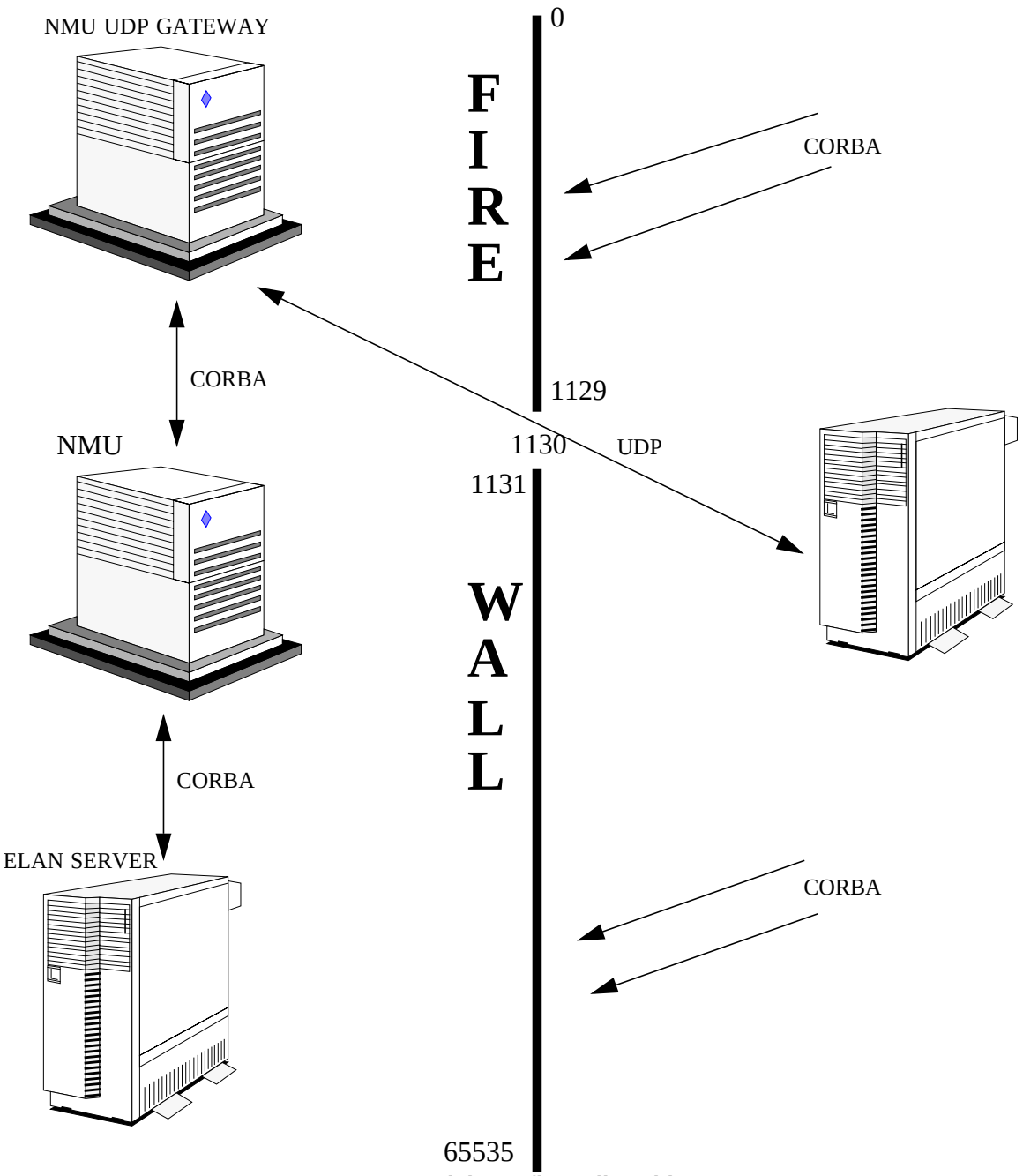


Figure 9. **Solving a firewall problem**

4.4 Server-side organisation

4.4.1 Introduction

This chapter presents the communication scheme between Elan servers and clients. This includes :

- the general communication scenario.
- the API for querying the servers and transmitting the results sets to the end user.
- the development of a driver to implement the communication between a server and a specific database system.

A schema representing the complete HTTP communication scheme in ELAN can be seen in Appendix. It shows the various components and the data streams.

4.4.2 Protocols and languages used in ELAN

a) The XML level

All information flowing through the network will be encoded in XML. The encoding system is defined by the ELAN's DTD (inspired from the Silfide Interface Language DTD). This DTD allows to encode user's working spaces (WS), requests on meta-data (QL), results for these requests (RS), and all information related to users (UI).

b) The MIME level

Because of efficiency, in particular in the framework of broadcasting requests from a server to another, not all needed information is not presented in XML documents. In general on the Internet, data like sound, images, etc.,. are encapsulated on a MIME layer. MIME allows to assign a data type (i.e., content-type), as well as to encapsulate heterogeneous data on a single data flow called MIME/Multipart: on a single channel, one single connection is enough in order to exchange multiple data.

c) The HTTP level

ELAN uses the HTTP protocol (like the whole Internet) as the main communication protocol. Using HTTP is extremely simple (two single Java instructions allow to open an HTTP connection. However, http is a "stateless" protocol. This implies that each time a new page is loaded, the user is effectively disconnected from the server and it keeps no information allowing to know who was the user and what he was doing.

1. The global scenario is:
2. Open the client/server connection;
3. send a request from the client to the server;
4. send a response from the server to the client;
5. close the client/server connection.

Thus, even after logging into a site, each page accessed must pass the user name and the password back to the server to verify the user's right to access the page. The client application (the browser) and the server application (the Web server) have no concept of local variables, local method calls, or objects. This problem is solved in part by using MIME/Multipart.

4.4.3 General Scenario

a) Connection

The user connects to the network through one of the registered servers. He can connect either with a client application (he will need to provide the address of the server) or with an applet. The server the user is directly connected to is referred further as the "login server". This server will act as a gateway, and all data exchanged between the client and the network (the login servers or other linguistic servers) will pass through it.

b) Scenario

Once the user is connected to one of the ELAN servers, all the network traffic (queries and results) between this specific client and the network will pass through the "broadcast" servlet on the login server. The main purposes of this servlet are to :

- broadcast a query to all servers selected by the user.
- broadcast a query to all servers appearing in a resources basket.
- merge the results from the differents servers into one results set to be transmitted to the user (see SuperResultSet below).
- maintain a cache memory of the results associated with the queries, until they are explicitly closed.

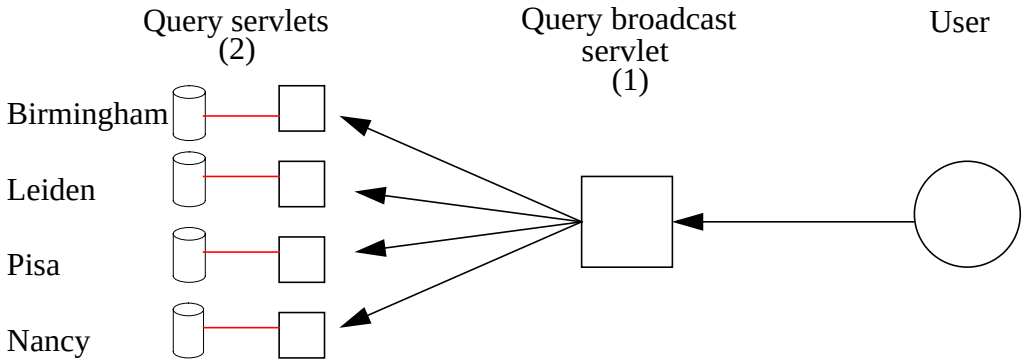


Figure 10. **broadcast of a query**

The user sends a query to the login server's broadcast servlet which will act as a broker and transmit it to the other selected servers.

Considering that the communication is implemented above the HTTP protocol, and that we have a general API with a built-in cache system, allowing connection to a server and sessions management, the java code to connect to a server would be :

```
QueryConnexion co = new QueryConnexion(URL);(1)
QueryStatement st = co.getStatement()(2)
QueryResultSet rs = st.sendQuery(CQL);(3)
```

(1) : QueryConnexion purpose is to initiate a communication with a server, by providing its address and the service we want to connect to, in the standard URL format used for HTTP connections.

(2) : Once the connection is instantiated, we ask for a "statement" object, that handle the sessions and cache systems. This class will be used to send a query, or close the session.

(3) : Using the statement object, a CQL query can be send. The sendQuery call will return a results set object, which can be used as a standard Java Enumeration to retrieve the results.

This scenario is used both for the “Client”/”Login Server” connection and the “Login Server”/”Other ELAN servers” connections. Both cases are illustrated in the following examples :

- The user applet connects to the Broadcast servlet, send a query(CQL) and gets the results set as an Enumeration.

```
QueryConnexion co = new QueryConnexion("http://elan.nancy.fr/BroadcastServ-
let");
QueryStatement st = co.getStatement()
```

```
QueryResultSet rs = st.sendQuery(CQL);
```

- The Broadcast servlet connects to the birmingham query servlet, send a query(CQL), gets the results set as an Enumeration and read the 10 first results.

```
QueryConnexion co=new QueryConnexion("http://elan.birmingham.uk/QueryServ-
let");
QueryStatement st = co.getStatement()
QueryResultSet rs = st.sendQuery(CQL);
for(int i=0;rs.hasMoreElements() && i<10;i++)
   Object o = rs.nextElement();
```

The main purposes of st (instance of the QueryStatement class) are :

- maintain a consistent connection between all components (user/BroadcastServlet and BroadcastServlet/QueryServlets) through a unique session ID (1 query=1 session id).
- store the result sets in a cache memory for as long as the session is valid.

A session remains valid until the user closes the connection or a time-out exception occurs.

The query is sent through the `sendQuery` method of the `Statement` class, which will return an Enumeration to read the entries from the results set, through the usual `hasMoreElement()` and `nextElement()` methods.

The nexElement() method returns one result at a time. It will do all that is necessary to retrieve the result from the results set, which means :

- for the client applet or application :

  - returns the result directly from the local cache, if its there.

  - contacts the login server and retrieves the next results.

- for the login server :

  - returns the next results directly from the local cache, if they are there.

  - contacts all servers involved in the current session, and retrieve the next results before to return them to the client.

The client connects to "Broadcast servlet" on the login server.

The login sever connects to the "Query servlets" on the resources servers.

As you can see in the general architecture schema at the end of this document, there are two levels of cache memory. One on the client applet or application and one on the login server.

c) Super Result Set

The Broadcast servlet will merge the result sets from all the elan servers that answered the user's query in a super results set. It is a class which purpose is to call the next() methods of the servers specific results sets, merge the data in a *convenient* way (i.e. build a new results set), and send it to the user.

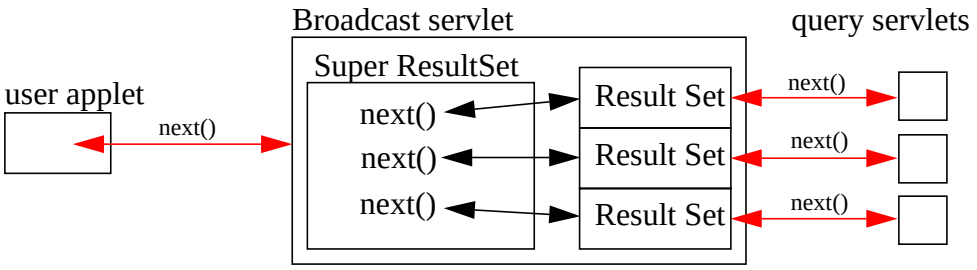


Figure 11. **broadcast of a query and merging of the results in a super results set**

d) session ID

It is possible that the result of a query is huge and cannot be sent at once back to the user.

In that case, a subset of the complete result set should be sent, and the remaining parts should be sent only on user request. Now HTTP is a connectionless protocol, which means that we need a mechanism to specify that a first sendQuery(...) call and the following next(...) calls belong to the same context.

For example, when a user asks his login server for the next entries of a results set, the server needs to identify the user and figure out to which former query it refers.

4.4.4 APIs

a) QueryConnection

- QueryStatement getStatement()

  returns a new QueryStatement object (i.e. a new session)

b) QueryStatement

- QueryResultSet sendQuery(String query)

  send a CQL query
- QueryResultSet sendQuery(String query, String basket)

  send a CQL query with a resources basket (in XML format)
- QueryResultSet sendQuery(String query, Vector servers)

  send a CQL query and a list of server (sid) to which the query should be sent
- String getSessionID()

  returns the session id
- void close()

  close the session

c) QueryResultSet

- booelan hasMoreElements()

  returns true if there's still results to be retrieved in this results set
- Object nextElement()

  returns the next result in this results set

- void close()

  close the session

- String getSessionID()

  returns the session id

- int size()

  returns the total number of results in this result set

- int[] getErrorCodes()

  returns the error codes in this results set

- String[] getErrorTexts()

  returns the error texts in this result set

- String[] getErrorSids()

  returns the ID's of the servers that issued errors

- String getErrorString()

  returns a string formatted with all errors, to be displayed to the user

4.4.5 The ELAN driver

Since the implementation of the linguistic databases is specific to each server, we need a mechanism to translate the CQL query into the language corresponding to the local database.

For this purpose, a common interface, the `QueryHandler` interface, is to be implemented by each server connected to the Elan network. This is the only part of the architecture to be developed specifically for a proprietary resources interrogation system. It will allow the communication between this system and the ELAN network.

In the ELAN scenario, the QueryServlet receives a query and transmits it to the QueryHandler driver which will connect to the local resources database, handle the query, and return the result as an ElanResultSet object.

a) the QueryHandler Interface

An ELAN driver must implements the following interface :

```
public interface QueryHandler
{
   // starts a new query on the whole corpus (typically to get references to
   // the documents matching the query and build a basket)
   public ElanResultSet executeQuery(String sessionID,
                  String CQLQuery);

   // starts a new query with the documents referenced in the basket
   public ElanResultSet executeQuery(String sessionID,
                  String CQLQuery, String basket);

   // gets the next entries of the result set if the context was too big to
   // send it all at once with the executeQuery call. executeQuery(...) with
   // the same sessionID should have been called before.
```

```
        public ElanResultSet next(String sessionID);

        // close the session identified by the session number "sessionID"
        public void close(sessionID);
    }
```

An example of implementation of a “fake” driver is available in Appendix.

b) Parsing an ELAN Query

The ELAN query is transmitted to the driver as a string containing an XML SIL document. It has to be parsed into an "ElanQuery" before to be used. See Appendix H. “Sample code of a “fake” Query-Handler driver” on page 91 to see how to do this.

The ElanQuery interface provides access to the ELAN query and allows to easily extract information from it. Its main methods are :

- short getQueryType()

  returns the type of the query. It can be :

  QUERY_WORDFREQ : word frequency query

  QUERY_WORDLIST : word list query

  QUERY_CONTEXT : context query

  QUERY_STATS : statistic query

  QUERY_CONC : concordance query

  QUERY_HEADER : header query

- String getHeaderQuery()

  returns the header query (CQL) or null

- String getTextQuery()

  returns the CQL query

c) Parsing a resources basket

A "SILBasket" class is to be used to parse and extract information from the basket XML string that can be sent with a query. It can also be used to build a new basket object.

A basket is basically made of "resources" object, which are made of a SID (the server from which the resource come from), a IDNO (a unique resource identifier on the sid server) and eventually a set of display attributes (title, author, etc.).

The main methods of the "SILBasket" class are :

parsing :

- Resource getResource(String sid, String idno)

  returns the resource identified by this sid/idno couple, or null

  Resource.getSId returns the resource sid

  Resource.getIdno returns the resource idno

- boolean containsResource(String sid, String idno)

  returns true is the resource identified by sid/idno is in this basket

- int getSize()

  returns the number of resources in this basket

- Enumeration getResources()

  returns an Enumeration for browsing through the resources in this basket.

building :

- boolean addResource(String sid, String idno)

  add a resource to the basket (without any attributes)

- boolean addResource(String sid, String idno, Hashtable attrs)

  add a resource to the basket with some display attributes in a hashtable

- Resource removeResource(String idno, String sid)

  remove the resource identified by the sid/idno couple from the basket

d) Building an ELAN results set

The ElanResultSet Interface can be used either to extract results from an existing results set or to build a new one. It allows easy manipulation of XML results sets documents that conforms to the DTD in Samuel Cruz-Lara's document "Result Sets in ELAN".

The ElanResultSetImpl class implements the ElanResultSet interface.

An instance of ElanResultSetImpl should be returned by the executeQuery(...) and next(...) methods of the QueryHandler implementations.

The methods in the ElanResultSet interface are :

Parsing :

- Enumeration getRecords()

  Initializes a Result Set (as an enumeration)

- getSize()

  returns the number of results in this set

- getTotalsize()

  returns the total number of results matching the initial query

- String getResultType()

  returns the type of the results set

- toString()

  returns the results set as a string (XML format)

- int[] getErrorCodes()

  returns the error codes in this result set

- String[] getErrorTexts()

  returns the error texts in this result set

- String[] getErrorSids()

  returns the ids of the servers that issued the errors

Adding results :

- void addAttribute(String name, String value)

  add an attribute to a results set
- void addRecord(String sid, String idno, String data, Hashtable attributes)
- void addRecord(String sid, String idno, String data)
- void addRecord(String idno, String data, Hashtable attributes)
- void addRecord(String idno, String data)
- void addRecord(String idno, Hashtable attributes)

  add a single result element to the results set (a record element)

  - sid is the server unique identifier on the network

  - idno is the resource unique identifier on the server

  - data contains raw data as a string or null

  - attributes contains a set of attributes stored in a hashtable.
- void addRecord(String record)

  add a new record element to the results set
- void setTotalSize(int ts)

  set the total number of results matching the query
- void setResultType(String type)

  set the result type

Adding Errors :

- void addError(String sid, int errorCode, String message)
- void addError(String sid, int errorCode)

  add an error message to the results set

  - sid is the server unique identifier on the network

  - errorCode is the code of the error (-1 is the generic error code)

  - message is an optional message to be displayed to the user

4.4.6 Driver installation

The server preferences file (`$HOME/xsilfide/silfide/preferences/ServerPrefs.xml`) contains a set of attributes and their values. This file is read by the server when it starts, and should specify the name of your driver (complete package path) in the "driver" attribute :

Note that your `CLASSPATH` variable must contain a path to your driver implementation, so that the Java class loader can locate it.

Below is an example of a `ServerPrefs.xml` file, with the driver attribute (in bold) pointing to the `QueryHandlerExample` driver in the `fr.loria` package.

```
<prefs>
   <pref name="status" value="online"/>
   <pref name="name" value="chris2"/>
   <pref name="nmua" value="godefroy.loria.fr"/>
   <pref name="nmup" value="1130"/>
   <pref name="driver" value="fr.loria.QueryHandlerExample"/>
```

```
</prefs>
```

4.5 Security Aspects

4.5.1 Introduction

Two kind of users can connect to the ELAN network : guest users or registered users. This mean that we need a mechanism to identify users and their right to access specific resources. For this purpose, a user database management system providing password protected connection and passwords encryption has to be developed, as well as a secure transaction layer to protect the communications between servers.

Two main security threats can be identified in the Elan architecture regarding secure and protected communications. We should :

- Control access to the NMU and especially to its administration methods.
- Secure the communications between servers (signature and/or encryption).

This basically means that we need to :

- Identify trusted clients on the NMU.
- Identify users on servers so only the authorized ones get access to protected resources.
- Make sure that no user can connect as someone else, in order to get specific privileges.
- Make sure that intercepting the data exchanged between servers is useless.

Our purpose in the next sections is to present differents methods that can be used to identify users at connection and secure transactions taking place in the network, either between resources servers or between these servers and the NMU.

4.5.2 Connecting to the ELAN Network as a registered user

To connect as a registered user, a login name and password has to be sent over the network so the server can identify the user and eventually allow access. Since it is pretty unsafe to send a password unencrypted over a network, this requires a mechanism to encrypt the password and ensure that intercepting it would be useless. This is the purpose of the OTP system.

OTP is for One Time Password and means basically that a password can be used only one time before to be invalidated. Thus, both the server and clients have to recalculate a new password for every new connection.

The password is calculated with two parameters : the secret key and the challenge key. The secret key is provided to both the server and the clients, but never transmitted upon the network, so it cannot be intercepted. The challenge key is a counter, which is incremented by the server after every connection. Before connecting, the client ask the server for the challenge key and with its secret key calculates the password to be transmitted. The server does the same and if the calculated passwords match when the client try to connect, the access is granted.

Controlling access to specific resources on specific servers involve the development of a real user database where users belongs to groups.

For example, the "guest" user would belong to a "guest" group with minimal access rights.

A registered user could belong to a group associated with a specific project and be allowed to access the corresponding resources.

The specification of such a user database management system is not this document purpose though...

4.5.3 Securing access to the NMU

Some of the methods provided by the NMU IDL interface are used for administrating the NMU database, and so have to be protected in a way a fake client implemented by some untrusted people could not execute them, and so trash the NMU database.

Therefore, we have to implement a system allowing only trusted clients to call these methods.

Two options are available for securing access to the NMU. The first one, the CORBA Security service, is implemented as a standard CORBA service. The second one is based on a password system.

a) The CORBA Security Service (CSS)

As a distributed computing architecture, CORBA provide a security service, based on the Security Reference Model (SRM). The Object Management Group (OMG) identifies the following security threats to a CORBA System :

1. An unauthorized user of the system gaining access to information that should be hidden from him.
2. A user masquerading as someone else, and so obtaining access to whatever that user is authorized to do, so that actions are being attributed to the wrong person.
3. Security controls being bypassed.
4. Eavesdropping on a communication line, so gaining access to confidential data.
5. Tampering with communication between objects : modifying, inserting and deleting items.
6. Lack of accountability due, for example, to inadequate identification of users.

In Elan, we are essentially interested in the fifth point. We want to allow access to administration methods of the NMU only to the administration clients. The other clients (the Elan servers) should never gain access to these methods, neither should any untrusted client. The following features are included in the countermeasures provided by the CSS system :

1. Controlling access to :
   - an IDL interface.
   - a subset of the implementations of an IDL interface.
   - interface operations and collections of interface operations.
2. Identification and authentication of :
   - the user to the CORBA system.
   - the client to the target.
   - the target to the client.

Though CSS provides exactly the kind of protection we need in Elan, it is not distributed with all CORBA Brokers... On the contrary, most of the free CORBA brokers implemented in Java implements a very thin subset of CORBA services, not including the security service.

b) Password protected access to the administration methods

The second option consists of providing a crypted password as a parameter to any of the administration methods defined in the IDL interface. As a consequence, the code would actually be executed only if the right password is provided.

The two methods of the NmuServer class that should be protected this way are *add* and *remove*, which respectively allow adding and removing of Elan servers. They could be defined in the IDL this way :

```
boolean add(in string password, in string server_name, in string
server_address, in string server_info);

boolean remove(in string password, in string server_name);
```

To ensure that intercepting the password is useless to a third person, an OTP system can be used, and a simple method can be added to the NMU IDL interface to allow the transmission of the challenge key :

```
string getChallengeKey();
```

4.5.4 Securing Server-Server communications

The requests made to the Elan servers contains identification and authentification information that the servers use to determine whether the user is allowed to access the data he wants to.

The point is if untrusted users get this information (by listening the network for example), they can modify the request by hand and connect to the network as a trusted client, receiving access to the corresponding resources.

There is no way to ensure none would intercept the requests when they are transmitted from one Elan server to another, but we can make this information useless with encryption or signature technologies.

a) SSL (Secure Sockets Layer)

The protocol is composed of two layers. The lowest, on top of some reliable transport protocol like TCP/IP, is the SSL Record Protocol, used for encapsulation of various higher level protocols. One of these, the SSL Handshake Protocol, allows servers and clients to authenticate to each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data.

SSL is well known on the Internet since it's usually used to secure transaction with the HTTP protocol, which in the above architecture is the application protocol.

The point is that not every HTTP servers implements this protocol, and that in Elan, we don't really need encryption but only authentification and integrity check of the messages being send between servers.

b) S/MIME

S/MIME (Secure/Multipurpose Internet Mail Extensions) provides a standard way to send and receive secure MIME data. Based on the MIME standard, it provide the following security services :

- authentication
- message integrity
- non-repudiation of origin, using digital signatures
- data security using encryption

In Elan, the CQL is already encapsulated in a MIME MULTIPART document, which means it would be very easy to implements this protocol using probably only the signature feature.

An implementation of the S/MIME standard in the Java programming language is available at http://jcewww.iaik.tu-graz.ac.at/

c) PGP Signature

PGP (*Pretty Good Privacy*) is a public key encryption system. This means that it does not depend on the encryption key being kept secret for its security. The public key is used to encrypt messages and is distributed to any instance that would like to send a secure message. A separate key, known as the secret key, is used to decrypt messages. PGP can be used either to encrypt or only sign messages. Since encryption using PGP is forbidden in some countries including France, we should not use this feature in the Elan network, but signing a message using this system seems to be a good solution. PGP encryption and signature features have been coupled with the MIME standard and is known as the PGP/MIME system.

The ELAN Client user's guide

Deliverable D3.1-2

Project Number	**MLIS-121**
Project Title	**European Language Activity Network (ELAN)**
Deliverable Number	D3.1-2
Work Package ID	WP3
Contractual date of delivery to EU	31 december 1999
Actual date of delivery to EU	17 décembre 1999
Deliverable Title	**The ELAN Client user's guide**
Authors	*Christophe de Saint-Rat*

Abstract

The ELAN project is a distributed language resources system, offering access to existing resources to their potential users throughout Europe. In order to serve the electronic multilingual resource market, our task is to specify and elaborate a network of inter-connected resource servers. This document defines the technical specifications of each nodes that form the ELAN network: the user interface of the ELAN client; the resource servers and the network management unit.

Keywords

Language resources, Distributed systems, Network, User interface

Download ELAN client installers

MacOS client installer :

http://www.loria.fr/projets/MLIS/ELAN/private/deliverables/elan/ElanInstallMacOS.tar.gz

Unix client installer :

http://www.loria.fr/projets/MLIS/ELAN/private/deliverables/elan/ElanInstallUnix.tar.gz

Windows client installer :

http://www.loria.fr/projets/MLIS/ELAN/private/deliverables/elan/ElanInstallWindows.tar.gz

5 Introduction

The ELAN network's purpose is to merge into one virtual database various linguistic resources spread among different systems, and provide a common way to query them and retrieve results.

To do so, an easy to use client application with a user-friendly interface has been developped.

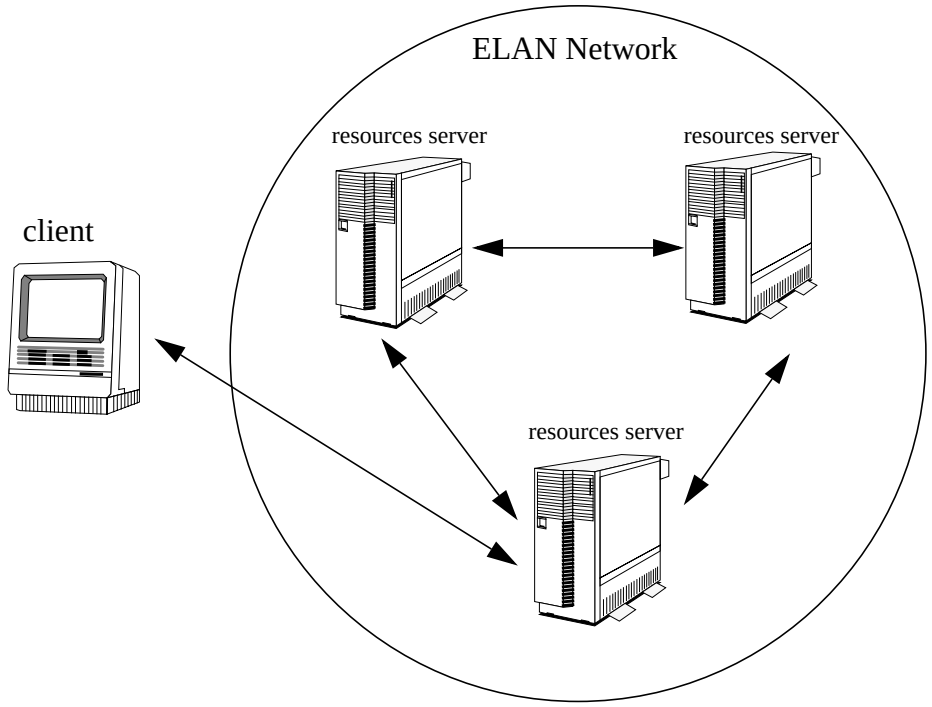
Using the ELAN network implies to follow a few steps :

- Connect to one server of the network either using the application client or the applet available on the login server's homepage.
- Select the servers to be queried.
- Select documents or corpora to work with by :

  - querying the servers

  - choosing resources

  - adding them in a "resources basket".
- Perform content queries on the selected documents (currently only corcordances queries are supported)



6 Connection to a server

The ELAN client is entirely developped in the Java programming language, which means that you can use it with any Java enabled operating system and computer. This includes a PC with Windows 95/98 or NT, PC with Linux, a Macintosh with MacOS or LinuxPPC, or any UNIX workstation.

Whatever system you are using, you will have to install a Java Virtual Machine, as well as the Swing Components (used for the user interface design).

Two configuration can be used :

- JDK 1.1.6 minimum with SWING 1.1.x
- JDK 1.2

If they are not already installed on your system, you can find these components as well as the installation instructions at Sun's web site : **http://java.sun.com**

6.1 Connection through the applet

If you have a swing enabled web browser, you can start the ELAN client as an applet directly by connecting to one of the servers referenced on the ELAN network.

Just point your browser to one of the server by entering its URL and follow the instructions on the page.

You will find a link to start the applet as a guest, or you can enter a login name and password to connect as a referenced user. Note that in that case you must first have been referenced on this server as a member. Please contact the server administrator to do so.

6.2 Connection with the client application

As an alternative it is also possible to download and install an ELAN client on your computer. The main advantage compared to the applet is that the connection will be faster since the required classes are locally installed on your computer, and don't need to be downloaded everytime.

This is also the only way to work with ELAN if you don't have a Swing enabled Web browser.

To dowload the clients installers, point your browser to one of the ELAN servers and download the client suitable for your system. Three client installers are available for Windows 95/98/NT, Macintosh and Unix systems.

When you start the client, you will be prompted to enter the login server address, and eventually a login name and password if you want to connect as a registered user. Leave these fields blank if you want to connect as a guest.

7 The Workspace

In the ELAN system, the user should be able to save different kind of information or data for later use. A registered user probably does not want to select again his client interface language everytime he connects to the network, or the servers or resources he usually works with.

As we see, there is a need for a private environment associated to each registered user. This is what is called the Workspace.

The Workspace is saved on the user's login server, and reloaded everytime he connects to the network, so his private environment is automatically restored from one session to the other.

A Workspace usually contains :

- the environment preferences (see "the preferences panel" below)
- the list of selected servers (see "selection of servers" below)
- the user's saved resources basket (see "selection of resources" below)

Note that only registered users can have a private workspace, and can save it on the server. Guest users will use a "default" workspace, and won't be able to save it on the server.

8 The User Interface

The ELAN client user interface is divided into three main zones :

- the top zone contains several tabs to browse through the various panels.
- the main center zone contains the currently selected panel.
- the botton zone is a status line to display extra informations, errors or messages sent by the servers.

The various "panels" are independant and allows selection of resources servers, querying of servers, basket management, user interface preferences selection etc.

The interface is available in several languages. Curently English, French, German and Dutch are supported.

8.1 The welcome Panel

You see this panel when you first launch the client application or applet. It just informs you that you are connected to the ELAN network, and allows you to quit with the "Close" button.

8.2 Selection of servers

This panel displays a list of all servers referenced on the NMU (Network Management Unit), that is, all servers you can contact through the ELAN network.

For each server, the following information is provided :

- the server's name
- the server's address on the internet (as a standard URL)
- the server's availability
- a short description of the server and its content

The last column of the servers table display the servers availability:

: the server is online and ready to answer queries.

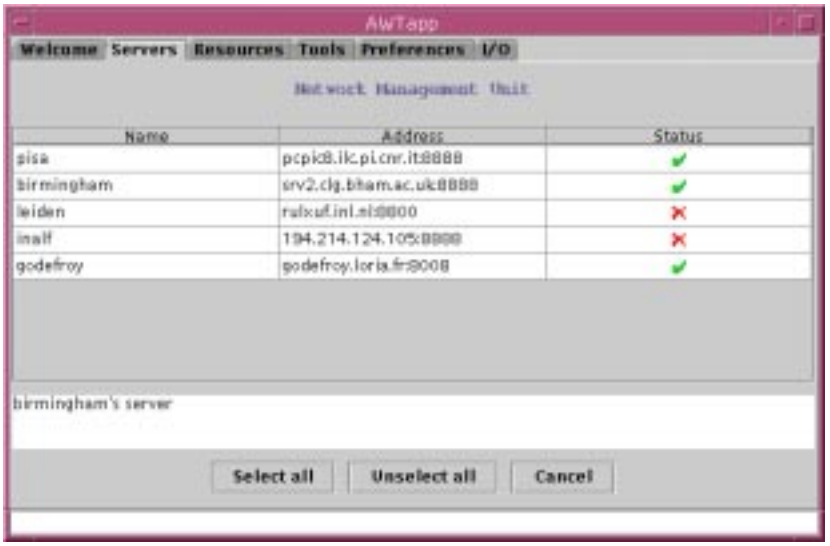
: the server is down and no queries will be sent to it.

The information box below the servers table will display the profile of a specific server, depending on the position of the mouse cursor. Move it over the servers table to get information about the servers.

You can select a server simply by clicking the corresponding line in the servers table. The selected server will be added to the servers section of your workspace.

You can also select or unselect all servers respectively with the "select all" and "unselect all" buttons. Multiple selections are allowed through "shift-click" and "control-click".

All following queries will be sent to the selected servers.



8.3 Selection of resources

Before to be able to send content queries, the ELAN scenario involves a phase of documents or sub-corpora selection. All documents will be saved as references in the current basket, which will later be sent with every content query.

This is done through 3 differents panels :

- A panel to send header queries and select individuals documents depending on their author, title or language.
- A panel to get a list of sub-corpora from the selected servers.
- A panel to display documents or sur-corpora references and add them in the current basket.

8.3.1 Header queries (selection of documents)

Here you can ask the selected servers for single documents references matching your query.
You have two ways for building your query:

- by directly typing your CQL query in the query text field.

- by using the graphic interface to build a query based on "titles", "authors" and "languages" attributes.

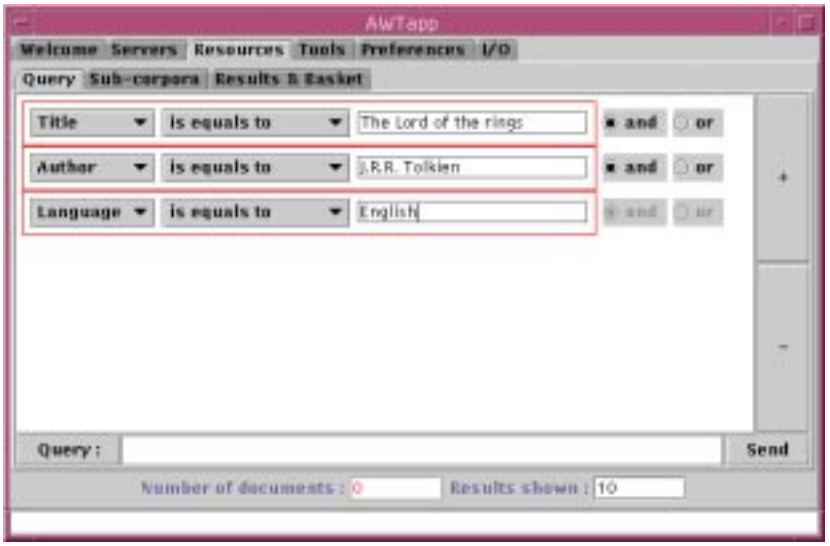
The query helping interface is made of a list of criteria and two button "+" and "-" on the right part to add or remove criteria. Each criteria contains 3 parts:

- a pop-up menu to choose the criteria attribute (title, author or language).
- a pop-up menu to choose an operator.
- a text field to enter the value the criteria should match, considering the selected operator.

After building your query with the graphic interface, you can preview it by clicking the "Query" button left to the query text field.

Note : The formalism used to build a query from the graphic interface might not be supported by some servers. In that case no results will be returned. Since this system is still under development, consider that typing directly your query or using only sur-corpora references is safer.

Once the query is completed, the client will automatically switch to the "resources selection" panel to display the results.

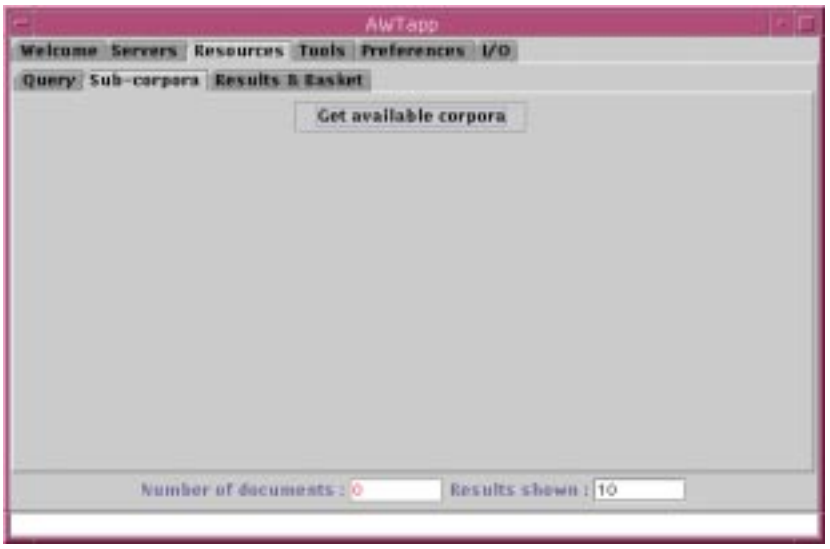


8.3.2 Get all corpora query

This panel is made of a single "Get all corpora" button that will send a request to all selected servers so they return a list of references for all sub-corpora they can handle.

Eventually, a single reference representing the complete server's corpus will be returned. Adding such a reference to the current basket means that content queries will apply on the whole server's corpus.

Once the query is completed, the client will automatically switch to the "resources selection" panel to display the results.



8.3.3 Documents selection

The document selection panel is basically divided in 3 parts :

- A botton line displays information about the current results set, that is the number of hits corresponding to the query and the number of documents or sub-corpora references actually displayed.
- The left part displays the documents references corresponding to the header query.
- The right part displays the content of the current basket, which will be sent with any further content query.

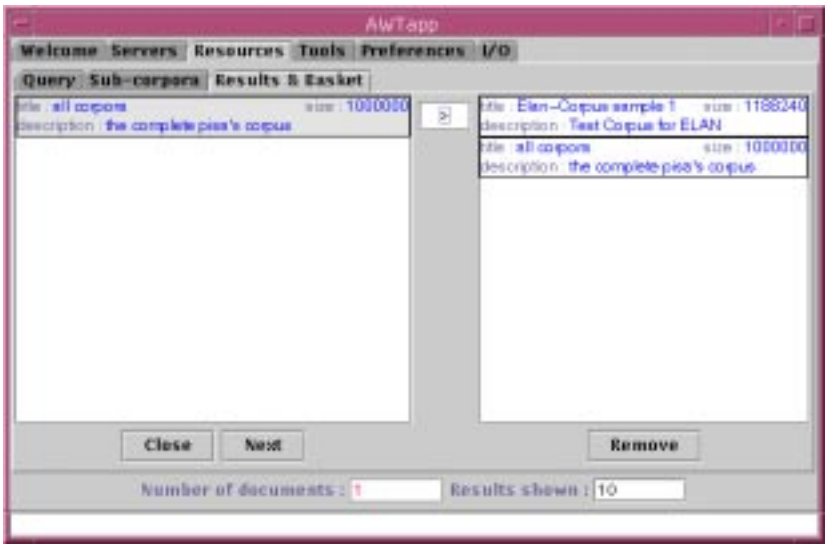
You can select documents by clicking on them, and you can use as usual "shift-click" and "control-click" for multiple selection.

Add documents in your basket simply by selecting them in the left list and clicking the ">" button between the documents list and the basket content list. The selected documents should appear in your basket content list on the right.

The "Next" button is used to display the next "results shown" results, if too many results have been returned to be displayed at once.

The "Close" button closes the current session and deletes the current results set.

The "Remove" button under the basket panel removes the selected documents from the current basket.



8.4 The tools panel

This panel presents differents tools to work with the resources saved in the current basket.

Queries will be sent to servers depending on the content of the current basket.

Though many tools might be available in the future, only a concordance line tool is provided currently.

8.4.1 Concordances queries

This tool allows you to send concordances queries in the CQL (Common Query Language) format.

Type your CQL query in the top text field and send it using the "Send" button. The status line will display a message to inform you that the query is being sent and processed.

The concordances lines will be displayed in a table, and centered on the node word, with left and rights contexts at both sides.

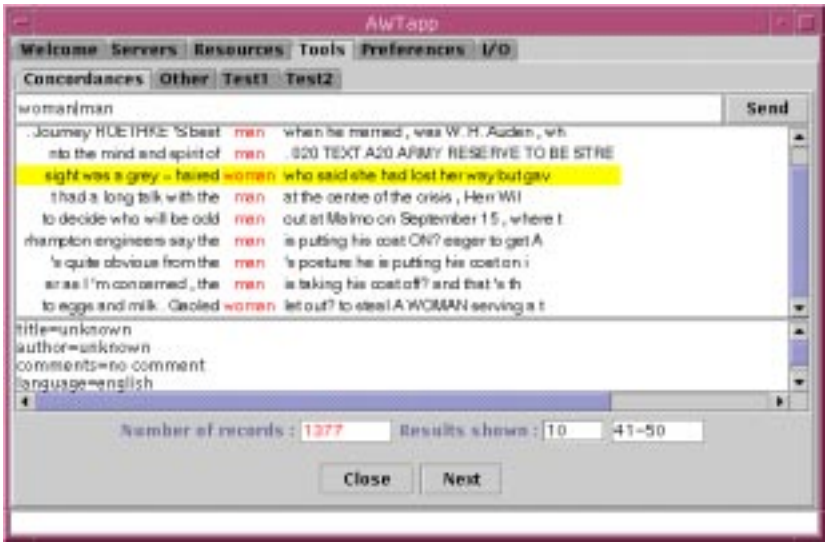
The bottom text zone displays extra information about a specific concordance line (document's title, author, language etc.). Just click on a concordance line to see this information.

Finally, the bottom line shows information about the result set being displayed :

- "Number of records" is the total number of hits of your query with the current basket.
- "Results shown" is the actual number of results displayed. You can retrieve the next results by clicking the "Next" button.
- The last zone displays the range of results being displayed from the current results set.

The "Next" button will retrieve the next "results shown" results from the current results set. You can specify in the "results shown" field the number of results you want to retrieve.

The "Close" button ends the session, and delete the current results set.

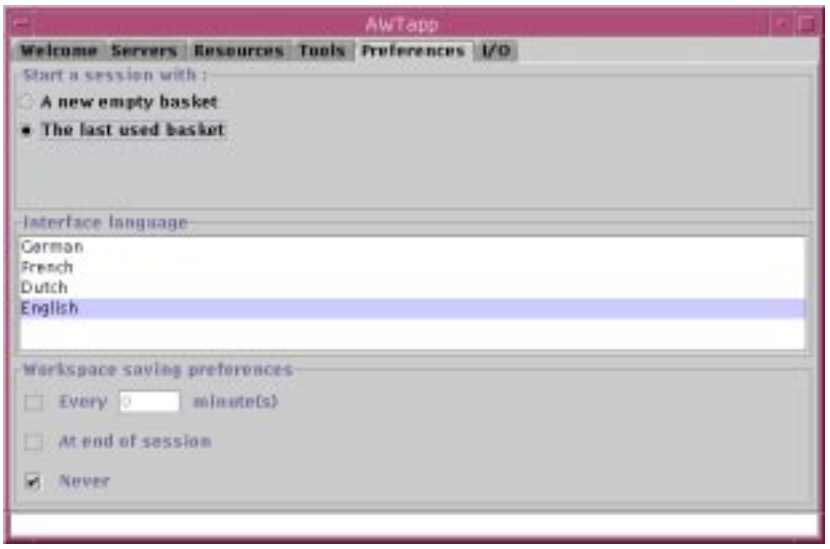


8.5 The preferences panel

This panel allows you to configure your environment. Three parameters can be configured :

- you can choose if the client starts with a new basket at each connection or if it starts with a new empty basket.
- you can choose the language for the interface (English, French, German or Dutch)
- you can choose how and when your workspace will be saved on the server

Note that the workspace (including your resources basket, servers selection and environment preferences) will be saved on the server for the next session only if you are a registered user.
Nothing will be saved on the server if you are connected as a guest.



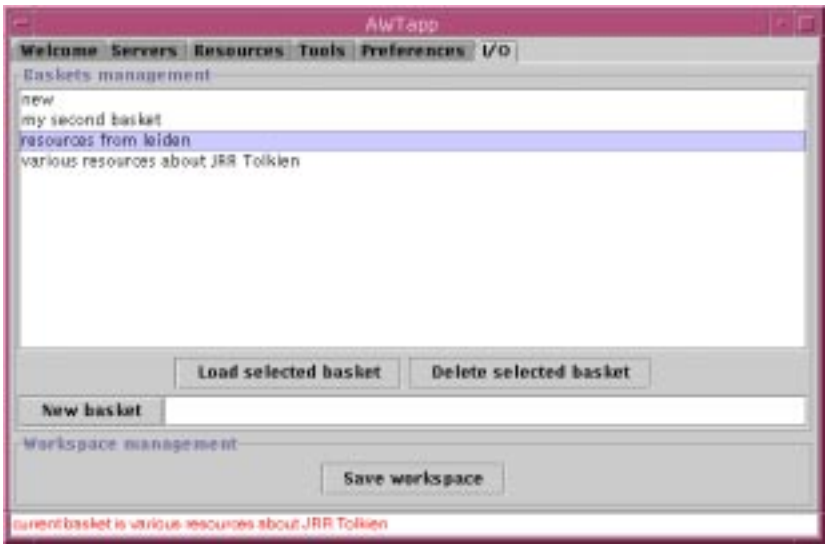
8.6 The i/o panel

This panel allows you to manage your resources baskets . Using this tool, you can :

- create a new empty resources basket
- delete the resources baskets
- reload baskets (make active saved basket)

All resources baskets are saved in the Workspace document.

Using the "Save Workspace" button, you can also save your workspace (including your resources basket, servers selection and environment preferences) on the server, if you are connected as a registered user.



The ELAN Server installation manual

Deliverable D3.2-2

Project Number	**MLIS-121**
Project Title	**European Language Activity Network (ELAN)**
Deliverable Number	D3.2-2
Work Package ID	WP3
Contractual date of delivery to EU	31 décember 1999
Actual date of delivery to EU	17 décembre 1999
Deliverable Title	**The ELAN Server installation manual**
Authors	*Christophe de Saint-Rat*

Abstract

The ELAN project is a distributed language resources system, offering access to existing resources to their potential users throughout Europe. In order to serve the electronic multilingual resource market, our task is to specify and elaborate a network of inter-connected resource servers. This document defines the technical specifications of each nodes that form the ELAN network: the user interface of the ELAN client; the resource servers and the network management unit.

Keywords

Language resources, Distributed systems, Network, User interface

Download ELAN server

http://www.loria.fr/projets/MLIS/ELAN/private/deliverables/elan/xsilfide.tar.zip

9 Required components

Some components are required in order to install an ELAN server:

- JDK v.1.1.6 minimum (v.1.1.8 is best)
  http://java.sun.com/products/jdk/1.1/index.html
- Swing components (JFC v.1.1 minimum)
  http://java.sun.com/products/jfc/index.html
- Servlet API (JSDK 2.1 minimum)
  http://java.sun.com/products/servlet/index.html
- Corba Broker (JacORB 1.0 minimum)
  http://www.inf.fu-berlin.de/~brose/jacorb/

10 The XSilfide archive

The ELAN server package is available as a "tar.zip" file that you can decompress with the following commands:

```
unzip xsilfide.tar.zip
tar -xvf xsilfide.tar
```

This will create an xsilfide directory that contains everything to start a new server. The JacORB corba broker and the Java servlets API are included in the release, but you have to install a JDK and the swing components separately.

The xsilfide directory is organized as follow :

lib/	
sxp.jar	XML parser
xlink.jar	xLink support
xpointer.jar	xPointer support
client.jar	the client applet and application
server.jar	the server classes
query.jar	communications between servers and clients
util.jar	various utility classes to be used in the architecture
silfideall.jar	complete archive with all the classes
nmu/	packages to access the NMU
lib/	
jacorball.jar	the JacORB CORBA broker
nmu.jar	package to connect to the NMU with CORBA
silver/	
classes/	the classes for the Silver HTTP server
lib/	
silver.jar	the Silver HTTP server jar file
servlet-2.1.0.jar	the Java Servlet API version 2.1
silfide/	
bin/	scripts to start the server (for UNIX and WINDOWS)
cache/	contains the cache files when the server is running
classes/	the Java classes (equivalent to lib/silfideall.jar file)
dtd/	the DTDs for the various XML documents
htdocs	the home of the server
index.html	the ELAN server homepage
jars/	jars files for the client applet
icons/	icons for silver
images/	images (ELAN logo etc.)
lib/	empty...
logs/	the silver server logs files
preferences/	the server configuration files
properties/	the silver properties files
realms/	not used

servlets/	the various ELAN servlets
users/	the users profile
workspaces/	the users workspaces
security/	
otp/	contains the classes for the «one time password» autenthification system
serverAdmin/	contains the server administration application

11 Installation

11.1 Environment variables

The CLASSPATH environment variable must be set to provide access to the required java packages. It should contain paths to:

- external components :

  $HOME/jdk1.1.8/lib/classes.zip

  $HOME/swing-1.1/swingall.jar

  $HOME/jsdk2.1/servlet.jar

  $HOME/JacORB1_0/classes/jacorb.jar

- ELAN components

  XML parser

  $ELAN_INSTALL_DIR/xsilfide/lib/sxp.jar

  Silver HTTP server

  $ELAN_INSTALL_DIR/xsilfide/silver/lib/silver.jar

  ELAN package

  $ELAN_INSTALL_DIR/xsilfide/lib/silfideall.jar

11.2 Silver HTTP server configuration

The xsilfide/silfide directory contains all the components specifics to the ELAN server.

11.2.1 httpd.properties configuration

Edit the httpd.properties file in the properties directory and set the following attributes :

```
# Where is installed the server ?
silver.root=$ELAN_INSTALL_DIR/xsilfide/silfide

# Hostname and port number
silver.host=localhost
silver.port=8888

# Server administrators email address
silver.admin.email=desaintm@loria.fr
```

11.2.2 other properties files

The other properties files contain various attributes but should not be modified.

- servlet.properties contains the servlets configuration and initialization arguments.
- mime.properties contains the MIME types recognized by the server.
- aliases.properties contains aliases defined by the server.
- silfide.server.properties contains the paths to some important directories in the xsilfide archive.

11.3 ELAN server configuration

The ELAN specific directories are located in xsilfide/silfide :

htdocs

WorkspaceApplet.html : HTML file to start the applet.

jars

silfideall.jar: ELAN archive containing the applet code and all needed files.

sxp.jar: XML parser.

silver.jar: the client use the MIME classes in this package

servlet.jar: for the applet/servlet communication

servlets

ConnectServlet.class : connect a user to the server.

BroadcastServlet.class : broadcast queries to the registered elan servers.

NMUSilfideClientUniversalServlet.class : connection to the NMU.

SILManagerServlet.class : management of SIL files.

SilfideStatusServlet.class : called by the NMU to check server status.

QueryServlet.class : answer the queries by calling the appropriate query driver.

properties

Contains the HTTP server properties files (httpd, servlet, aliases, mime, etc.)

preferences

Contains the ELAN server preferences. (the ServerPrefs.xml file)

workspaces

Contains the users saved workspaces and the "guest" workspace.

cache

Contains the results sets cache files when the server is running,.

11.3.1 ServerPrefs.xml configuration file

Edit the ServerPrefs.xml file in the silfide/preference directory and set the following attributes appropriately :

<pref name="status" value="online"/>

server status : should not be modified.

<pref name="name" value="no-name"/>

server name or sid : this is the server's identification reference on the network. It must be unique in the network and registered on the NMU. Please contact the NMU administrator before to set this attribute.

<pref name="nmua" value="godefroy.loria.fr"/>

address of the NMU server. Should not be modified but on NMU administrator request.

<pref name="nmup" value="1130"/>

port of the NMU server. Should not be modified but on NMU administrator request.

<pref name="driver" value="fr.loria.silfide.driver.QueryHandlerEmptyImpl"/>

ELAN driver name : modify this to provide access to your site driver. Note that your CLASS-PATH variable must contain a path to the driver.

<pref name="namingContext" value="ELAN"/>

the naming context to be used to connect on the NMU. (should not be changed)

<pref name='nmuConnectMode' value='corba'/>

protocol to be used to connect to the NMU. Can be either "corba" to connect directly or "udp" to use the udp gateway.

<pref name='broadcastTimeout' value='3'/>

this attribute specifies the time-out (in minutes) the broadcastServlet will use when contacting a resource server. After this delay, the connection will be cancelled.

<pref name='serverPort' value='8888'/>

specify here the port your server will use. This value must match the silver.port property in the httpd.properties file.

<pref name='guestAccess' value='yes'/>

should be "yes" if you want to allow guest connections to your server.

12 A user-friendly configuration tool...

In the silfide/bin directory, launch ./admin to start the configuration tool. Its purpose is to edit the different properties and configuration files. Note that you shouldn't modify any properties but those described in this document.

You can edit a property by double clicking on its key or value. You should then be able to edit it and validate your changes by hitting the return key. Note that for some properties, you cannot modify the key. Sometimes you wont be able to add or remove any properties from the file, but only edit the actual properties values.

The interface is basically made of tables displaying the properties from the various files, and accessible through tabs.

Some buttons allow the administrator to add and remove properties, and validate the modifications :

- Add : add a new property to the file. You must enter the key and value of the new property in the new line that will appear in the table.
- Remove : remove the selected property from the file.
- Apply : validate the modifications and save them in the property file. Note that for the modifications to take effect, you must restart the server.

Below are some screen shots of the administration tool :

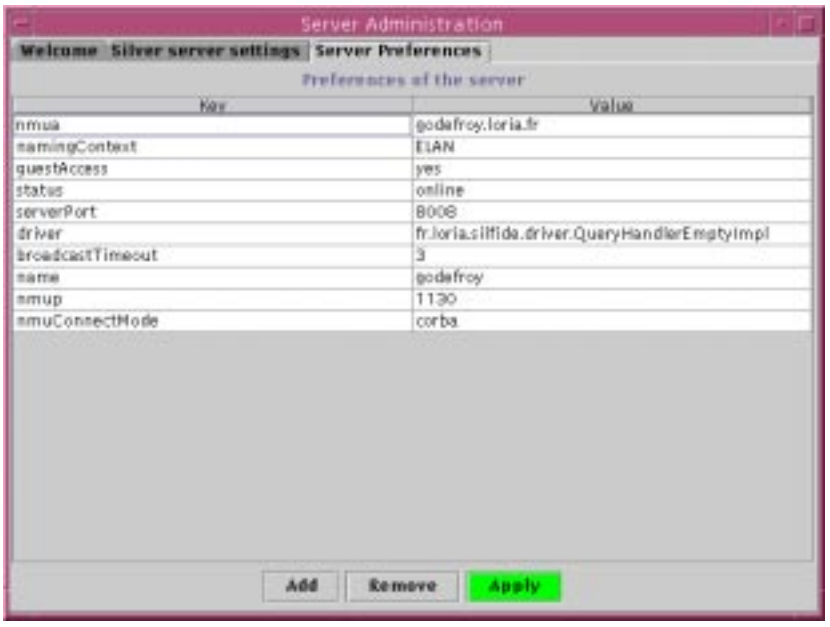


Figure 12. **Edition of the ServerPrefs.xml file**

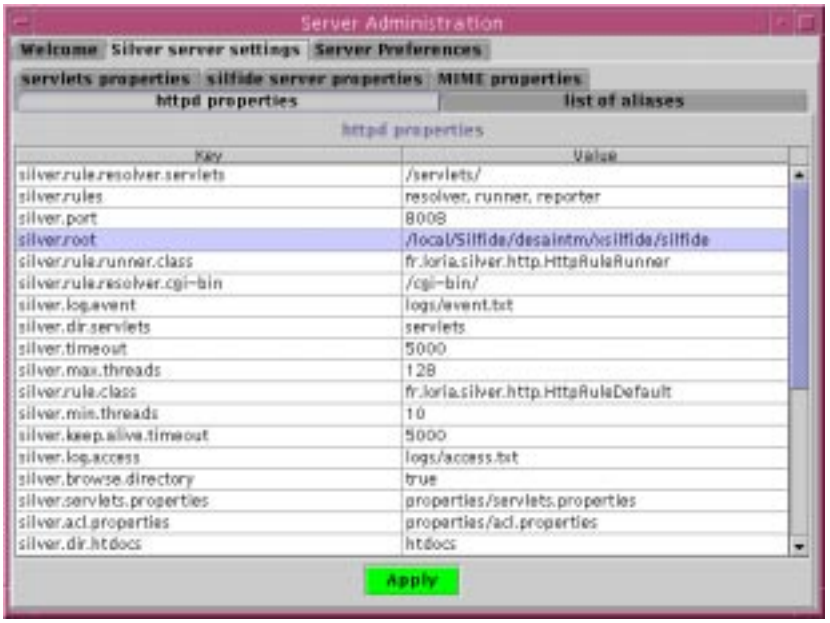


Figure 13. Edition of the httpd.properties file. The "silver.root" property is selected. Only the "silver.root", "silver.port" and "silver.admin.email" properties should be modified

13 Starting the server...

Go to the silfide/bin directory and launch the "server" or "server.bat" script, respectively on Unix and Windows systems. you should see something like:

```
XSILFIDE_ROOT=./../..
Initializing HttpServer...
Silver (version 0.32) , serving at http://godefroy.loria.fr:8888/
HttpServer is running...
```

14 Testing...

You should be able to see the server's homepage at http://server.where.com:port where "server.where.con" is the address of your server and "port" the port you decided to use.

You can use any Swing enabled browser to connect to the homepage and launch the applet either as a guest or a registered user.

If you dont have a swing enabled browser, you can launch the ELAN client either as an applet using the AppletViewer tool or an application.

To connect with the applet as a guest :

```
appletviewer http://server.where.com:8888/servlets/ConnectServlet
```

To connect with the applet as a registered user :

```
appletviewer "http://server.where.com:8888/servlets/ConnectSer-
vlet?  LOGIN=login_name&PASSWD=asasasasasasasa"
```

To connect using the application :

```
java fr.loria.silfide.client.WorkspaceApplication
```

you will have to provide the server address, a login name and a password. Leave the login and password fields blanks to connect as a guest.

```
-or-
```

```
java fr.loria.silfide.client.WorkspaceApplication server_url
login_name password
```

The ELAN common query language (CQL)

Deliverable D3.3-1

Project Number	**MLIS-121**
Project Title	**European Language Activity Network (ELAN)**
Deliverable Number	D3.3-1
Work Package ID	WP3
Contractual date of delivery to EU	t0+9
Actual date of delivery to EU	30 November 1998
Deliverable Title	**The ELAN common query language (CQL)**
Authors	*Peter van der Kamp & Pieter Masereeuw, INL, Leiden*

15 Foreword

The CQL described in this document can be regarded as a maximum model. Although all the software partners involved with implementing the CQL will do the utmost to be completely compatible with the described CQL, small differences between the partners can remain. These differences are due to the use of pre-existing software brought into the ELAN project. Differences are described in the user documentation.

16 Introduction

This document describes the ELAN Common Query Language (CQL).

16.1 What is the ELAN Common Query Language and why is it needed?

The ELAN project aims to make parts of the PAROLE and TELRI corpora available for retrieval via the Internet. For various reasons, it was decided not to create a new retrieval system. Instead, existing systems had to be reused in the context of the ELAN project. A number of institutions has been found willing to cooperate in the process of making their corpus retrieval software available for this purpose.

The existing systems have more or less comparable functionalities; however, they differed in the way in which the user had to interact with them - i.e., formulate queries, perform statistical tests, view output. ELAN aims to overcome these differences by realizing a common query language and a common user interface by using standard components. The common user interface will be based on Java and Internet browser technology, esp. the Netscape Navigator and Microsoft Internet Explorer platforms.

For this user interface to be able to "speak" to the various retrieval engines, a common language has to be defined. The language that will translate the ELAN query language into the command syntax for the various query systems is called the **ELAN Common Query Language**.

The institutions that make their retrieval engines available for use within the context of ELAN are:

1. LORIA UMR 7503 CNRS, INRIA & Universités de Nancy
2. University of Birmingham, Department of English, School of Humanities, Birmingham, United Kingdom
3. CNR - Istituto di Linguistica Computazionale, Pisa, Italy
4. Instituut voor Nederlandse Lexicologie (Institute for Dutch Lexicology), Leiden, The Netherlands

16.2 Design objectives

The design objectives for the ELAN Common Query Language are:

1. simplicity - the language should be easy to use
2. power - the language should allow non-trivial queries, like adjacency queries, part-of-speech queries, boolean combinations and so on
3. if there is a conflict between the power concept and the simplicity concept, simplicity wins
4. implementability - it should be possible to translate CQL expressions into the local query format of all ELAN retrieval systems
5. even being easy to use, the language should permit the creation of a wizard-like interface that helps the user formulate a query - in some circumstances even a simple language requires some special notation, for instance when looking for:

   *part-of-speech tags (**PoS** tags)*

   words separated by a given number of other words

discrimination between uppercase and lowercase characters

SGML markup

etc.

6. the language should support hierarchically structured documents, marked up with SGML
7. this support of SGML should not be dominant - users unaware of markup should not be forced to think about it, let alone be confronted with SGML.

Despite the identical user interface and common query language, different concepts in the various retrieval program may cause subtle differences in query results. For instance, the Leiden approach is based on retrieving sets of phrases (Leiden jargon for paragraphs or sentences), whereas, e.g., the Birmingham approach is based on retrieving positions in the corpus where the match is found. One of the consequences of the Leiden approach is that searching for words will never cross phrase boundaries: where the Birmingham system silently crosses such boundaries during the match, the Leiden system will not. This can, in some circumstances, result in different output.

17 Components of the ELAN Common Query Language

In this section we will discuss:

The context of a query

Features supported in the ELAN Common Query Language

Query syntax

Before we describe the features and the syntax of the language, we will provide list of example queries. We hope that this will make the rest of this document easier to read.

Find all occurrences of *real example*

```
real example
```

Find some forms of the **verb** *ski*

```
ski | skies | skiing | skied & V
```

Find some forms of the **verb** sky near a **noun** from the list *ball, &painting* or *picture* (where *near* means: *within a distance of 5 words to the left or the right*):

```
sky | skies | skied | skying  & V 5:5 picture | painting | ball & N
```

Find questions (trick: search a question mark at the end of a sentence):

```
? </S>
```

Find all words that start with house:

```
house*
```

Advanced: regular expression search - *color* or *colour*

```
regex(colou?r)
```

Advanced: combining case-sensitive and case-insensitive matching (note: normally, case sensitivity applies for the whole query, and is a feature selectable in the user interface) - the *Nobel Prize for linguistics* or *Linguistics* (or *LINGUISTICS*, etc.)

```
caseless(linguistics) 3:3 Nobel Prize
```

17.1 The context of a query

A query is always made within a certain context. For an ELAN query, this context consists of:

corpus selection.

The corpus to be queried, or a subset of a corpus

type of the query.

contextual (e.g., concordance lines)

statistical (miscore, t-score)

parameters relevant for the given query type, e.g.**:**

case-sensitivity of the query (default is case sensitive matching, a lowercase

a does not match an uppercase A)

output parameters for the concordance lines

type of statistical test to be performed

optional header query .

The header query operates on the TEI headers of each document in the selected corpus. If there is a match, the document corresponding to the TEI header is included in the corpus to be queried by the text query (see below); if there is no match, the corresponding document will be excluded.

If no header query is present, the entire (sub)corpus selection will be used.

More on header queries in section 3.1 below.

text query.

The text query performs the actual query. It operates on the documents defined by the corpus selection, possibly modified by the header query. The processing of the results depends on the type of the query: either output is sent back to the user, or statistical tables are generated on which the statistical test can further operate.

When the user wants to perform a query, the entire context will be passed from the client in the web browser to the servers defined for the given corpus selection. The context will be represented in the form of an XML Document Type Definition (DTD), a draft version of which is given in Appendix A.

17.2 Features supported in the ELAN Common Query Language

In this paragraph we will discuss the features that are supported in the ELAN Common Query Language. In order to keep things simple, we do not include the actual syntax here; the actual notation will be discussed in the next paragraph.

The ELAN Common Query Language will provide the following features:

single-word expression.

Single-word expressions correspond with one word, word-plus-PoS-code or SGML tag in the corpus. A single-word expression can be:

word expression.

A word expression is used to search the corpus for a **word** or a **combination of a part-of-speech tag and a word**:

Words are defined as:

*** literal word.** A literal string that represents a **word** or **punctuation**.

*** wildcard.** Wildcard searching makes it possible to specify that certain parts of a word (tag, etc) can consist of arbitrary characters, for instance, `hous*` would select all words that start with `hous`. Wildcard characters can be used any number of times anywhere in a word.

*** list of alternative words.** E.g., searching for the word `house` or `houses`

*** function.** Each ELAN server will define a set of functions. The effect of the functions is the same as searching for a list of alternative words (the preceding option). In fact, one might say that a function is substituted by a list of alternatives, as defined by the function and its parameters. Functions may help the user by automatically generating lists of alternative words. An obvious example is the generation of all inflected forms of a given stem.

Each server will at least support the following two functions:

`caseless()`

The result is the list of all alternatives of its arguments. For example, if the preceding sentence were the corpus, `caseless(the)` would be a list consisting of *The* and *the*. The `caseless()` function may be abbreviated to `case()`.

The argument of `caseless()` is interpreted as if it were a literal word or a wildcard. Hence, `caseless(*ily)` will match all words ending in ily, uppercase as well as lowercase.

`regex()`

The result will be a list with all words in the current corpus selection that match the regular expression string passed as argument. The regular expression string will need to match from the start of a word upto its end. For instance, the result of `regex([Tt]h.)` could be a list consisting of *The, the, Tho, tho, Thy* and *thy*. The result will not contain words like *ethic* or *than*.

If the user has indicated that the overall setting for the query is to match case-insensitively, the `regex()` function will also be case-insensitive.

The regular expression variant to be used will be Unix-style regular expressions.

Combinations of a part-of-speech tags and a word are used to search for a combination of a word and a PoS tag. It is not possible to search for the combination of a word and an SGML tag (see below), or for the combination of an SGML tag and a PoS tag. It is possible to search for a PoS tag only - i.e., a PoS tag corresponding to an arbitrary word.

PoS searches consist of a **word**, a **PoS operator** and a **PoS tag**.

Instead of a ***word*** one can also use one of the alternatives listed above (wildcards, lists of alternative words, functions).

The **PoS operators** are:

*** and.** The *and* operator indicates that the second operand should also match on the same position in the corpus as the first operand. A typical application of the and operator would be a word with a certain PoS tag.

The *and* operator specifies that the word or words to the left of the operator should be below a <W> SGML tag whose attribute matches one of the PoS tags that are to the right of the operator.

*** and not.** The *and not* operator inverts the effect of the *and* operator: if the *and* operator would match, the *and not* operator will fail, and vice versa.

A **PoS *tag*** is a morphosyntactic tag as defined for the Parole corpus. Like a word, a PoS tag can be:

- a sequence of characters that together constitute the start of a PoS tag.

- a wildcard that will match the start of a PoS tag (wildcards for PoStags have the same notation as those for words)

- a list of alternative tags (also using the same notation as for words)

Matching behaviour for PoS tags will be different than that of words: where a word (or word wildcard) must match entire words in the corpus, PoS tags (or PoS wildcards) only need to match the start of PoS tags in the corpus. This definition was done for convenience, because the Parole tags are to a large extent hierarchically structured with the leftmost character being most significant. Thus, the user can ask for a PoS V (for *verb*), without having to worry about any other characters that may follow.

SGML tag.

If the corpus supports SGML tags, they can be queried. SGML tags can be expressed as:

*** a literal string.** A tag that complies with SGML conventions

*** a wildcard.** In SGML tags, wildcards are only allowed in the attribute values, not in tag names or attribute names.

If any attributes are given within an SGML tag, the order in which they are given is irrelevant for the match. An SGML tag in the corpus that has more attributes than specified in the query expression will match nevertheless.

All matching is case sensitive, unless a global property of the query is changed. This property can be changed within the user interface and will be passed to the server as one of the parameters of the query context as described in section 2.1.

multiple-word expression.

Multiple-word expressions are combinations of the single-word expressions defined above.

juxtaposition. Two single-word expressions next to each other in the expression specify that a match should be found in the corpus where these words (tags, etc.) are adjacent. Words are adjacent if the first word is immediately to the left of the second word, where any intervening tags are ignored (but intervening punctuation is *not* ignored).

span . The *span* operator indicates that two multiple-word expressions should occur within a given distance from each other. The span operator can specify the size of the distance and whether the expressions should occur in a left-to-right order or in any order.

If SGML tags are used among the multiple-word expressions, there will be no attempt to make sure that an open SGML tag and a closing tag belong to each other. If matching tags are required in the expression, refer to the section about hierarchy expressions (below).

Hierarchy expression.

Given the fact that many corpus texts to be supported in ELAN are heavily tagged with the TEI tagging system, the CQL provides a means to combine a search for words with a search for the hierarchical structure of these documents. For instance, one may search for a word that is below a TEI <HEAD> tag. Users may formulate their searches in terms of words and tags, but apart from that, they may also want to express that some hierarchical relation between the tags should exist.

Hierarchy expressions find an important application in the area of header queries. For example, the user may want to select documents published in1995 by searching for a <DATE> tag that is below a <PUBLICATIONSTMT> tag, where the string `1995` is found within the domain governed by the <DATE> tag.

Due to conceptual differences, the Leiden server will have restrictions for text queries that search for word combinations that cross the boundaries of a phrase (e.g., a paragraph) or the header.

17.3 Query syntax

The full syntax of query expressions is described in appendix B.

In this paragraph, we will provide the actual notation for the features that were discussed in section 2.2.

Between words and operators, all whitespace (blanks, spaces, tabs) is ignored.

single-word expression.

Single-word expressions are word expressions (including part-of-speech tags) and SGML tags. PoS tags can only occur in combination with words (or with the * wildcard, which stands for *any word*)

word expression. A word expression is a literal word, a wildcard, a list of alternative words or a function that represents such a list of words.

*** word.**

literal word

for CQL purposes, a word is defined as a sequence of non-blank characters, with the exception of the following *special characters*:

& means *and* (see below)

~& means *and not* (see below).

| means *or* (see below)

: part of *span* operator

< and **>** delimit SGML tags

(and **)** mark functions; also reserved for future use (hierarchy expressions)

" and **'** used for quoted words and quoted function parameters (see below)

^ Reserved for future use (in hierarchy expressions)

Searching character entities, like é for é, is not supported. Instead, all characters should be entered in their Unicode (ISO 10646) representation (wysiwyg (what you see is what you get): users need not bother with character entities, they only work with real printable character). Unicode characters will match the corresponding character entities in the corpus data. The user interface will somehow have to provide a facility for easily entering special characters.

Examples of literal words:

`cat`

`dog`

`élan`

Note that the original CQL definition had no facility to enter a word between single or double quotes. Quotes were mainly introduced in order to make it easier to enter special regular expression characters in the `regex()` function. Other uses of quoted strings are their for reasons of orthogonality.

wildcard

The following wildcard characters are also special, but may be part of a word, where they have a special interpretation:

_ (underscore) matches an arbitrary character

***** matches zero or more arbitrary characters. An * by itself means: any word.

**** makes the following character not-special, eg. \&,_ or \\.

Examples of wildcards:

*****	any word
un*	words starting with un
***able**	words ending in able
un*ily	words starting with un and ending in ily
AT\&T	The *AT&T* company

quoted words

If a literal word or wildcard contains many special characters, it is more convenient to enter them within quotes. Within quotes, all special characters mentioned above (under literal word) become not-special. However, the wildcard characters (_, * and \, see above) keep their wildcard interpretation.

The user may use single quotes as well as double quotes - the only requirement is that the start and the end of the string have the same quote. Within double quoted strings, single quotes can be freely entered; likewise, a single quoted string may contain double quotes. For a detailed discussion, see Appendix C.

Examples of quoted words and wildcards:

*** "I'm".** same as I\'m

*** '2.5"'.** 2.5 inch; same as 2.5\"

*** "AT&T".** same as AT\&T

*** I'*.** Words starting with I', same as I\'*

*** '***-hotel'.** same as ***-hotel - probably not a hotel where you want to go with four little children

*** '\'', "\"".** even the absurd may be entered

list of alternative words

A list of alternative words consists of literal words, quoted words and/or wildcards, separated by a vertical bar character |.

Examples of alternatives:

```
house | houses
*ity | *ness
```

function

A function consists of a name followed by a number of parameters. The parameters are given between round parenthesis and are separated by commas. The name of the function is a sequence of unaccented letters in the ascii range defined by a..z and A..Z. The parentheses should follow the name without intervening whitespace. Function names and parameters are case sensitive. Function parameters are made of any characters, except whitespace, commas and round parentheses. If necessary, these should be escaped with a back slash \. Alternatively, and perhaps more conveniently, function parameters may be enclosed within quotes. In that case, the same rules apply as for quoted words, with the difference that functions define their own interpretation of parameters (e.g., `caseless("wh*")` matches all words starting with wh, Wh, WH, wH, while `regex("wh*")` will match words like w, wh, whh, and so on). See Appendix C for a description of the interpretation of function arguments.

Examples of functions:

```
flex(house)
```

This would (perhaps, depending on the corpus) be equivalent to the list house | houses. Note that a server need not necessarily have a flex() function; depending on the language, a server may offer different variants of flex() in order to deal with language-specific phenomena.

```
regex("[aeiouy]+")
```

This would be equivalent to the list of words that consist only of vowels. Note that regex([aeiouy]+) is allowed as well, since none of the characters is special.

```
caseless(house)
```

This would (perhaps, depending on the corpus) be equivalent to the list of all lowercase and uppercase variants of the word house, e.g. `house | House | HOUSE`.

`regex("koning(in)?")`

Matches the Dutch words koning (king) and koningin (queen). Note that you can't write this as regex(koning\(in\)?) (as this would match the 'words' koning(in and koning(in) - note: parentheses are correct!

*** combination of a word and a part-of-speech tag.**

part-of-speech operator

The PoS operators are:

`&` for the *and* operator
`~&` for the *and not* operator

Examples of the use of PoS operators:

`walk & V` : The word *walk* with a PoS tag that starts with V (verb)
`* & N` : Any noun
`flex(house) & N` : All inflected forms of the stem house that have a PoS tag starting with N (note: the flex() function is example-only)

part-of-speech tag

A PoS tag obeys to the same rules as a word. For convenience, a PoS tag is implicitly extended with a * wildcard character. This was done for convenience, so that the user can enter

`walk & N`

if the PoS tag for walk really is something like `Ncfs--`.

Instead of a PoS tag, one may also use:

- a wildcard. Wildcard characters in PoS tags obey the same rules as those for words.

- a quoted word (See above)

- a list of alternative PoS tags. A list of alternaive PoS tags has the same format as a list of alternative words.

Examples of somewhat elaborate use of PoS tags:

`dictionary & N_[-m*][sp]`
The word dictionary with a tag N, followed by any character, etc,
`* & A | R`
Any word with a PoS tag starting with A or R.

SGML tag.

An SGML tag is enclosed within angle brackets < and >. The normal SGML notation applies, so, for instance, a slash / following the < indicates a closing tag.

Within attribute values, the wildcard characters _ * and \ have the same interpretation as within words. Wildcard characters are not allowed within element or attribute names.

multiple-word expression.

Multiple-word expressions are combinations of the ***single-word expressions*** defined above.

juxtaposition.

Two single-word expressions can be combined by putting them next to each other, with intervening whitespace.

Examples of juxtaposition:

`central heating`

* &A * &A * &N : two adjectives followed by a noun

span.

The span operator will match the single-word or juxtapositional expressions that surround it. The syntax of the query language forbids more than one span operator in an expression. This was decided in order to prevent a combinatorial explosion when translating the ELAN Common Query Language to the local format.

A span operator consists of two numbers separated by a colon :. The span operator operates on two sequences of one or more words - one sequence to the left (S_L) and one sequence to the right (S_R). The numbers in the span operator define the distance between S_L and S_R in words. The distance between two words is defined as follows: if all words in a text are numbered, the distance of two words is the number of the rightmost word minus the number of the leftmost word. This implies that two adjacent words have a distance of 1 and that a distance of 0 is not possible between two words.

the **first** number of the span operator defines the distance between S_L and S_R if S_R is positioned to the **left** of S_L

the **second** number defines the distance between S_L and S_R if S_R is positioned to the **right** of S_L

As an example, the query `draws attention` is identical to the query `draws 0:1 attention`.

When counting intervening words, SGML tags are disregarded; punctuation is counted as words.

Note: the exact definition of word and punctuation will be defined by the tokenization strategy of each Elan server - this implies that there may be some variation between servers.

Examples of a span:

`draws 1:3 attention` would match:

```
draws attention
draws my attention
draws his sister's attention
attention draws
```

but not:

`draws more than my attention` (the distance between draws and attention is 4, not 3)

`attention and draws` (distance is 2, not 1)

Hierarchy expression.

During the development of the CQL, all partners agreed about the features of the CQL. Also, notational issues were for the largest part agreed upon. However, a consensus about the syntaxis of hierarchy expressions has not been reached. Two notations have been proposed, one by Leiden and one by Birmingham. Rather than discussing the two approaches in detail, we will try to show the difference by means of a number of relevant examples. A final choice still has to be made, where, among others, aspects like notational comfort will have to be taken into account.

TABLE 1.

Description	Leiden notation	Birmingham notation
The name *Lubbers* within a <HEAD> tag	<HEAD>(Lubbers)	Lubbers within(HEAD)
The word *premier* followed by *Lubbers* within a <HEAD> tag	<HEAD>(premier Lubbers)	premier Lubbers within(HEAD)
Searching for an italic word between other words, e.g. "at *the* bank"	at <HI rend=IT>(the) bank	at (the within(HI,rend=IT)) bank
Looking for certain question-answer patterns: a sentence that contains a ?, followed by a second sentence with the word yes, followed by a third sentence that also contains a ?	<S>(?^) <S>(yes) <S>(?^) Note: ^ anchors to the end	?within(S) (</S>within(S)yes) ? within(S)
Header query: select all freely available documents with source date 1992	<PUBLICATIONSTMT> (<AVAILABILITY STATUS=FREE>) <SOURCEDESC> (<DATE>(1992))	(<AVAILABILITY STATUS=FREE> within(PUBLICATIONSTMT)) within(TEIHEADER) (1992 within(DATE) within(SOURCEDESC))

18 Processing CQL queries

18.1 Text queries and header queries - what they do and how they cooperate

Queries are applied to a set of corpora or subcorpora available on one or more ELAN servers. The user defines this set by selecting from a list of available corpora. This list is composed from the corpora and subcorpora available at all ELAN servers for which the user is authorized.

For some applications, users may want to further refine the selection made from the standard list. For instance, the granularity of the list may allow selection of newspapers published in a given year, but the user wants to limit his domain to all articles written on fridays (perhaps because at that day papers always have articles from a certain category).

In order to do so, the user can query the TEI header of the various documents in the selected corpus - this is done by so-called *header queries*. If a header matches the user's query, the document to which the header belongs is included in the current subcorpus selection. Of course, documents of headers that don't match are *excluded*.

It will be clear that the effect of header queries and text queries is different:

header queries select documents

text queries select a context - say, a concordance line - in the corpus

Nevertheless, the query language for both queries is identical. This is because both header and content are made up with SGML tags. Of course, the header query facility will not be available for documents that do not have a header. Likewise, text queries on documents without SGML markup will not be able to specify SGML tags, PoS tags (which are represented as SGML tags) and hierarchy expressions.

18.2 Glue

A full description of context in which text queries and header queries are transmitted to an ELAN server (which may broadcast edited copies of them to other ELAN servers) is outside the scope of this document. Nevertheless, in order to obtain an understanding on how an ELAN server will process a query, a few words need to be said about it.

The ELAN user interface will wrap the user's text query (and the optional header query) in an XML document - this is transparent to the user, it's just part of the protocol by which the client (user interface) and the server talk to each other. This document will also contain other information - for details, please refer to section 2.1.

Part of the entries in the XML document is the kind of operation that is to be performed - type of the query in section 2.1. The following types have currently been envisaged:

output a set of concordance lines (context size will also be specified in XML terms). The format of the output will be an XML document that the user interface understands sothat it can do formatting and highlighting in the user interface.

output a table with statistical information (type of statistical test and parameters will also be supplied). This table, formatted as an XML document, will be formatted by the user interface

output larger context given a certain id (the id corresponds to a position in the corpus)

output a list of words matching a given regular expression

output the frequency of a given word, also given a certain (sub)corpus selection.

Part of the protocol will originate from the server instead of from the user interface - for instance, if a corpus selection spans corpora that are distributed over several servers, one server will be responsible for merging the results coming from other servers. As another example, in order to do statistical computations, the responsible server may need to know the frequency of a word and ask that to the appropriate server.

The surrounding protocol has not yet been fully established. For example, the client may request that certain concordance lines are removed from the result set (filtering). A user may want to do that once it becomes apparent that a query contains several easily identifiable examples that are beside the point and would disturb statistics. Filtering the output may be more convenient that rephrasing the query.

Issues like this are part of the surrounding protocol and not part of the CQL proper.

Server hardware and software requirements

Deliverable D3.3-2

Project Number	**MLIS-121**
Project Title	**European Language Activity Network (ELAN)**
Deliverable Number	D3.3-2
Work Package ID	WP3
Contractual date of delivery to EU	t0+9
Actual date of delivery to EU	30 November 1998
Deliverable Title	**Server hardware and software requirements**
Authors	*Peter van der Kamp & Pieter Masereeuw, INL, Leiden*

1 Introduction

This document describes the hardware and software requirements for ever site in order to act as a server within ELAN.

2 Requirements

2.1 General requirements

Independent of the situations described in paragraph 2.2, Specific requirements, the following general requirements apply to act as a server within ELAN.

1. fast connection to Internet, not based on modem or comparable technology
2. running an appropriate http-daemon (e.g.Apache), configured to accept servlets
3. running a Java Virtual Machine, compliant with the specifications of Java Development Kit (JDK) 1.1, in order to run servlets
4. the format of the data to be accessed should be level zero (no markup) or TEI DTD compliant.

2.2 Specific requirements

With respect to the specific requirements, two situations can be distinguished.

1. The site has its own corpus retrieval software

   In this case the only specific requirement is that the provider must be able to interface the ELAN CQL with its local query engine.

2. The site has no corpus retrieval software

   In this case the specific requirements depend on the corpus exploitation software to be installed:

   Leiden software:

   Leiden software:

   For precompiled binaries:

   At least a SUN UltraSparc I, 128 Mb memory and free disk space of at least twice the size of the corpus. Operating system: Sun Solaris 2.6 or higher.

   If the software has to be built for the provider's equipment:

   Unix operating system

   Unix Make utility or (preferred) GNU make 3.77

   GNU egcs compiler release 1.0.3. The version can be checked with the g++ --version command. The answer should be egcs-2.90.29 980515 (egcs-1.0.3 release)

   GNU gcc 2.8.1 can also be used.

   GNU Bison 1.25

   Flex 2.5.4

   Java Development Kit 1.1.8 or higher

   James Clark sgml norm

   Birmingham software

   For precompiled binaries:

   No specific machine required as long as it is equipped with as much memory as possible.

   At least Java Development Kit 1.1

Storage requirements are about 75% of the original corpus file size for non-annotated material, and about 80% for annotated XML files.

If the software has to be built for the provider's equipment:

Not applicable, as JAVA binaries are delivered.

Pisa software

For precompiled binaries

At least a Pentium II-300 with 64Mb and free disk space of at least twice the size of the corpus. Operating Systems: Windows NT or Windows 95/98

If the software has to be built for the provider's equipment:

Java Development Kit 1.1 or better

Appendix

A. JDK Supported Encoding

JDK Software Release: 1.1.x

The InputStreamReader, OutputStreamWriter, and String classes can convert between Unicode and the following set of character encodings:

Character Encoding	Description
ASCII	ASCII
ISO8859_1	ISO8859_1
ISO8859_2	ISO8859_2
ISO8859_3	ISO8859_3
ISO8859_4	ISO8859_4
ISO8859_5	ISO8859_5
ISO8859_6	ISO8859_6
ISO8859_7	ISO8859_7
ISO8859_8	ISO8859_8
ISO8859_9	ISO8859_9
Big5	Big5, Traditional Chinese
Cp037	USA, Canada(Bilingual, French), Netherlands, Portugal, Brazil, Australia
Cp1006	IBM AIX Pakistan (Urdu)
Cp1025	IBM Multilingual Cyrillic: Bulgaria, Bosnia, Herzegovinia, Macedonia(FYR)
Cp1026	IBM Latin-5, Turkey
Cp1046	IBM Open Edition US EBCDIC
Cp1097	IBM Iran(Farsi)/Persian
Cp1098	IBM Iran(Farsi)/Persian (PC)
Cp1112	IBM Latvia, Lithuania
Cp1122	IBM Estonia
Cp1123	IBM Ukraine
Cp1124	IBM AIX Ukraine
Cp1250	Windows Eastern European
Cp1251	Windows Cyrillic
Cp1252	Windows Latin-1
Cp1253	Windows Greek
Cp1254	Windows Turkish
Cp1255	Windows Hebrew

Character Encoding	Description
Cp1256	Windows Arabic
Cp1257	Windows Baltic
Cp1258	Windows Vietnamese
Cp1381	IBM OS/2, DOS People's Republic of China (PRC)
Cp1383	IBM AIX People's Republic of China (PRC)
Cp273	IBM Austria, Germany
Cp277	IBM Denmark, Norway
Cp278	IBM Finland, Sweden
Cp280	IBM Italy
Cp284	IBM Catalan/Spain, Spanish Latin America
Cp297	IBM France
Cp33722	IBM-eucJP - Japanese (superset of 5050)
Cp420	IBM Arabic
Cp424	IBM Hebrew
Cp437	MS-DOS United States, Australia, New Zealand, South Africa
Cp500	EBCDIC 500V1
Cp737	PC Greek
Cp775	PC Baltic
Cp838	IBM Thailand extended SBCS
Cp850	MS-DOS Latin-1
Cp852	MS-DOS Latin-2
Cp855	IBM Cyrillic
Cp857	IBM Turkish
Cp860	MS-DOS Portuguese
Cp861	MS-DOS IcELANdic
Cp862	PC Hebrew
Cp863	MS-DOS Canadian French
Cp864	PC Arabic
Cp865	MS-DOS Nordic
Cp866	MS-DOS Russian
Cp868	MS-DOS Pakistan
Cp869	IBM Modern Greek
Cp870	IBM Multilingual Latin-2
Cp871	IBM IcELANd
Cp874	IBM Thai

Character Encoding	Description
Cp875	IBM Greek
Cp918	IBM Pakistan(Urdu)
Cp921	IBM Latvia, Lithuania (AIX, DOS)
Cp922	IBM Estonia (AIX, DOS)
Cp930	Japanese Katakana-Kanji mixed with 4370 UDC, superset of 5026
Cp933	Korean Mixed with 1880 UDC, superset of 5029
Cp935	Simplified Chinese Host mixed with 1880 UDC, superset of 5031
Cp937	Traditional Chinese Host miexed with 6204 UDC, superset of 5033
Cp939	Japanese Latin Kanji mixed with 4370 UDC, superset of 5035
Cp942	Japanese (OS/2) superset of 932
Cp948	OS/2 Chinese (Taiwan) superset of 938
Cp949	PC Korean
Cp950	PC Chinese (Hong Kong, Taiwan)
Cp964	AIX Chinese (Taiwan)
Cp970	AIX Korean
EUC_CN	GB2312, EUC encoding, Simplified Chinese
EUC_JP	JIS0201, 0208, 0212, EUC Encoding, Japanese
EUC_KR	KS C 5601, EUC Encoding, Korean
EUC_TW	CNS11643 (Plane 1-3), T. Chinese, EUC encoding
GBK	GBK, Simplified Chinese
ISO2022CN	ISO 2022 CN, Chinese
ISO2022CN_CNS	CNS 11643 in ISO-2022-CN form, T. Chinese
ISO2022CN_GB	GB 2312 in ISO-2022-CN form, S. Chinese
ISO2022JP	JIS0201, 0208, 0212, ISO2022 Encoding, Japanese
ISO2022KR	ISO 2022 KR, Korean
JIS0201	JIS 0201, Japanese
JIS0208	JIS 0208, Japanese
JIS0212	JIS 0212, Japanese
KOI8_R	KOI8-R, Russian
MS874	Windows Thai
MacArabic	Macintosh Arabic
MacCentralEurope	Macintosh Latin-2
MacCroatian	Macintosh Croatian
MacCyrillic	Macintosh Cyrillic
MacDingbat	Macintosh Dingbat

Character Encoding	Description
MacGreek	Macintosh Greek
MacHebrew	Macintosh Hebrew
MacIcELANd	Macintosh IcELANd
MacRoman	Macintosh Roman
MacRomania	Macintosh Romania
MacSymbol	Macintosh Symbol
MacThai	Macintosh Thai
MacTurkish	Macintosh Turkish
MacUkraine	Macintosh Ukraine
SJIS	Shift-JIS, Japanese
UTF8	UTF-8

B. The ELAN Workspace DTD

```
<!-- DOCTYPE ws [ -->
<!-- ======================================== -->
<!-- Silfide : Work Space                     -->
<!-- Auteur  : Patrice Bonhomme, 06 Jan 1999  -->
<!-- ======================================== -->
<!--
     PUBLIC
     "-//Silfide//DTD SIL Work Space 0.5 Draft 19990106//EN"
-->

<!ELEMENT ws (prefs,basket+,servers,histos?)>
<!ATTLIST ws crdate CDATA #IMPLIED
             update CDATA #IMPLIED>

<!ELEMENT prefs (pref*)>
<!ELEMENT pref EMPTY>
<!ATTLIST pref name  CDATA #REQUIRED
               value CDATA #REQUIRED>

<!ELEMENT basket (resource*)>
<!ATTLIST basket name CDATA #REQUIRED>
<!-- name : name of the database ??? -->

<!ELEMENT resource EMPTY>
<!ATTLIST resource idno CDATA #REQUIRED
                   sid  CDATA #REQUIRED>

<!-- href : hypertext reference URL/URI -->
<!-- key  : index key in the database   -->
<!ELEMENT servers (server*)>
<!ELEMENT server EMPTY>
<!ATTLIST server sid  CDATA #REQUIRED>

<!ELEMENT histos (histo*)>
<!ELEMENT histo EMPTY>
<!ATTLIST histo n    CDATA #IMPLIED
                date CDATA #IMPLIED>
<!-- date :a date in ISO standard form (yyyy-mm-dd) -->
<!-- ]> -->
```

C. A simple workspace instance

```
<sil type="workspace" version="0.5" sid="loria">
   <uid type="user">
      <login id="chris"/>
      <passwd></passwd>
      <access>
         <default group="guest"/>
      </access>
   </uid>
   <ws>
      <prefs>
         <pref value="yes" name="AutoSave"/><pref value="fr" name="lang"/>
         <pref value="0" name="MinutesSave"/><pref value="yes" name="EndSave"/>
         <pref value="last" name="StartBasket"/>
      </prefs>
      <basket name="ressources de birmingham">
         <resource idno="Elan-LOB" label="Elan-Corpus sample 1" idType="subcorpus"
               description="Test Corpus for ELAN" sid="birmingham" size="1188240"/>
      </basket>
      <basket name="mes resources">
         <resource idno="Elan-LOB" label="Elan-Corpus sample 1" idType="subcorpus"
               description="Test Corpus for ELAN" sid="birmingham" size="1188240"/>
         <resource idno="/xuapar/data/validated/book.ankhher/vidal.val"
               title="Spectrum van regressie en reïncarnatie"
               author="Henry de Vidal de St . Germain"
               comments="no" sid="leiden" language="Dutch" size="unknown"/>
      </basket>
      <basket name="new"/>
      <basket name="resources de leiden">
         <resource idno="/xuapar/data/validated/book.ankhher/liekens.val"
               title="NLP en spirituele ontwikkeling"
               author="Paul Liekens" comments="no" sid="leiden" language="Dutch"
               size="unknown"/>
         <resource idno="/xuapar/data/merged/jnljun95.selected_for_elan"
               title="NOS Journaal juni 1995" author="anonymous"
               comments="no" sid="leiden" language="Dutch" size="unknown"/>
         <resource idno="/xuapar/data/merged/jnljul95.selected_for_elan"
               title="NOS Journaal juli 1995" author="anonymous"
               comments="no" sid="leiden" language="Dutch" size="unknown"/>
      </basket>
      <servers>
         <server sid="godefroy"/>
```

```
        <server sid="leiden"/>
      </servers>
      <histos/>
   </ws>
</sil>
```

D. SIL DTD

```
<!-- DOCTYPE sil [ -->

<!--
        Silfide Interface Language DTD
        Version 0.5

        Draft: Thu Jan  7 22:22:57 MET 1999

        Author: Patrice Bonhomme
-->

<!-- Typical usage:

        <!DOCTYPE sil SYSTEM "http://www.loria.fr/projets/XSilfide/dtd/sil.dtd">
        <sil>
          ...
        </sil>
-->


<!-- SIL Module declaration -->

<!ENTITY % SIL.ws SYSTEM "ws.dtd">
<!-- PUBLIC "-//Silfide//DTD SIL Work Space 0.5 Draft 19990106//EN" -->
<!-- SYSTEM "http://www.loria.fr/projets/XSilfide/dtd/ws.dtd" -->
%SIL.ws;

<!ENTITY % SIL.ui SYSTEM "ui.dtd">
<!-- PUBLIC "-//Silfide//DTD SIL User Information 0.5 Draft 19990106//EN" -->
<!-- SYSTEM "http://www.loria.fr/projets/XSilfide/dtd/ui.dtd" -->
%SIL.ui;

<!ENTITY % SIL.ql SYSTEM "ql.dtd">
<!-- PUBLIC "-//Silfide//DTD SIL Query Language 0.5 Draft 19990106//EN" -->
<!-- SYSTEM "http://www.loria.fr/projets/XSilfide/dtd/ql.dtd" -->
%SIL.ql;

<!ENTITY % SIL.rs SYSTEM "rs.dtd">
<!-- PUBLIC "-//Silfide//DTD SIL Result Set 0.5 Draft 19990106//EN" -->
<!-- SYSTEM "http://www.loria.fr/projets/XSilfide/dtd/rs.dtd" -->
%SIL.rs;
```

```
<!-- SIL extensions default declaration -->

<!ENTITY % SIL.x.dtd '' >
%SIL.x.dtd;

<!ENTITY % SIL.x.ent '' >
%SIL.x.ent;

<!-- SIL extension element names -->
<!ENTITY % SIL.x.e ''>

<!-- SIL extension attribute names -->
<!ENTITY % SIL.x.a ''>

<!-- SIL module element names -->
<!ENTITY % SIL.module.e '%SIL.x.e; ws | ui | ql | rs'>

<!-- SIL module attribute names -->
<!ENTITY % SIL.module.a ''>

<!-- SIL Document structure -->

<!ELEMENT sil (uid, (%SIL.module.e;)+)>
<!ATTLIST sil type    (%SIL.x.a; workspace | user | query | resultset) "workspace"
              crdate  CDATA #IMPLIED
              update  CDATA #IMPLIED
              lang    CDATA #IMPLIED
              sid     CDATA #REQUIRED
              version CDATA #FIXED "0.5">

<!-- lang : [RFC1766] language value
     sid  : Silfide server identification -->


<!-- ================== -->
<!-- User IDentification -->
<!-- ================== -->

<!-- validated by the Silfide server admin. -->

<!ELEMENT uid   (login, passwd, access)>
<!ATTLIST uid    type (user | provider | both) "user">
```

```
<!ELEMENT login    EMPTY>
<!ATTLIST login    id ID #REQUIRED>

<!ELEMENT passwd   (#PCDATA)>

<!ELEMENT access   (default,group*)>

<!ELEMENT default   EMPTY>
<!ATTLIST default   group IDREF #IMPLIED>
<!-- cross reference to a <group id="..."> element -->

<!ELEMENT groupEMPTY>
<!ATTLIST groupid ID #REQUIRED>
<!-- id : group identification key : inalf, lingua, etc. -->

<!-- ]> -->
```

E. ResultSets DTD

```
<!-- DOCTYPE rs [ -->

<!-- ====================================== -->
<!-- Silfide : Result Set                   -->
<!-- Auteur  : Patrice Bonhomme, 06 Jan 1999 -->
<!-- ====================================== -->

<!--
     PUBLIC
     "-//Silfide//DTD SIL Result Set 0.5 Draft 19990106//EN"
-->

<!ELEMENT rs (meta, result)>

<!ELEMENT result (record)*>
<!ATTLIST result id      ID             #IMPLIED
                 type    CDATA#IMPLIED
                 content CDATA          #IMPLIED>

<!ELEMENT record (#PCDATA | attr)*>
<!ATTLIST record id    ID    #IMPLIED
                 sid   CDATA #IMPLIED
                 idno  CDATA #REQUIRED
                 count CDATA #IMPLIED>
```

F. A simple Results set instance

```
<!DOCTYPE sil SYSTEM "sil.dtd">
<sil version="0.5" type="workspace" sid="loria">
  <uid type="user">
    <login id="desaintm"/>
    <passwd>!@#$%^&*&%</passwd>
    <access>
      <default group="inalf"/>
      <group id="inalf"/>
      <group id="greco"/>
      <group id="lingua"/>
    </access>
  </uid>
  <rs>
    <meta>
      <attr name="totalSize">100</attr>
      <attr name="size">10</attr>
    </meta>
    <result type="header">
      <record sid="nancy" idno="FRW1"><attr name="title">Le Notaire de Chantilly</
attr><attr name="author">Gozlan, Léon</attr><attr name="language">Français</
attr><attr name="size">141201</attr></record>
      <record sid="nancy" idno="FRW2"><attr name="title">Mademoiselle de la Sei-
gliere</attr><attr name="author">Sandeau, Jules</attr><attr name="language">Fran-
çais</attr><attr name="size">86057</attr></record>
      <record sid="nancy" idno="FRW3"><attr name="title">Histoire Romaine</
attr><attr name="author">Michelet, Jules</attr><attr name="language">Français</
attr><attr name="size">144022 </attr></record>
      <record sid="nancy" idno="FRW4"><attr name="title">Récits des temps mérovin-
giens</attr><attr name="author">Thierry, Augustin</attr><attr name="lan-
guage">Français</attr><attr name="size">90232</attr></record>
      <record sid="nancy" idno="FRW5"><attr name="title">Sous les tilleuls</
attr><attr name="author">Karr, Alphonse</attr><attr name="language">Français</
attr><attr name="size">140807</attr></record>
      <record sid="nancy" idno="FRW6"><attr name="title">Le parfum de la dame en
noir</attr><attr name="author">Leroux, Gaston</attr><attr name="language">Fran-
çais</attr><attr name="size">124157</attr></record>
      <record sid="nancy" idno="FRW7"><attr name="title">Le Père Goriot</
attr><attr name="author">Balzac, Honoré de .</attr><attr name="language">Fran-
çais</attr><attr name="size">309234</attr></record>
     <record sid="nancy" idno="FRW8"><attr name="title">Contes cruels</attr><attr
name="author">
       Villiers de L'Isle-Adam, Auguste de</attr><attr name="language">Français</
attr><attr name="size">34564</attr></record>
      <record sid="nancy" idno="FRW9"><attr name="title">Une brève histoire du
temps</attr><attr name="author">Hawking, Stephen W.</attr><attr name="lan-
guage">Français</attr><attr name="size">???</attr></record>
```

```
      <record sid="nancy" idno="FRW13"><attr name="title">La fiancée d’Achille</
attr><attr name="author">Alki Zeï </attr><attr name="language">Français</
attr><attr name="size">???</attr></record>
    </result>
  </rs>
</sil>
```

G. Query DTDs

```
<!--
     * The MLIS/ELAN DTD
     *
     * Version : 0.21
     * Author : Pieter Masereeuw, Patrice Bonhomme, Samuel Cruz-Lara
     * Date : 1999-05-05
     * Status:  By way of experiment, setting the scope for further
                discussions.
     * Changes: the following query types have been modified:
                -  concordance type,
                -  context type,
                -  word list type, and
                -  word frequency type.
               We should also note that informations such as the
               name of a server or as the list of the subcorpora
               to be used, are not included in this DTD.
               Instead, this kind of information will be encoded
               into the MIME part of a "query".
     * To do:   The list of statistical tests is
                incomplete, as is probably the list with query types.
     *
-->

<!--
     *
     * ELAN CQL DTD - parameter entities
     *
-->

<!ENTITY % contextAttrs
           "left  CDATA #IMPLIED
            right CDATA #IMPLIED">

<!ENTITY % contextAttrsRequired
           "left  CDATA #REQUIRED
            right CDATA #REQUIRED">
<!--
     Number of context words left/right of center word.
-->

<!--
```

```
     *
     * ELAN CQL DTD - elements + attribute lists
     *
-->

<!--
     The following elements define query types.
-->

<!ELEMENT elanQuery
          (concQuery     |
           statsQuery    |
           contextQuery  |
           wordListQuery |
           wordFreqQuery |
      headerQuery)>

<!--
     Introduces a query that returns a set of concordance lines.
     Parameters :
        - optional query header
        - text query
-->
<!ELEMENT concQuery
          (headerQuery?, textQuery)>

<!ATTLIST concQuery %contextAttrs;>

<!--
     Introduces a query that returns a table with statistical
     information, dependent on the type attribute. The
     parameters for the statistical test are given within
     statParam elements.
-->
<!ELEMENT statsQuery
          (headerQuery, textQuery, statParam*)>

<!--
     Type of the statistical test to be performed (not complete,
     actually available tests advertised by local server). Depending
     on the value of the type attribute, zero or more statParam tags
     may follow.
-->
```

```
<!ATTLIST statsQuery
          type (miscore | tscore | zscore) #REQUIRED>

<!--
     Contains a word or other expression that serves as the parameter
     for the statistical test.
-->
<!ELEMENT statParam (#PCDATA)>

<!--
     Gives, where necessary, the type of the parameter, as defined by
     the type of statistical test.
-->
<!ATTLIST statParam
          paramType CDATA #IMPLIED>

<!--
     Introduces a context query.
     Note: no headerQuery and textQuery.
     The IDString is an id by which a server can send back a
     larger context than just a concordance line (the size is
     defined by the left and right attributes). The exact format
     of the id is server-specific; the client can only use values
     that were supplied earlier by the server (for example, as
     hyperlinks/anchors - note that the client should store the
     server name as well as the id. Also note that the
     subcorpora attribute of the (only) <server> tag will be ignored
     - the id should be sufficient. Finally, note that some servers
     (e.g.Leiden) may want the id to contain information about the
     (position of the) server words for the context attributes to be
     meaningful - fortunately,all this stuff is of no concern for the
     client as long as it stores the id.
-->
<!ELEMENT contextQuery
          (IDString)>

<!ELEMENT IDString (#PCDATA)>

<!ATTLIST contextQuery %contextAttrsRequired;>

<!--
     Introduces a word list query type.
     The wildCard contains a regular expression or a CQL expression
```

```
     for which all matching words should be returned in a tabular
     format, with or without frequency information. All frequencies
     are given relative to the current (sub)corpus selection.
-->
<!ELEMENT wordListQuery
          (headerQuery?, wildCard)>

<!--
     Format of word list. Default is to return words plus frequency
     information.
-->
<!ATTLIST wordListQuery
          output (word | wordPlusFreq) "wordPlusFreq">

<!--
     A wildCard may be a CQL or a regular expression.
     Does anybody remember in what situation is this word list
     going to be used ???
-->
<!ELEMENT wildCard (#PCDATA)>

<!ATTLIST wildCard
          type (regex | textQuery) #REQUIRED>

<!--
     Note: no textQuery.
     Contains a word for which the frequency is to be returned. The
     frequency is given relative to the current (sub)corpus
     selection.
-->
<!ELEMENT wordFreqQuery
          (headerQuery?, word)>

<!ELEMENT word (#PCDATA)>

<!--
     End of query type elements; start of CQL proper elements.
-->

<!--
     Contents is a CQL expression that uses the TEI header for
     subcorpus selection.
-->
```

```
<!ELEMENT headerQuery (#PCDATA)>

<!--
     Contents is a CQL expression that works on the contents of
     the documents (within <TEXT>). For CDATA see above.
-->
<!ELEMENT textQuery (#PCDATA)>

<!-- ELAN Result Set -->

<!ELEMENT elanResultSet (meta, result)>
```

H. Sample code of a “fake” QueryHandler driver

```
/*
 * @(#)       QueryHandlerTestImpl.java
 *
 * Copyright  1999 (C) CHRISTOPHE DE SAINT-RAT
 *            UMR LORIA (Universities of Nancy, CNRS & INRIA)
 *
 */

package fr.loria.silfide.driver;

import fr.loria.silfide.util.*;
import fr.loria.silfide.query.*;
import fr.loria.silfide.server.prefs.*;

import java.util.Hashtable;
import java.util.Random;
import java.util.Enumeration;
import java.io.StringReader;

import fr.loria.mlis.elan.*;
import fr.loria.xml.DocumentLoader;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import fr.loria.silfide.sil.*;

/**
 * <code>QueryHandlerTestImpl</code> is a fake ELAN driver implementing
 * the <code>QueryHandler</code> interface. Its purpose is to show how
 * to write a driver. It generates and returns fake results (documents
 * or concordance lines).
 *
 * @version1.0
 * @authorChristophe de Saint-Rat
 */
public class QueryHandlerTestImpl implements QueryHandler
{
    /**
     * list of last names to generate fake authors when generating documents
     */
    private static String tabLastNames[] = {"romary","husson","viscogliosi","bon-
homme","cruz-lara"};
```

```
    /**
     * list of first names to generate fake authors when generating documents
     */
    private static String tabFirstNames[] = {"nadia","jean-luc","patrice","lau-
rent","samuel"};
    /**
     * list of languages to generate fake languages when generating documents
     */
    private static String langues[] = {"francais","allemand","japo-
nais","anglais","espagnol"};
    /**
     * list of words to generate fake results
     */
    private static String mots[] = {"cette","unité","dont","la","créa-
tion","a","été",
                "officialisée","le","19","décembre","1997","par",
                "la","signature","du","contrat","quadriennal","avec",
                "le","ministère","de","l'education","nationale","de",
                "la","recherche","et","de","la","technologie","par",
                "une","convention","entre","les","cinq","partenaires",
                "succède","ainsi","au","centre","de","recherche","en",
                "informatique","de","Nancy","(CRIN)","et","aux",
                "équipes","communes","entre","celui-ci","et","l'unité",
                "de","recherche","INRIA","de","Lorraine"};


    /**
     * number of fake resources to generate
     */
    private static int NB_RESOURCES = 300;


    /**
     * maximum number of resources to be returned at once
     */
    private static int MAX_RETURN = 50;


    /**
     * local server sid
     */
    private String sid;


    /**
     * sessions table
     */
    private Hashtable sessions;
```

```
 /**
  * queries table
  */
 private Hashtable queries;

 /**
  * random number generator used to generate the fake results
  */
 private Random rand;

 /**
  * XML document loader
  */
 private DocumentLoader loader;

 /**
  * returns a fake idno
  */
 private String getIdno()
 {
return Long.toString(Math.abs(rand.nextInt()));
 }

 /**
  * returns a fake first name
  */
 private String getFirstName()
 {
int x=(new Double(tabFirstNames.length*Math.abs(rand.nextFloat()*)).intValue());
if (x>=tabFirstNames.length) x= tabFirstNames.length-1;
return tabFirstNames[x];
 }

 /**
  * returns a fake last name
  */
 private String getLastName()
 {
int x=(new Double(tabLastNames.length*Math.abs(rand.nextFloat()*)).intValue());
if (x>=tabLastNames.length) x= tabLastNames.length-1;
return tabLastNames[x];
 }
```

```
 /**
  * returns a fake language attribute for the resource
  */
 private String getLangue()
 {
int x=(new Double(langues.length*Math.abs(rand.nextFloat())).intValue());
if (x>=langues.length) x= langues.length-1;
return langues[x];
 }

 /**
  * returns a fake language title for the resource
  */
 private String getTitle()
 {
StringBuffer s=new StringBuffer();
for (int i=0;i<4;i++)
    {
   int x=(new Double(mots.length*Math.abs(rand.nextFloat())).intValue());
   if (x>=mots.length) x= mots.length-1;
   s.append(mots[x]);
   s.append(" ");
    }
return s.toString();
 }

 /**
  * returns a fake size attribute for the resource
  */
 private String getTaille()
 {
return Integer.toString(Math.abs(rand.nextInt()));
 }

 /**
  * returns a fake comments attribute for the resource
  */
 private String getCommentaire()
 {
StringBuffer s=new StringBuffer();
for (int i=0;i<20;i++)
    {
```

```
   int x= (new Double(mots.length*Math.abs(rand.nextFloat())).intValue());
   if (x>=mots.length) x= mots.length-1;
   s.append(mots[x]);
   s.append(" ");
    }
return s.toString();
 }

 /**
  * creates and returns a new fake resource with all attributes in a hashTable
  */
 private Hashtable getNewResource()
 {
Hashtable h = new Hashtable();
h.put("idType","document");
h.put("title",getTitle());
h.put("author",getFirstName()+" "+getLastName());
h.put("language",getLangue());
h.put("size",getTaille());
h.put("comments",getCommentaire());
return h;
 }

 /**
  * instanciation of the new driver and initialization
  */
 public QueryHandlerTestImpl()
 {
ServerPrefs prefs = new ServerPrefs();
sid = prefs.getPref("name");
rand = new Random(System.currentTimeMillis());
sessions = new Hashtable();
queries = new Hashtable();
loader = new DocumentLoader();
 }

 /**
  * perform a new query with no associated basket (usually a header query)
  *
  * @paramsessionIDthe session ID of the new query
  * @paramquerythe query as a SIL document XML string
  */
 public ElanResultSet executeQuery(String sessionID, String query)
```

```
 {
// decoding of the query
AbstractElanQuery eq=null;
String qh="";
try
    {
      // instanciation of a reader to read the query string sequentially
      StringReader sr = new StringReader(query);
      // instanciation of a new SIL document processor
      SILProcessor sp = new SILProcessor();
      // creation of an XML Document corresponding to the XML string
      Document se = loader.loadDocument(sr);
      // parsing of the XML document
      SIL sil = sp.process(se);
      // extraction of the ElanQuery part of the SIL document
      eq = (AbstractElanQuery)sil.getModule();
      switch(eq.getQueryType()) // gets the query type
  {
  case ElanQuery.QUERY_HEADER:
      qh = eq.getHeaderQuery(); // gets the header query text
      break;
  case ElanQuery.QUERY_CONC:
  case ElanQuery.QUERY_CONTEXT:
  case ElanQuery.QUERY_STATS:
  case ElanQuery.QUERY_WORDFREQ:
  case ElanQuery.QUERY_WORDLIST:
      qh = eq.getTextQuery(); // gets the content query text (CQL)
      break;
  default:
      qh = "";
  }
     System.out.println("got Query from : "+sil.getUserID().getLogin());
    }
catch(Exception e)
    {
      System.out.println("query parsing error : "+e.toString());
    }

Integer icpt = new Integer(0);
sessions.put(sessionID,icpt);
int cpt = 0;

if (!"getAvailableCorpora".equals(qh))
```

```
    {
   // returns documents references that match the query

   // create a new result set object
   ElanResultSetImpl rsp = new ElanResultSetImpl(new Meta(),new Result());

   // sets the total number of documents matching the query
   rsp.setTotalSize(NB_RESOURCES);

   // add each individual documents references to the result set
   for(int i=0;i<MAX_RETURN && cpt<NB_RESOURCES;i++)
       {
      // add a document (sid+idno) + a list of display attributes
      rsp.addRecord(sid,getIdno(),null,getNewResource());
      cpt++;
       }

   sessions.put(sessionID,new Integer(cpt));
   if (eq!=null) queries.put(sessionID,eq);
   return rsp;
    }
else
    {
   // returns all available sub-corpora.
   // if you do not support this feature, you should return an identifier
   // for your whole corpus
   ElanResultSetImpl rsp = new ElanResultSetImpl(new Meta(),new Result());
   rsp.setTotalSize(1);
   Hashtable corpora = new Hashtable();
   corpora.put("idType","subcorpus");
   corpora.put("label","all corpora");
   corpora.put("description","the complete "+sid+"'s corpus");
   corpora.put("size","1000000");
   rsp.addRecord(sid,"allCorpora",null,corpora);
   sessions.put(sessionID,new Integer(cpt));
   if (eq!=null) queries.put(sessionID,eq);
   return rsp;
    }
 }

 /**
  * perform a new query with an associated basket (usually a content query)
  *
```

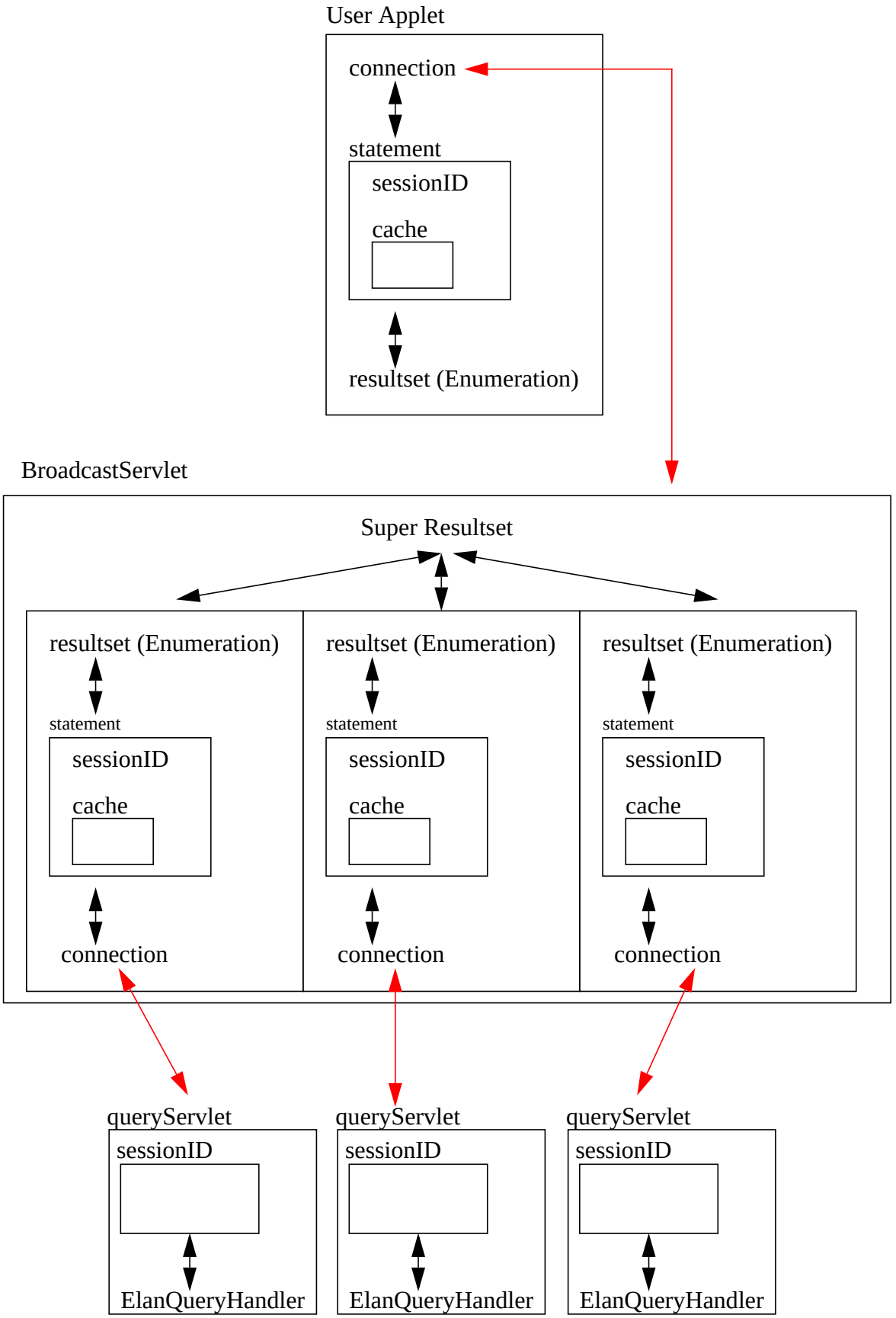
```
     * @paramsessionIDthe session ID of the new query
     * @paramquerythe query as a SIL document XML string
     * @parambasketthe basket as a SILBasket document XML string
     */
    public ElanResultSet executeQuery(String sessionID, String query, String bas-
ket)
    {
// System.out.println(query);
// System.out.println("------------------------------");
// System.out.println(basket);
   AbstractElanQuery eq=null;
   String qh="";
   try
       {
         StringReader sr = new StringReader(query);
         SILProcessor sp = new SILProcessor();
         Document se = loader.loadDocument(sr);
         SIL sil = sp.process(se);
         eq = (AbstractElanQuery)sil.getModule();
         switch(eq.getQueryType())
     {
     case ElanQuery.QUERY_HEADER:
         qh = eq.getHeaderQuery(); // gets the header query text
         break;
     case ElanQuery.QUERY_CONC:
     case ElanQuery.QUERY_CONTEXT:
     case ElanQuery.QUERY_STATS:
     case ElanQuery.QUERY_WORDFREQ:
     case ElanQuery.QUERY_WORDLIST:
         qh = eq.getTextQuery(); // gets the content query text (CQL)
         break;
     default:
         qh = "";
     }
        System.out.println("got Query from : "+sil.getUserID().getLogin());
      }
   catch(Exception e)
       {
         System.out.println("query parsing error : "+e.toString());
       }

   // decoding of the basket
   try
       {
```

```
      // instanciation of a reader to read the basket string sequentially
      StringReader sr = new StringReader(basket);
      // creation of an XML Document corresponding to the XML string
      Document sbd = loader.loadDocument(sr);
      // extraction of the root element from the basket document
      Element sbe = sbd.getDocumentElement();
      // parsing of the basket element
      Basket sb = SILBasket.process(sbe);
      // extraction of the basket resources
      for (Enumeration en = sb.getResources();en.hasMoreElements();)
          {
        Resource r = (Resource)en.nextElement();
        String r_sid = r.getSid();
        if (r_sid.equals(sid)) // verify if the resource is on this local server
            {
           String idno = r.getIdno(); // gets the resource identifier
            }
         }
       }
    catch(Exception e)
        {
          System.out.println("basket parsing error : "+e.toString());
        }


    Integer icpt = new Integer(0);
    sessions.put(sessionID,icpt);
    int cpt = 0;

    ElanResultSetImpl rsp = new ElanResultSetImpl(new Meta(),new Result());
    rsp.setTotalSize(NB_RESOURCES);
    Hashtable htres = new Hashtable();
    for(int i=0;i<MAX_RETURN && cpt<NB_RESOURCES;i++)
        {
//    rsp.addRecord(sid,getIdno(),qh+" "+sessionID+" "+cpt);
      htres.put("leftContext",getCommentaire());
      htres.put("node",qh);
      htres.put("rightContext",getCommentaire());
      htres.put("info",sessionID+" "+cpt);
      rsp.addRecord(sid,getIdno(),null,htres);
      cpt++;
      htres.clear();
       }
```

```
sessions.put(sessionID,new Integer(cpt));
if (eq!=null) queries.put(sessionID,eq);

return rsp;
 }

 /**
  * ask for the next results of the former query identified my sessionID
  *
  * @paramsessionIDthe session ID of the former query
  */
 public ElanResultSet next(String sessionID)
 {
Integer icpt = (Integer)sessions.get(sessionID);
if (icpt==null) return null;
int cpt = icpt.intValue();
if (cpt>=NB_RESOURCES) return null;
ElanResultSetImpl rsp = new ElanResultSetImpl(new Meta(),new Result());
rsp.setTotalSize(NB_RESOURCES);
String query=null;
ElanQuery eq = null;
try
    {
      eq = (ElanQuery)queries.get(sessionID);
      if (eq!=null)
  {
    switch(eq.getQueryType())
      {
      case ElanQuery.QUERY_HEADER:
     query = eq.getHeaderQuery();
     break;
      case ElanQuery.QUERY_CONC:
      case ElanQuery.QUERY_CONTEXT:
      case ElanQuery.QUERY_STATS:
      case ElanQuery.QUERY_WORDFREQ:
      case ElanQuery.QUERY_WORDLIST:
     query = eq.getTextQuery();
     break;
      default:
     query = "";
      }
  }
```

```
        }
    catch(Exception e) {}
    if (ElanQuery.QUERY_HEADER==eq.getQueryType())
        for(int i=0;i<MAX_RETURN && cpt<NB_RESOURCES;i++)
      {
          rsp.addRecord(sid,getIdno(),null,getNewResource());
          cpt++;
      }
    else
       for(int i=0;i<MAX_RETURN && cpt<NB_RESOURCES;i++)
      {
          rsp.addRecord(sid,getIdno(),query+" "+sessionID+" "+cpt);
          cpt++;
      }
    sessions.put(sessionID,new Integer(cpt));
    return rsp;
     }

     /**
      * close the session identified by sessionID
      *
      * @paramsessionIDthe session ID of the former query
      */
     public void close(String sessionID)
     {
    if (sessions.containsKey(sessionID))
        sessions.remove(sessionID);
    if (queries.containsKey(sessionID))
        queries.remove(sessionID);
     }
}

//EOF QueryHandlerTestImpl
```

I. General communication model between servers in the ELAN Network


User Applet
connection
statement
sessionID
cache
resultset (Enumeration)
BroadcastServlet
Super Resultset
resultset (Enumeration)
statement
sessionID
cache
connection
resultset (Enumeration)
statement
sessionID
cache
connection
resultset (Enumeration)
statement
sessionID
cache
connection
queryServlet
sessionID
ElanQueryHandler
queryServlet
sessionID
ElanQueryHandler
queryServlet
sessionID
ElanQueryHandler