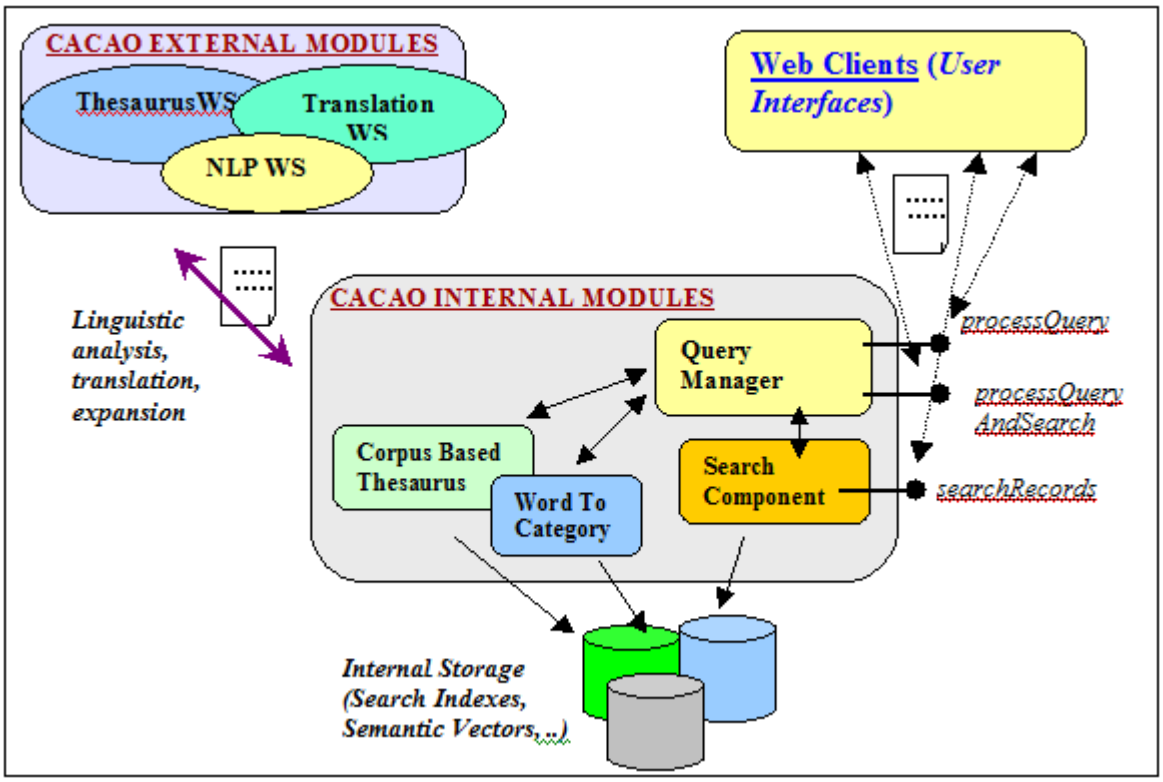
# Interacting with CACAO Web Services: UI Scenarios

## 1 Summary: CACAO CLIR




The functionalities of the CACAO CLIR engine emerge from the interactions of a set of internal and external modules. The internal components consist of "linguistic agnostic" modules that contain the core logic of the system and focus on the management of the information harvested from libraries and the data inferred from it (i.e. search indexes, corpus-based semantic vector, terms associated to library categories). The external components instead provide the CACAO architecture with linguistic analysis capabilities and resources (i.e. lemmatization and named entities recognition, bilingual dictionaries, thesauri) and consist of Web Services that adhere to the communication interface required by the CLIR system.

## 2 WEB SERVICES

The functionality of CACAO CLIR system are exposed to clients (user interfaces) through 2 Web services.

## 2.1 SearchWS

**searchWS** (i.e. http://www.cross-library.com:8080/axis2/services/CACAO_searchWS_2), performs the actual search task on the records collected from the libraries and stored in the search index and returns an XML documents with details on facets and records retrieved for the query.

```
<CACAO_ResponseDocument>
 <CACAO_Facets>
        <FacetField field="libraryID"/>
        <FacetField field="language"/>
        <Facet count="3502" filed="libraryID" value="GOETTINGEN"/>
        <Facet count="2747" filed="libraryID" value="Test telap"/>
        <Facet count="6" filed="libraryID" value="MEK"/>
          <Facet count="2" filed="libraryID" value="Repository of Pole HSTL, CNRS - Centre
Alexandre-KOYRE/CRHST"/>
        <Facet count="4143" filed="language" value="en"/>
        <Facet count="887" filed="language" value="fr"/>
        <Facet count="728" filed="language" value="it"/>
        <Facet count="642" filed="language" value="de"/>
    </CACAO_Facets>
    <CACAO_RetrievedRecords recordsFound="7559" recordsReturned="10" recordsStartFrom="0">
        <Record recordRelevance="0.0841985">
            <libraryID>FUB Test repository</libraryID>
            <recordID>oai:unibz.it:library:opac/0359251</recordID>
                                    <telRecordID_URL>http://pro.unibz.it/opacuni/search.asp?
mediennr=0359251</telRecordID_URL>
            <ISBN>88-14-10093-4</ISBN>
            <OAI_set>DEFAULT</OAI_set>
            <title>L' economia mondiale : una prospettiva millenaria</title>
            <subject>Economia mondiale</subject>
            <subject>QM 000</subject>
            <description>Tit. orig.: ¬The¬ world economy</description>
            <contributor>fub:person &gt; Graziola, Giancarlo</contributor>
            <publisher>Milano, Giuffrè</publisher>
            <type>dcterms:DCMIType &gt; Text</type>
            <language>it</language>
              <creator>fub:responsibility &gt; Angus Maddison ; presentazione all' edizione
italiana di Giancarlo Graziola</creator>
            <creator>Maddison, Angus</creator>
        </Record>
        <Record recordRelevance="0.07679898">
```

The service is based on a SOLr engine that is embedded within the web service and exposes a single operation:

- *String searchRecords*(String query, String[] facetFieldNames, int startFrom, int totResults)

the **query** string must be conform to the Lucene Query Syntax (see http://lucene.apache.org/java/2_3_2/queryparsersyntax.html), **facetFieldNames** can be null if no

faceting is desired or contain any fieldnames that the user want as a search facet. (any field in the search index that has been tokenized using a keyword tokenizer is a valid facet field candidate) **startFrom** and **totResults** are used for paging.

## 2.2 QueryWS

**queryWS** (i.e. http://www.cross-library.com:8080/axis2/services/CACAO_queryWS) receives as input the monolingual request from users and returns a document listing the translations and expansion of the search terms from the monolingual request; it also compose a default query using the original terms and translations (+ the expansions when present) and includes it in the reponse document (see an example of response document below)

```
<CACAO_ResponseDocument>
    <CACAO_InputQuery inputLanguage="en">
        <SearchQuery>title:(économie)^2.0 subject:(économie)^1.5 tableOfContents:
(économie)^1.0 description:(économie)^1.0 title_lemmatized:(économie_noun)^2.0   title:
(Keynes)^2.0 subject:(Keynes)^1.5 tableOfContents:(Keynes)^1.0 description:(Keynes)^1.0
namedEntity_PERSON:(Keynes)^3.0 contributor:(Keynes)^3.0 creator:(Keynes)^3.0 </SearchQuery>
        <SearchTerm category="NOUN" confidence="1.0" isNE="false"
            isTranslation="true" lang="fr" lemma="économie">économie</SearchTerm>
        <SearchTerm category="NOUN" confidence="1.0" isNE="false"
            isTranslation="true" lang="fr" lemma="économe">économe</SearchTerm>
        <SearchTerm category="NOUN" confidence="1.0" isNE="false"
            isTranslation="true" lang="fr" lemma="économique">économique</SearchTerm>
        <SearchTerm NEcategory="PERSON" category="NOUN" confidence="1.0"
            isNE="true" lang="fr" lemma="Keynes">Keynes</SearchTerm>
        <SearchTerm category="NOUN" confidence="1.0" isNE="false"
            lang="en" lemma="economy">economy</SearchTerm>
        <SearchTerm NEcategory="PERSON" category="NOUN" confidence="1.0"
            isNE="true" lang="en" lemma="Keynes">Keynes</SearchTerm>
```

The **queryWS** Web Service exposes th following operations

- String *processQuery*(String query, String langFrom, String[] targetLanguages);
- String *processQueryCustom*(String query, String langFrom, String[] targetLanguages, boolean translateWithSystran, boolean expandQueryWithSemVect, boolean expandWithWordNet, boolean expandWithWordToCat )

Both of the previous operations translate the original query in the targetLanguages[]. However the *processQueryCustom* operation allows for requesting specific query expansions. The *processQuery* operation instead expands the search terms according to the system default setting (NO for the actual demo)

The **queryWS exposes** an additional set of operations that compound in single call a consecutive request to the queryWS and then to the searchWS: first processing the query (using one of the previous

*processQuery* operations) and then subsequentely calling the *searchRecords* operation of the **searchWS** with the lucene query specified in the <SearchQuery> element of the **queryWS** response document.

- String *processQueryAndSearch*(String query, String langFrom, String[] targetLanguages, String[] facetsFields, int startFromResult, int totResults);
- String *processQueryAndSearchCustom*(String query, String langFrom, String[] targetLanguages, String[] facetFieldNames, int startFrom, int totResults, boolean translateWithSystran, boolean expandQueryWithSemVect, boolean expandWithWordNet, boolean expandWithWord2Cat)

The first operation uses the default options for the expansion while the second one allows for the request of specific expansions.

# 3 Interacting with the Web Services

**queryWS** *processQuery operation* translates and expands the original input query and in the response document it includes a lucene query automatically composed by the service using the translated/expanded search terms. Such lucene query specifies to search for each <SearchTerm> in specific fields of the search index and with different weights (i.e. **`title:(économie)^2.0 subject:(économie)^1.5 .. )`**. The automatic query generations specify for each <SearchTerm> a set of default fields (title, subject, description) and if the term is recognized to be a named entity (see Keynes in the previous example) additional fields are used (creator, ..)

The so generated lucene query can be inputed to the **searchWS** as it is (ending in the same result as if processQueryAndSearch was invoked) or it can be programmatically modified and then submitted to the searchWS. Another option (for very specific or customized interfaces) would be to programmatically build the lucene query from the <SearchTerm> elements of the response document.

The ending query MUST still be compliant to the lucene query syntax.

## *3.1 Manipulation examples*

Two examples of programmatic query manipulation concerns faceting and user proposed translations.

- A facet constraint can be added to the service generated query with the formalism facetFiled:restrictionValue preceded by the '+' operator that states that the term **must** be present in the resultset (i.e. by adding +language:en when the user selects “en” from the facets cloud in the interface)
- In order to add a new search terms in a existing query already submitted and constituting the lucene query to be manupulated, here Q0 (i.e. the uses proposes a new keyword for the search: “dog” ); in this situation the user interface can obtain the translation/expansion of the proposed term by invoking the *processQuery operation* of the **queryWS** and then append the lucene query present in the response document (*title:(chienne)^2.0 subject:(chienne)^1.5 ...*) to Q0 and then submitt the resulting query to the **searchWS**
- If the user proposes an alternative translation for a given term the lucene query must be

generated programmatically by removing the wrong translation from the list of <SearchTerms> and adding the proposed one and then generate the lucene query.

- Any valid search parameters can be added programmatically to the query on response to some user interaction. (i.e. creator:Dante )