SoLDES : Service-oriented Lexical Database Exploitation System

Mehdi Ben Abderrahmen[1], Bilel Gargouri[2], and Mohamed Jmaiel[1,3]

[1] ReDCAD Laboratory, University of Sfax
ENIS, B.P 1173, 3038 Sfax, Tunisia
`mehdi.benabderrahmen@redcad.org`
[2] MIRACL Laboratory, University of Sfax
FSEGS, BP. 1088, 3018 Sfax, Tunisia
`bilel.gargouri@fsegs.rnu.tn`
[3] Digital Research Center of Sfax,
B.P. 275, Sakiet Ezzit, 3021 Sfax, Tunisia
`mohamed.jmaiel@enis.rnu.tn`

Abstract In this work, we focuses on the assisted exploitation of lexical databases designed according to the LMF standard (Lexical Markup Framework) ISO-24613. The proposed system is a service-oriented solution which relies on a requirement-based lexical web service generation approach that expedites the task of engineers when developing NLP (Natural Language Processing) systems. Using this approach, the developer will neither deal with the database content or its structure nor use any language query. Furthermore, this approach will promote a large-scale reuse of LMF lexical databases by generating lexical web services for all languages. For evaluating this approach we have tested it on the Arabic language.

Keywords: LMF, Lexical Database, Exploitation, Requirement, Interrogation, Web Service, Automatic Generation.

1 Introduction

The majority of NLP systems require lexical resources which make lexical component one of the most important in the field of NLP. Lexical resources are the key element for NLP systems. On the basis of the different needs, several studies dealt with modeling and implementing these resources in different forms (i.e. simple lexicons, relational lexical databases, XML lexical databases). They tried to cover most of the linguistic levels [6]. However, these studies have some drawbacks that can be divided into two classes: some problems related to the lexical product contents (i.e. linguistic coverage) and others related to their interrogation (i.e. integration with the NLP systems). For the first class, it can be noted that lexical products remain very dependent on target NLP systems in the choice of lexical entry structures and the implementation technology. The linguistic coverage is consequently limited. For the second class, the diversity of model and platform implementation limits the possibilities of interoperability and reuse, in particular, when integrating the same lexical product into different NLP systems. This diversity leads to the heterogeneity of the interrogation mode, update and result presentation.

In order to standardize lexical database design, the internal description format and the multilingual representation, the LMF (Lexical Markup Framework) [7], which is a novel standard under the reference ISO-24613, was proposed. This project proposes an approach for the design of XML databases starting from a single meta-model in UML for all the programming languages and enables to represent multilingual lexical data in connection with all the linguistic levels. Some illustrations of LMF were already proposed (i.e. El-Madar Dictionary for the Arabic language [11] [12] and Morphalou [14] database for the French language). However, the LMF project is interesting only in lexical data representation. It has not yet covered the interrogation and exploitation of the lexical database for the possible needs of the NLP systems.

In this work, we are interested in the use of LMF lexical databases in order to satisfy the whole requirements (i.e. NLP systems requirements, user requests, import and export of external resources). In particular, we focus on the NLP system requirements. In this context, we propose an oriented service based system for the exploitation of the lexical services of LMF standard lexical databases. This system considers two main phases in the deployment and exploitation of LMF services:

The first phase corresponds to the set up of a service oriented architecture for LMF services allowing the interrogation of LMF database. It covers three main steps starting from i) the specification of the main NLP requirements, then ii) their formulation to concrete queries and finally iii) the implementation and the deployment of web services that execute these queries to interrogate LMF database. In this phase, all the steps are realized off-line. An NLP application could then find out a specific web service among the offered ones to interrogate the LMF database. This proposal made the task of engineers easier when developing NLP applications by discharging them from mastering the database structure, a query language or even from formulating queries.

The second phase corresponds to the proposal of an automatic lexical web service generation approach based on the NLP requirements. In fact, in the case where an NLP application developer couldn't find a web service corresponding to his specific requirement, the approach enables to handle this specific requirement online and generate then automatically the corresponding query and web service. The resulted service is then added to the lexical web service library to enrich the exploratory capacity of the given LMF database.

The remainder of this paper is organized as follows: in the next section, we will present our service-oriented LMF database exploitation system that we called SoLDES. We will present the different steps leading to the realization of such system. Then in the third section we will present a requirement-based approach for the generation of lexical web services. We will give details about the lexical web service generation tool developed in this context. After that, we will give some illustrations of the use of our system and examples of the generation of some lexical services starting from the expression of specific requirements. Before the conclusion, we will show some related works that tried to offer solutions in order to have access to lexical resources in general.

2 Service-Oriented LMF Database Exploitation System: SoLDES

The exploitation and the reuse of lexical databases in general, go through the provision of a well designed interface to query the stored lexical data. The development of NLP (Natural Language Processing) applications becomes then easier and more structured. Such interface could be seen as a specific querying system dedicated to Standard Lexical Database like LMF databases. In this direction, we propose our service-oriented exploitation system called SoLDES to interrogate LMF databases (fig.1). This section will give an overview of the SoLDES system then detail the three main steps leading to the realization of such system.

2.1 SoLDES Overview

SoLDES, as any service-oriented system, provides an extensible web service library for NLP application developers. These web services are lexical services allowing the interrogation of LMF databases. They are developed based on NLP system requirements depicted as queries. In fact, the NLP requirements in term of lexical resources were firstly identified. Then we have matched to each requirement a corresponding query (cf. 2.2). Similar queries were gathered to provide parameterized queries (cf. 2.3). Finally, we have designed a service model for each query pattern (cf. 2.4). Hence, the proposed services are developed regarding the depicted service model corresponding to the NLP requirements. Once the web services are developed and hosted in the service library, their corresponding WSDL file (contains the web service functionality description) is generated and published into a specific UDDI representing a lexical service catalog. This service directory enables the full deployment of lexical web services. It will be consulted by NLP application developers looking up for a specific lexical service fulfilling their requirements. If a service is found, the developer subscribes to this service to enable its invocation and achieve the NLP application development. It should be noted that sometimes, a specific requirement could not be satisfied by only one lexical service, but rather by several ones. For this reason, we consider a service composition layer in the client side which enables the web service composition to provide an added-value service that could meet complex lexical requirement of NLP applications. We will detail an example in (cf. 4). The SoLDES system presented could be seen within a business model involving three main actors:

- Lexical Service Provider LSP: this is the actor who develops and provides lexical web services. He is in direct contact with LMF database.

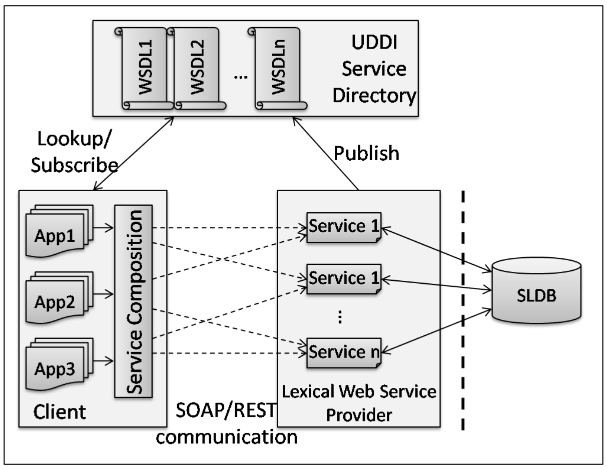

Fig. 1. Service-oriented LMF Database Exploitation System: SoLDES

– Lexical Service Client LSC: he is an actor who will consume lexical services, mainly represented by NLP applications;
– Lexical Service Directory LSD: this is the UDDI directory allowing the publication of lexical web services by LSP. It offers the LSC lookup tools to look for a specific lexical service and make subscription once found.

In this way, the only concern of a client using SoLDES is to consult the LSD and invoke the needed lexical service that will directly provide required lexical information from LMF database. Hence, the client is discharged from likely arduous tasks like:

– Knowing and mastering the structure of LMF database, its DTD or XML schema, used extensions, connection mechanisms;
– Learning a query language (like XQuery) and mastering the query formulation, even the complicated ones, usage flexibility, etc. The interaction with LMF database is used exclusively through web services;
– Ensuring integration of LMF database to NLP applications, no more effort will be needed since the web services became the most widespread integration tool.

The client, in general, only has to understand the web service functionality which is independent from both the implementation technologies and the LMF database location. Hereafter, we will detail the three main processes involved in the SoLDES system, namely, the requirements identification, the query formulation and the web service implementation.

2.2 Requirement Identification

Starting with NLP leader applications such as “Segmenter”, “Lemmatizer” and “Morphological Analyzer”, the first step is to determine the set of lexical needs required by these applications according to the studied linguistic level (morphological, syntactic or semantic). As shown in Fig.2, a NLP application (i.e. App.1) can use one or more linguistic levels. Indeed, the “Morphological Analyzer”, for example, requires only a morphological analysis but the “Segmenter” needs, the syntax analysis, as well. Each level expresses a number of lexical needs to be met from lexical database (LDB). A lexical database that covers all the aspects of a natural language must provide lexical information about all the linguistic levels and thus will satisfy all types of needs that can express an NLP application. During the requirement acquisition step we tried to identify the most generic needs by a flattened way starting with basic needs. Thus, the most complex needs can be solved through a decomposition into basic needs. The requirement identification operation requires the involvement of linguistic experts (in our case, the target language is Arabic) and the NLP application developers.

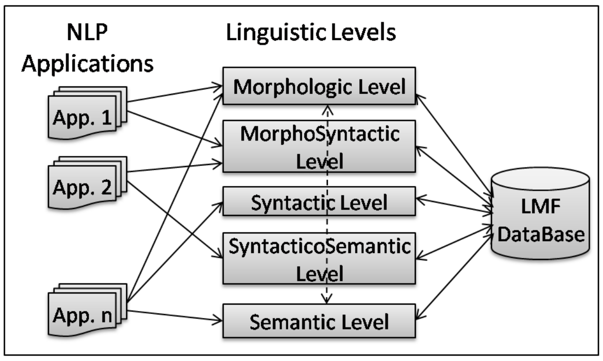

Fig. 2. Requirement identification through linguistic level

2.3 Query Formulation

The requirements identified in the previous step are expressed in a human-readable manner (abstract form). In order to make them understandable using a machine, they have to be transformed and formulated as queries (concrete form). In fact, for each requirement, we have matched a query expressed manually with XQuery language. This transformation process requires good skills and knowledge of both XQuery syntax and LMF database structure. This is what we have actually done to spare the complexity for lexical service clients. The result of the transformation is a set of various queries. Normally, the next step is to implement a web service to execute each query. However, we noticed the existence of similarity between some queries having the same output type. Indeed, these similar queries correspond, in general, to the same requirement but with different input parameters. In this case, we gathered each set of similar queries into one parametrized query implemented by only one configurable service (service with several methods) instead of having a web service for each similar query. The main objective of this operation is organizational. For example, there could exist two similar queries: the first deals with the consistency between Prefixes and Suffixes and the second with the consistency between Prefixes, Infixes and Suffixes. The similarity here resides in the inputs ”Prefix” and ”Suffix” common in both queries. As a result, only one service will be devoted to both queries but with different capabilities.

2.4 Web Service Implementation

After the formulation of the query emerging from NLP requirement identification, the development of the web service will be performed. Indeed, a service is an application entity that handles several tasks, like:

– The connection to the lexical database. In our case, the connection and then the interrogation of the lexical database are made through DataDirectX-Query implementing the XQJ (API XQuery for Java);
– Catching NLP requirements as input parameters for the service. This information is read as external variables whose values will be injected in the XQuery query;
– Executing the query and giving back result to the NLP application.

A service could be composed of one or several methods according to the requirement type (elementary or complex). All the aforementioned tasks are achieved as processing within the service methods. Therefore, the granularity of a service is defined by the association of a service method with an elementary requirement. Hence, a service with several methods generally fulfills a complex requirement. When a service contains only one method, the corresponding requirement should be simple (elementary).

For example, a consistency service that asks whether the pair (affixes, root) is consistent or not. This service contains several methods as shown below.

```
boolean coherenceRootSuffix (String root, String suffix)
boolean coherencePrefixSuffix (String prefix, String suffix)
boolean coherencePrefSufInf (String prefix, String suffix, infix String)
```

The development of a lexical web service is followed by the generation of a standard WSDL file description. This file contains a service capability description which will help index and reference the service within a lexical service directory (UDDI). The service capability description precises for each method its inputs and output types and the service address. The deployment of a lexical web service is done by adding both the WSDL file to the Lexical service directory (UDDI) and the lexical web service to the service library. The lexical web service is hence ready for use by NLP application. The latter has only to look for lexical web service corresponding to its requirement to exploit LMF database capabilities.

2.5 Discussion

The SoLDES system as proposed made the task of engineers easier when developing NLP systems by discharging them from mastering the database structure, a query language or even from formulating queries. The interrogation of an LMF database was transformed to a web service application allowing the fluency of the interoperability between NLP system and LMF databases. However, since the SoLDES system is based on a three-step process performed manually and offline, a consistent problem is raised; What happens when an NLP system lexical requirement doesn't match any lexical web service provided by the library? This will lead to the limitation of the exploitation of lexical databases. Although the requirement identification step has involved linguistic experts, all requirements couldn't be conceived and then expressed. There are always some NLP applications that have a specific requirement not already expressed before and should be considered and covered. For this reason, we proposed as a second phase of SoLDES system, a lexical web service generation approach enabling to overcome this problem. This approach enables to automate all the steps leading to the generation of a lexical web service. This is detailed in the next section.

3 Requirement-Based Lexical Web Service Generation Approach

3.1 Approach Overview

In order to cover the aforementioned problem, we proposed an approach for the generation of lexical web services. This approach is composed of three milestones with everyone of which is based on the use of a specific module. In the

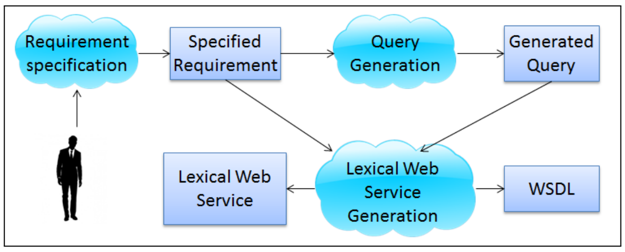

Fig. 3. Requirement-Based Lexical Web Services Generation Approach

first milestone, the NLP developer specifies his requirements in term of Inputs and Outputs (Input and Output here represent the values taken from the Data Categories Registry DCR which is a standard under the number ISO-12620 [10]) through a user-friendly interface belonging to the first module called Requirement Specification. The Input part is generally conditioned by an LMF-QL grammar compliant formula. At the end of this step, the Requirement Specification module generates an abstract form of the user's need.

In the second milestone, the Query Generation module transforms the abstract form of the requirement to a concrete form written in XQuery.

Finally, in the last milestone, the two forms (abstract and concrete) go to the third module which will generate both the lexical web service and its description in WSDL language. After the generation of the query based on the requirement provided by the user itself through the user-friendly interface, the tool also helps the generation of a web service corresponding to the generated query regardless of its complexity. The result of the web service generator tool is two outputs: the web service code and the WSDL file.

3.2 Requirement Specification

The requirement specification module helps NLP developer through a user-friendly interface to easily specify his needs in terms of lexical data from LMF database. This requirement is defined by a pair (Input, Output) which represents its abstract form. The Output represents the required result, but the Inputs represent a filter that can restrict the scope of the possible required result. For the definitions of this pair, we used data categories taken from the DCR (Data Categories Registry). To define Inputs, we removed all types of abstract symbols, particularly, those used in other languages such as XQBE [4], in order to facilitate the requirement specification. Then, we replaced these symbols by a selection of attributes used in the LMF database, which represent data categories (DC). For the definition of the query Output, we proceeded in the same manner, with the difference that for the Input we were obliged to work only with attributes (leaves), and for the Output we could work either with attributes or elements (Internal Nodes). The query Input must be precise and expressed in term of attribute-value; consequently an attribute, which is in an internal node (element which contains many entries), cannot belong to the Input of the query. More information about this step could be found here [2].

3.3 Query Generation

This module handles the abstract form of the user's requirement. The Output of the generation corresponds to an XQuery based query meeting the expressed requirements. The facility introduced by this tool lies in the fact that the Input elements make the database structure abstract and do not refer to any particular level of the database. These levels which are generally present in queries, will be determined by data categories selected in the requirements. With this intention, we proposed a high-level language called LMF-QL [2]. This language allows specifying the logical relation between different data categories selected in Input part. The LMF-QL grammar is composed of a set of terminals Σ, a set of non terminals V, an axiom S and a set of transformation rules P. Among the terminals, we used logical operators of two types: unary operator 'not' that can be applied on one data category or on an expression and which represents disjunction, and binary operators 'or'/'and' that can be applied to two data categories.

```
G= (Σ, V, S, P)
Σ= {or, and, not,(, ), $E_i$}
V = {S}
P = { S −> (S) | not(S) | (S or S) | (S and S) | $E_i$}
```

The query generation is done after the requirement specification (abstract form) by translating this form into XQuery syntax (concrete form). The generated queries may have different forms depending on the number of entries set in the input and the output of the query. The Table 1 shows the classification of queries according to the number of inputs and outputs. Each form represents a general

Table 1. Query class patterns

Number of Inputs	Number of Outputs	Result Type	Requirement Type	Number of Classes of patterns
1	0	Boolean	Existence	2
2	0	Boolean	Consistency	4
$i \geq 1$	$j \geq 1$	Different types	General Requirement	2^{i+j}

model for a class of requirements as shown in table I. The first class is to express the requirement of the existence of a category of data. It is divided into two patterns because it has two different query models. The second class is used to test the consistency between the two categories or the other data. For example, we can test the consistency between the prefix and the suffix. The third class represents the general case. The number of requests generated depends on both the number of entries and that of outputs. In general, the number of classes verifies equation (1):

$$Number of classes = 2^{(Number of Inputs + Number of Outputs)} \tag{1}$$

3.4 Lexical Web Service Generation Tool

Like the automation of the query generation, the service generation represents an efficient step to ease the interrogation of LMF databases. Besides, having a standard interface, like web services, increases the interoperability aspect of the lexical resources. In this context, we developed a specific tool based mostly on the query generated by the Query Generator Module (see fig.4). The query is firstly

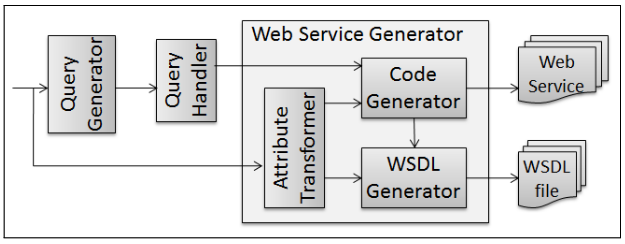

Fig. 4. Lexical Web Service Generation Tool

sent to the Query Handler Module in order to check its consistency. It's a kind of query compiler that verifies the syntax of the query before execution, which can help avoid exceptions due to syntax when calling the web service. The query can then be transferred to the Code Generator Module to start manufacturing the web service. The last cited module also needs some inputs that will be gathered from Attribute Transformer module. These inputs give information about the user's needs required in the service code generation. The Attribute Transformer is responsible for gathering information and transforming it to the suitable forms in order to make it available for both modules "Code Generator" and "WSDL Generator".

Attribute Transformer As already mentioned, the user's requirements are involved in all the steps of lexical service generation process. The main role of the Attribute Transformer Module is to gather the user's requirement information in its abstract form in order to transform it to suitable variables that will be used later by both modules "Code Generator" and "WSDL Generator". The information about the user's requirements is the inputs and output parameters like their Data Categories, variables names, entry options, etc. This information could also be a simple text that the user uses to describe query capabilities. The Attribute Transformer Module catches this information and structures it by creating dedicated variables with suitable types. The goal is to make it ready for use by the next modules.

LWS Code Generation This module will build the web service around a service skeleton that contains 4 main parts:

– Connection to the LMF database.
– Query parameter binding operation
– Execution of the query

– Preparation of the result of the query

a) The connection to the LMF database: This part of the service code allows the connection to the lexical database. Actually, there is only one database we are working with. Besides, the database network address and the connection and configuration parameters are hardcoded. However, this information could be easily provided by a user as input parameters.

b) Query parameters binding operation: The code generated in this part serves the preparation of the query before being executed. In fact, each query contains input and output parameters. The values of these parameters have to be prepared before being injected in the query for execution. Two cases are supported (both cases can be combined):

– The first case where the input value is already expressed by the user while providing the requirements (handwritten or chosen among predefined values list of the corresponding data category). In this case, the input value is hardcoded in the query. No input value injection will be done.
– This the case where the user doesn't provide values to his inputs but rather chooses to provide them as external values (as variables). Only the data category of each input is specified. In this case, the inputs are considered as arguments in the web service method. These arguments have to be bound respectively to the variables already declared in the query. Thereby, the inputs values will be correctly transmitted to the query when invoking the web service. This case enables the reusability of a configurable lexical query.

In this last case, the Code Generator Module requires the user's requirements, precisely information about the inputs and output parameter (data category, variable name, value entry option, etc), to generate the binding operation code. This information is caught and structured by the Attribute Transformer Module to make it available for other modules (like WSDL Generator explained later).

c) Execution of the query: Once prepared, the query has to be executed. This part of the service skeleton enables the generation of the service code responsible for the execution of the query. The execution is based on the DataDirectXQuery implementing the XQJ (API XQuery for Java).

d) Preparation of the result of the query: The execution of the query generates an output corresponding to the user's requirement. This part of the service code retrieve the result of the query execution and transforms it to the suitable format specified by the user when generating the query. The possible formats can be a simple string, XML section, Array, Object, etc. The generated service code is obtained as Java class source code. We used Apache Axis native JWS (Java Web Service) to expose and deploy our code as a web service. The generated bundle contains a deployable web service with all required resources ready for use in web application. However, our web service generator tool helps modify the WSDL file automatically generated by Apache Axis Framework. Indeed, we propose to enrich the basic WSDL file with lexical information that helps index and reference lexical web service in the lexical service directory UDDI.

WSDL File generation The last remaining step is to generate the WSDL file corresponding to the generated web service. Once again, the user's requirements are involved. Indeed, the module WSDL Generator enriches the basic Axis generated the WSDL file with specific annotations that help NLP application developers to find out the most appropriate web services. The content of these annotations is provided by the Attribute Transformer Module. It contains a human readable description of the generated query provided by the NLP developer when specifying his requirement. Besides, it also contains the definition of the query input and output parameters representing the lexical data categories. There are two kinds of annotations enriching the WSDL file:

a) Annotation describing the query capability: The content of this annotation is a human readable text provided by the user when specifying his requirements. It describes the functionality of the generated query. In fact, there is only one annotation of this type per generated WSDL file. This annotation is added as a predefined XML element called <wsdl:documentation> under the WSDL element <wsdl:operation> under the WSDL element <wsdl:portType> representing the method of the service. This is an example:

```
<wsdl:portType name="Affixes_Service">
   <wsdl:operation name="Prefixes_List">
   <wsdl:documentation>This method gives the list of prefixes</wsdl:documentation>
```

b) Annotation describing the input and output parameters: This annotation define each input and output of the query. Actually, there are annotations as many as inputs and outputs. This kind of annotation is added using the predefined XML element <wsdl:documentation> under the element <wsdl:part> under the WSDL element <wsdl:message>. The text added in the XML tag <wsdl:documentation> is the definition of the data category of the corresponding input or output.

```
<wsdl:message name="SyntacticFunctionRequest">
   <wsdl:part name="syntacticHead" type="xsd:string">
     <wsdl:documentation>syntacticHead: central element of a subcategorization
           frame</wsdl:documentation>
   </wsdl:part>
   <wsdl:part name="VoiceProperty" type="xsd:string">
     <wsdl:documentation>VoiceProperty: is the class of properties that concern the
           grammatical encoding of the relationship between the verb and the nominals in
           a subject−predicate configuration.</wsdl:documentation>
     </wsdl:part>
</wsdl:message>
```

3.5 Synthesis

The automatic generation of lexical web services based on the NLP requirements helps overcome the lack of new specific web service in an initial service library of SoLDES system. Hence, SoLDES became a complete service-oriented architecture covering even new specific lexical requirements. NLP developers no longer fear the complexity of LMF database query with such a tool. In order to assess and to show the usability of SoLDES system, we will give, in the next section, some illustrations of the use of our system and examples of the generation of some lexical services starting from the expression of specific requirements.

4 Illustration: Lexical Services for Arabic Language

In this section, we present a set of lexical service requirements identified in the context of Arabic language although most of these requirements could be the same for other languages. Then we present an example of implementation of some services. We will show later our service library. We finish this section by giving two case studies that give an example of how to integrate these services in the context of an NLP application.

4.1 Characteristics of the Arabic Language.

The recent years have seen a considerable progress in the field of NLP. The Arabic language does not make exception but it has been much less studied from the data-processing point of view than English or French. This may be due to difficulties related to this language. Indeed, by its morphological and syntactic properties, the Arabic language is considered a difficult language to master in the NLP field [3]. Among these properties we can quote:

– The letters change form of presentation according to their position in the word which is written from the right to the left.
– An Arabic word is written with consonants and vowels. The vowels are added above or below letters (بَ, بُ, بِ, بْ). They facilitate the reading and the correct understanding of a text to differentiate words having the same representation.
– In Arabic, a word can mean a whole sentence thanks to its mixed structure which is an agglutination of elements of grammar.
– Arabic is an inflected language. Indeed, the terminations make it possible to distinguish the mode from the verbs and the function of the names.

4.2 Requirement Identification

The lingware development complexity is due to the multitude of needs that may present a linguistic application in terms of information, in particular, of lexical nature. In our experimentation, we studied a set of applications dealing with the morphology of Arabic in order to identify their lexical needs. Hereafter, we give the results of this study for some of these applications namely: Arabic Text Tagger, Morphological analyzer, spellchecker, education through interactive software.

Arabic Text Tagging This application can be defined as the set of operations that can switch between plain text, free of linguistic information, and a sequence of elementary lexical units (lemmas) accompanied by morphosyntactic labels. This definition implies successively the choice of the basic unit of segmentation, the process of segmentation itself, the lemmatization of the units and the association of linguistic information to the lemmas. The phase of lemmatisation requires the exploitation of the Lexical DataBase in order to join to a lemma all morphological features that correspond to it. These features can change according to the lemma [17].

Table 2. Text Tagging

Requirement	Input	Output
Part-of-Speech Tagging	Verb	Conjugation Mode , GramPerson, Scheme
	Deverbal	Type, Chained Verb, GramGender, GramNumber
	Noun	human characteristics Human/Non-Human, Proper Noun/Common Noun, GramGender, GramNumber
	Function Word	GramGender, GramNumber

Morphological Analysis and Synthesis This is a program that can recognize a word in the various forms it can take in sentences. For each found form, the elements must be isolated and morphological features deduced out of the context associated with them. The morphological processing of Arabic must cover both generation processes (or synthesis) and analytical (or recognition). The synthesis process must allow the generation of a word starting from a root, a scheme and a set of morphological specifications. Based on [9], we were able to identify all the needs for these two processes. Table 3 shows these needs.

SpellChecker This kind of application deals with the errors related to the lexical level. Error handling does not take into account the context of the word to check. Lexical errors are those related to membership of the words to the language. The need for the LDB within the framework of this system can be summarized in the Table 4.

Interactive Teaching The teaching of Arabic through interactive software must be able to intervene both in recognition (which corresponds to certain aspects of the difficulties encountered when reading or looking for a word in a

Table 3. Morphological Analysis and Synthesis

Requirement	Input	Output
Morphological Analysis	Base	Boolean (Exist or not)
	P, S et I	Boolean(Exist or not)
	P, S et I	Boolean(Consistent or not)
	Word	Root, chained word
Morphological Synthesis	Root	Boolean(Exist or Not)
	P, S, I et Root	Boolean (Consistent or not)
	Root	List of Associated Base
	Root	Exist or Not

Table 4. SpellChecker

Requirement	Input	Output
Orthography	Word	Exist or Not
	Selection	List of Prefixes
	Selection	List of Suffixes
	Selection	List of Infixes
	Selection	List of Roots
	Prefix, Suffix	Boolean (Consistent or Not)
	Prefix, Suffix et Infix	Boolean (Consistent or Not)

dictionary or BDL) and in producing (as regards the construction problems of Arabic words at the time of expression). Both approaches will need two processes of analysis and synthesis which have been mentioned in Table 6. Thus, we could address in the Table 5 the interactive teaching needs.

Table 5. Interactive Teaching

Requirement	Input	Output
interactive teaching	Verb	Grammatical Features
	Non Diacritical Word	List of all Diacritical Forms
	Diacritical Word	PartOfSpeech, (Prefix, Infix, Suffix) list of chained roots
	Root	All associated forms

Synthesis All the previously cited lexical needs were translated into XQuery queries and its corresponding lexical web services that were generated using

Table 6. conjugation synthesis and analysis

Requirement	Input	Output
synthesis Conjugation	Verb	Conjugated forms in all modes
	Verb + conjugation mode	Conjugated forms in this modes
	Verb + conjugation mode + Person	Conjugated Form
Analysis Conjugation	Conjugated Form	Person, Gender and Number
	Conjugated Form	Related forms
	Accomplished Form	Unaccomplished Form

our SoLDES system. All these generated services could be then used in the development of NLP applications.

4.3 Arabic Text Spell Checker (ATSC): First Case Study

To experience our approach, we proceeded to the redevelopment of the system [8] “Arabic Text Spell Checker”. The choice of this system is justified by the richness of its lexical resource requirements and also the availability of its code. ATSC uses a lexicon that is formed by the following files: a file for roots, a file for the prefixes, a file for the suffixes, a file for the infix, a file containing the consistency matrix between prefixes and suffixes, and a file containing the consistency matrix between prefixes, suffixes and infixes. During the redevelopment of ATSC, we focused on the file access. Indeed, we proceeded in two stages:

– locating the access levels to files in the code of the application and determine the needs that involve from this access.
– Searching from our lexical web service library services that can meet these needs. A new service has been developed for every unmet need.

Second, we tried to see if some ATSC features could be provided by one or more of our services. Thus, we have located the “decomposition” service which allows for a correct word to give its components in terms of root, infix, prefix and suffix. As experimental results, we note that after using our SoLDES system, we have reduced in the ATSC new version the number of used java classes from 22 classes in the original version to 7 classes in the new service-oriented version. Furthermore, the use of SoLDES system for generating lexical web services has led to the reduction of the number of lines of code. This is due to the replacement of the methods of existence check of the suffixes, prefixes, infixes and root by the simple service invocation from our Lexical Service Directory (LSD). The methods of verification in the original version are very complex with the instructions of opening and reading files, the iterative loops and multiple conditional blocks. All this is replaced by the simple method binding() for the service invocation which considerably reduced the effort and the time of the development.

4.4 Automatic Summarizer for Arabic: Second Case Study

For the evaluation of our work, we will present another case study that integrates some of the lexical services from our library during the process of summarizing an Arabic text that we first present. The production process of an automatic summary is based on five modules namely the segmentation module which is the first step that enables to segment the source text into sentences[1]. The result is sent to another module called the morphological analysis module to give the morphological features of each word of the text. The syntactic analysis module receives the output of the morphological analysis module and shows the syntactic structure of each sentence in the summary. The extraction module for the relevant sentence extracts the most important phrases. Finally, the revision module helps obtain a refined extract by eliminating the redundant phrases. In this process, we will focus on modules that can offer low-level linguistic information, namely morphological and syntactic analysis.

In this context, among the requirements, a developer of an automatic summarizer for Arabic concretely requires two specific low-level lexical services that we call “Requirement 1” and “Requirement 2”.

Requirement 1 Here, the developer needs to look for the morphological features of a given lexical entry. He uses our tool to precise the different parameters of his requirement. In this case, the input is a lexical entry having a “Written Form” as Data Category. The developer has to choose to provide this input as an external value. In fact, no value will be entered at this level. The value of the lexical entry will be provided later while using the generated service. The output which the developer wait for is a “WordForm” element supporting several kind of Data Categories like “Grammatical Number”, “Grammatical Person” and “VerbFormAspect”. The developer chooses the XML section as output format and integrate it in the whole summarizer system. The generated query is shown here:

```
declare variable $x as document−node(element(*,xdt:untyped)) external;
declare variable $u1 as xs:string external;
declare variable $u2 as xs:string external;
for $a0 in
$x//LexicalResource/Lexicon/LexicalEntry/WordForm where $a0/DC/@att=”writtenForm”
      and $a0/DC/@val=”$u1”
return for $a1 in $a0 where $a1/DC/@att=”voice” and $a1/DC/@val=”$u2”
return <result>{$a0}</result>
```

Once the query is generated, the developer generates the corresponding Web Service to be integrated in the whole system. We present here the result of the invocation of the generated service with the values of data categories (the variable $u1 will take the value “كَانَ” and $u2 will take the value “activeVoice”.

```
<?xml version="1.0" encoding="UTF−8"?>
<result>
  <WordForm>
      <DC att="writtenForm" val="كَانَ"/>
      <DC att="grammaticalNumber" val="singular"/>
      <DC att="grammaticalGender" val="masculine"/>
      <DC att="verbFormAspect" val="accomplished"/>
      <DC att="voice" val="activeVoice"/>
      <DC att="person" val="thirdPerson"/>
   </WordForm>
</result>
```

Requirement 2 In this case, the developer requires to know the syntactic function of a lexical entry having the voice property value equal to “Active Voice” and having a given syntactic behavior. Actually two inputs are given here:

– Input 1: represents the syntactic behavior having “Type” as data category and will be provided as external value.
– Input 2: having the “Voice Property” as data category and its value is “activeVoice”. It will be a hardcoded value.
– Output: having the data category “SyntacticFunction”.

Hereafter, we present the query corresponding to this requirement.

```
declare variable $x as document−node(element(*,xdt:untyped)) external;
declare variable $u1 as xs:string external;
for $a0 in
$x//LexicalResource/Lexicon/SubcategorizationFrame where $a0/DC/@att="type" and $a0/
       DC/@val="$u1"
return for $a1 in $a0/LexemeProperty where $a1/DC/@att="voice" and $a1/DC/@val="
       passiveVoice"
return
for $a2 in $a1/../SyntacticArgument/DC where $a2/@att="syntacticFunction"
 return <syntacticFunction>{$a2/@val}</syntacticFunction>
```

The result of this query will be as follows after invoking the service with the value of $u1=”فِعلٌ متعَدِي لِمَفعولَيْنِ أَحَدُهُمَا بِأَدَاةٍ”:

```
<?xml version="1.0" encoding="UTF−8"?>
<syntacticFunction val="نَائِبُ الفَاعِلِ"/>
<syntacticFunction val="مَفْعُولٌ بِهِ"/>
```

5 Related Work and Discussion

The main objective of our current research is to promote the use of lexical resources through enhancing the interoperability between involved actors, namely, end users applications, NLP applications and lexical databases. The first solution is the use of normalization (standard) of principles and methods related to the lexical resources in the context of multilingual communication and cultural diversity. This is done through the adoption of LMF standard for lexical databases. This can bring NLP application developers to less dependency on a specific lexical database structure, but this is not enough. In fact, a service oriented approach effortlessly allowing the access to lexical standard resources represents a second step solution we are adopting. This solution was also adopted by several scientific communities having looked to facilitate the extraction of lexical resources according to the needs of users or NLP applications. The proposed work in [5] is based on the use of web services for the use of NLP features in a multicultural context. However, the proposed services don't tackle the access to the lexical resource .Besides, the proposed work does not address the Arabic language and its specificities since it doesn't adopt LMF standard. Finally, the generated web service is specific to their platform while our work generates a web service according to the need of the user and can be used by any NLP application. In the same context, researchers in [16] present an architecture to connect customers to NLP frameworks through the use of web services. However, NLP subsystems do not support standard database and the proposed approach does not display solution for the needs of the customer's alignment with the standard structure. Other works propose the use of a web service but are limited to a specific usage. We mention the work [15] which deals with the development of a RESTful Web service to access WORDNET-type semantic lexicons. It generates LMF compliant XML data. We can also mention the web service tool for automatic extraction of MWE (Multi-Word Expression) lexicons [13]. The usage here is limited to the creation of Lexical Resources.

6 Conclusion and Future Work

The research carried out by the ISO around LMF project showed the interest granted by all the community of NLP researchers to the lexical component in this field. Starting from this, our work takes the continuation of the representation aspect to cover the LMF lexical database exploitation. In addition, on the basis of various needs in lexical resources and technical difficulties facing users (i.e., NLP applications developers), we proposed to use a Requirement-based approach for the generation of lexical web services to ensure easier exploitation of lexical resources and minimize the developers' efforts. Indeed, this approach discharges the developers from several tasks: knowledge of the database structure, master of a query language, etc. Moreover, this approach gives the possibility of serving different users in different corners of the world starting from a single database. On another side, the interrogation of the lexical database remains transparent for the users. Indeed, we worked with parametrized queries generating according to the listed needs. The query list remains extensible to cover new needs. The results of our work are being tested for many linguistic levels in the framework of Arabic spell checker application and during a summarizing process. The lexical database (ElMadar Dictionary)[4] used in our work is LMF compliant. Currently, we are planning to test our approach on various lexical databases in conformity with LMF, possibly for other languages. In the near future, we will work on the refinement of the description generated with the lexical service (WSDL description file) in order to facilitate the discovery of our services. Subsequently, we will study the integration with applications and composition scenarios that could give more help to assist NLP developers in the development of new lingwares.

References

[1] Abdallah, M.B.: Réalisation d'une plateforme de résumé automatique de textes arabe. Master's thesis, Ecole Nationale d'Ingénieurs de Sfax (2008)

[2] Abderrahmen, M.B., Gargouri, B., Jmaiel, M.: LMF-QL: A graphical tool to query LMF databases for NLP and editorial use. In: Vetulani, Z., Uszkoreit, H. (eds.) Human Language Technology. Challenges of the Information Society, Third Language and Technology Conference, LTC 2007, Poznan, Poland, October 5-7, 2007, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5603, pp. 279–290. Springer (2007), http://dx.doi.org/10.1007/978-3-642-04235-5_24

[3] Aljlayl, M., Frieder, O.: On arabic search: Improving the retrieval effectiveness via a light stemming approach. In: Proceedings of the Eleventh International Conference on Information and Knowledge Management. pp. 340–347. CIKM '02, ACM, New York, NY, USA (2002), http://doi.acm.org/10.1145/584792.584848

[4] Braga, D., Campi, A., Ceri, S.: Xqbe (xquery by example): A visual interface to the standard xml query language. ACM Trans. Database Syst. 30(2), 398–443 (Jun 2005), http://doi.acm.org/10.1145/1071610.1071613

[5] Bramantoro, A., Tanaka, M., Murakami, Y., Schafer, U., Ishida, T.: A hybrid integrated architecture for language service composition. In: Web Services, 2008. ICWS '08. IEEE International Conference on. pp. 345–352 (Sept 2008)

[6] Francopoulo, G.: Proposition de norme des Lexiques pour le traitement automatique du langage. http://pauillac.inria.fr/atoll/RNIL/TC37SC4-docs/N07.pdf (2003)

[7] Francopoulo, G., George, M.: Model Description, pp. 19–40. John Wiley & Sons, Inc. (2013), http://dx.doi.org/10.1002/9781118712696.ch2

[8] Hamadou, A.B.: Vérification et correction automatiques par analyse affixale des textes écrits en langage naturel : le cas de l'arabe non voyellé. Ph.D. thesis, Faculté des Sciences de Tunis (1993)

[9] Hassoun, M.: Conception d'un dictionnaire pour le traitement automatique de l'arabe dans différents contextes d'application. Ph.D. thesis, Université Claude Bernard-Lyon I (1987)

[4] elmadar.miracl-apps.com

[10] ISO 12620:2009: Terminology and other language and content resources – Specification of data categories and management of a Data Category Registry for language resources, vol. 2009. ISO, Geneva, Switzerland (Dec 2009)

[11] Khemakhem, A., Gargouri, B., Haddar, K., Ben Hamadou, A.: LMF for Arabic, pp. 83–98. John Wiley & Sons, Inc. (2013), http://dx.doi.org/10.1002/9781118712696.ch6

[12] Khemakhem, A., Gargouri, B., Hamadou, A.B.: LMF standardized dictionary for Arabic language. In: Proceedings of the 1st International Conference on Computing and Information Technology (ICCIT 2012). pp. 522–527 (2012)

[13] Quochi, V., Frontini, F., Rubino, F.: A MWE acquisition and lexicon builder web service. In: Kay, M., Boitet, C. (eds.) COLING 2012, 24th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, 8-15 December 2012, Mumbai, India. pp. 2291–2306. Indian Institute of Technology Bombay (2012), http://aclweb.org/anthology/C/C12/C12-1140.pdf

[14] Romary, L., Salmon-Alt, S., Francopoulo, G.: Standards going concrete: from LMF to Morphalou. In: The 20th International Conference on Computational Linguistics - COLING 2004. coling, Genève/Switzerland (2004), https://hal.inria.fr/inria-00121489

[15] Savas, B., Hayashi, Y., Monachini, M., Soria, C., Calzolari, N.: An lmf-based web service for accessing wordnet-type semantic lexicons. In: Chair), N.C.C., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., Rosner, M., Tapias, D. (eds.) Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10). European Language Resources Association (ELRA), Valletta, Malta (may 2010)

[16] Witte, R., Gitzinger, T.: A General Architecture for Connecting NLP Frameworks and Desktop Clients using Web Services. In: Kapetanios, E., Sugumaran, V., Spiliopoulou, M. (eds.) 13th International Conference on Applications of Natural Language to Information Systems (NLDB 2008). LNCS, vol. 5039, pp. 317–322. Springer, London, UK (June 24–27 2008)

[17] Zaafrani, R.: Développement d'un environnement interactif d'apprentissage avec ordinateur de l'arabe langue étrangère. Ph.D. thesis, ENSSIB/Université Lumière Lyon 2 (2002)