

Using Weakly Structured Documents to Fill in a Classical Database

F. LaFOREST, Information Systems Laboratory, France
A FLORY, Information Systems Laboratory, France

Electronic documents have become a universal way of communication due to Web expansion. But using structured information stored in databases is still essential for data coherence management, querying facilities... We thus face a classical problem—known as “impedance mismatch” in the database world—: two antagonist approaches have to collaborate. Using documents at the end-user interface level provides simplicity and flexibility. But it is possible to take documents as data sources only if helped by a human being: automatic documents analysis systems have a significant error rate. Databases are an alternative as semantics and format of information are strict: queries via SQL provide 100% correct responses. The aim of this work is to provide a system that associates document capture freedom with database storage structure.

The system we propose does not intend to be universal. It can be used in specific cases where people usually work with technical documents dedicated to a particular domain. Our examples concern medicine and more explicitly medical records. Computerization has very often been rejected by physicians because it necessitates too much standardization and form-based user interfaces are not adapted to their daily practice. In this domain, we think that this study provides a viable alternative approach. This system offers freedom to doctors: they would fill in documents with the information they want to store, in a convenient order and in a more free way. We have developed a system that allows to fill in a database quasi automatically from documents paragraphs.

The database used is an already existing database, that can be queried in a classical way for statistical studies or epidemiological purposes. In this system, the document fund and the database containing extractions from documents coexist. Queries are sent to the database, answers include data from the database and references to source documents.

Introduction

Information capture is an important barrier for end-users software acceptance. In domains where the end-user is not compelled to use a computer or is demanding because of activity constraints, classical computerized systems have difficulty being accepted and widely used. Difficulties are more accurate when documents are the paradigm used to manipulate information. One can find many domains of this type: lawyers, doctors... use technical documents to store the information they need. These are domains where computerization is particularly little used. The example of the medical domain is obvious. Doctors are not satisfied by today's systems and prefer using paper-based medical records. Many trials have been -and are still- conducted in this field, but success has not completely come. The main barrier concerns information capture speed and facilities compared to computerized systems advantages. Capture forms have been chosen by computer scientists because they have the great advantage to provide important querying capacities, as they are most often easily related to a database. Capture forms do not satisfy

physicians as they cannot adapt to each case encountered. Forms impose data to be captured, the order in which to capture and a strict format for each data.

With the current prevalence of the Web and consequently of electronic documents, the next idea that comes is to use electronic documents as a basis for the system. This idea has the first advantage to remove the mismatch between paper-based documents and capture forms. Information is captured in electronic documents and queries on documents can be made using a dedicated document querying language. To go forward in this idea, we have to notice that one can find 3 different types of documents :

- Free text documents only contain information and formatting instructions (these instructions are also known as physical structure). These documents are very easy to write, but very difficult to query. Free text analysis is still a research domain, the results are not yet satisfying enough to be used in sensitive domains. Compared to paper-based documents, systems based on free text documents still do not provide enough precision to be widely used.

- Strongly structured documents contain information and semantics guides (also known as logical structure). These guides are most of the time represented by tags circling information pieces. SGML ([ISO, 1986]) documents are good examples of such documents. These documents set a structure that the user has to follow. This structure is defined in a Document Type Definition (DTD) which provides tags, composition rules of tags, compulsory tags, attributes for tags... This structure is not as rigorous as forms structure: no format is imposed for data. Moreover, only effectively captured information appears in each document. In forms all fields are present even if not filled for the current case. Queries on strongly structured documents can be made using dedicated query languages like SgmlQL [Lemaitre, 1998] or UnQL [Buneman, 1996]. These languages are currently under the form of prototypes and answers lack precision.

In systems that link strongly structured documents to a database, each information to be stored in the database is tightly tagged so that there is a one-to-one relationship between data in the database and tagged information in documents. The database stores a representation of the tree structure of the document, without any treatment on information pieces: filling a database is thus rather easy but does not provide the same facilities as a real information system database filled through a form: queries are not as precise as queries on atomic data. This approach still does not satisfy end users, as writing such documents is time consuming and too much constraining. As each stored information piece has to be tightly tagged, the DTD looks much like a disguised form.

- Weakly structured documents may be seen as an intermediate level between free text documents and strongly structured documents. Weakly structured documents use DTDs containing optional tags (compulsory tags are defined only to contain the identification of the document subject e.g. patient and doctor IDs). Most tags delimitate paragraphs rather than data. A paragraph contains free text which may include many data. For example, a prescription paragraph contains one sentence in which one can find a medication name, a dose, a frequency and a duration. That is the type of documents our system manages. To query data stored in such documents, we link the DTD to a relational database. In each paragraph one can find information which should belong or not to the database. These documents have a double advantage: (1) to be easily captured and read by the end-user and (2) to be semantically more precise than free text, thus easier to analyze automatically.

In the following of this article, we use the medical record for illustrations. Here is a rapid tour of our system. The end-user captures information in a weakly structured document. He tags paragraphs according to a DTD conforming to his application domain. For example, tags can be <prescrip-

tion>, <past history>, <diagnosis>... Once a document is validated by the end-user, an analyzer extracts data from the tagged paragraphs, builds SQL queries and sends its results to a database. At the opposite of usual document databases, the database contains a classical information system database (tables like "patient" and "prescription", no "paragraph" nor "keyword" tables). For each tagged paragraph, the analyzer can extract many data to be stored in many tables. Relationships between paragraphs and the database are stored under the form of patterns to be recognized and links to the database. These patterns are defined a priori by the application designer at the same time as the DTD.

The medical record domain seems a good candidate for this system for many reasons. The first is that medicine is still an art that cannot be strongly structured, so that no form-based user interfaces are adequate. Documents are used in the daily work. For chronic diseases, records rapidly become thick: searching information looks like mining. Doctors are interested in automated systems that could help in information retrieval for the care of one patient or for statistical purposes. They also appreciate security improvement (prescriptions validation...). The second reason is that the language used by physicians has already been classified. There exists many classifications, ontologies and thesauri referencing the medical language. This is an advantage for the analyzer, that has referentials to identify data. Classifications of diseases are written for years, medications thesauri can be found in any country, acts are listed in social security files... The third reason is that many paragraphs have standard ways of writing and patterns can be defined for a quite large use. For example, the prescription paragraph should include a medication name (from a thesaurus), a duration (always followed by a duration unit and very often preceded by a word like "during"), a frequency (always followed by a frequency unit) and so on... Each sub-part of the prescription sentence can be represented by a few patterns to search.

This article presents in a deeper way the system we defined. Next section gives position of our work compared to the literature. Next, a system overview presents the differences between a classical documents database and our system and then presents answers to the main questions one can think of. In section 4, we give the architecture of our system using the UML formalism. Section 5 provides a description of the first prototype we have written and an analysis example based on a medical record. We then conclude with the current progress state of this project.

Related works

Much has been done on documents in the literature. Querying free-text documents is a difficult task that has been studied for years. Recall and precision are difficult to ensure and balance [Salton, 1986]. Research activities in this domain are still numerous. A lot of studies are dedicated to a specific application domain, like medicine [Blanquet, 1999], law

[Jackson, 1998], or automobile ads [Embley, 1998a]. Passage retrieval [Kaszkiel, 1997], excerpts and resume construction [Goldstein, 1999], ontology-based indexing [Stairmand, 1997] are some of the directions currently studied to improve free-text mastering.

Studies on structured documents rapidly grew with the arrival of SGML [ISO, 1986]. SGML is well adapted to documentation structuring and is widely used in technical documentation. DTDs propose many mechanisms to define structuring rules for documents. Systems to query SGML documents are both using words, as in free-text, but also the document structure seen as a graph. Some languages use pattern-based search algorithms [Laforest, 1999], [Lakshmanan, 1996], [Baeza, 1996] based on tree representations of documents. Some propose a SQL-like user interface [Neven, 2000], [Kilpelainen, 1993]. Many structured documents querying languages use this principle [Lemaitre, 1998], [Goldman, 1997], [Buneman, 1996], [Deutsch, 1999a], [Mendelzon, 1997], [Konopnicki, 1998], [McHugh, 1997], [Abiteboul, 1997a], [Abiteboul, 1997b] for querying documents written in any structuring language.

Another way to query structured documents is to store information pieces into a database. Some have adapted HTML by adding specific database tags [Dobson, 1995]. Others have adapted the database to documents [Christophides, 1994], [Stonebraker, 1983], [Cluet, 1999], [Cohen, 1998], [Bohm, 1997], [Atzeni, 1997], [Deutsch, 1999b], [Abiteboul, 2000]. The database contains tables like “paragraph”, “section”... that do not provide any semantics to the information stored. A variant of these approaches proposed in [Frenot, 1999] uses XML semantic tags adapted to an application domain so that queries are contextually oriented by tags semantics. Storing documents in such a database does not allow to take advantage of SQL. In these approaches, columns of the database do not contain formatted data but text pieces. Queries cannot thus be made on a standard way, exact matching and joins have to be re-defined and are difficult to implement. These systems also index text pieces. Using indexes follows the same idea as using indexing databases on file-stored documents.

Structured maps [Delcambre, 1997] propose an additional modeling construct that provides structured and managed access to data. They are based on Topic Navigation Maps [Biezunski, 1995]. The main idea is to provide a database that is manually filled to index information pieces in documents. The database is not generic but domain related (e.g. “painter” and “painting” tables). It allows classical querying on the database. Each data in the database is related to the source information piece so that the database answers provide the linked text pieces. The system also provides topic navigation maps, i.e. documents that refer to all documents concerning a topic.

In [Riahi, 1998] and [Embley, 1998a], one can find an approach which is closer to ours, even if dedicated to HTML

documents. The main idea is to build a real application domain database associated to an important “classical” indexing feature, so that queries are sent to the database rather than processed on documents. In [Riahi, 1998] the database is filled using the very few semantic tags offered by HTML (keyword, author) and mainly using formatting information: it uses the geographical location of tags to “guess” data semantics. Rules to extract information from documents have to be written for each type of document, i.e. for each site. Moreover, each site has to ensure presentation coherence rules for all documents. The problem faced by this project is mainly due to HTML and its lack of semantics. The model of the database is highly dependent on documents contents and it seems very difficult to define one general model that would satisfy an application domain treated by many different independent Web sites. [Embley, 1998a] uses an ontology to describe each data with all its essential properties. The database to be filled is automatically created according to the ontology structure. A main concept in the database represents the main subject of the data (e.g. one person in an obituary). Documents are separated into different records, each record concerning one main subject instance. Data properties mainly include PERL5-based rules patterns to extract data from HTML text, as well as relationships between the main subject of a record and all its data. This project is interesting for the capture of data into databases that contain few tables, in other words in domain that are “narrow in ontological breadth” [Embley, 1998b].

As a conclusion, we can say that documents are more and more used in electronic systems. But systems that include documents management do not see them as a new way to capture data. Documents contain information for the end-user. Systems offer information retrieval techniques to search for documents, but do not provide complete data management. The type of documents implies different indexing techniques, but queries made by end-users are always of the type “give me documents that ...”. Our proposal goes further: we use documents as data sources, so that we can fill in a database. Data in the database can be searched for themselves, independently of source documents. Queries made by end-users can be of the form “select data from database Tables where...”.

We have found very few proposals dealing with this issue. Some use manual database filling. Others are based on HTML documents and face much difficulties as HTML is not semantically tagged. The last ones use XML strongly structured documents, that are dedicated to system-to-system data interchange but cannot be written by end-users. We propose to use XML weakly structured documents and to take profit of an application domain where the language is already classified in thesauri, databanks or classifications.

Overview of the DRUID system

The Documents- and Rules-based User Interface for Databases (DRUID) system intends to use weakly structured documents as a user interface paradigm. Internal rules are dedicated to extract information from these documents and thus to fill in a database. Documents are seen as a new way to capture data, and not as an additional system parallel to the classical information system.

Classical Documents Databases vs. Our System

Classical systems that associate a database and a documentary fund are not dedicated to manage data but to manage the documentary fund. Their aim is to provide rapid access to documents and tools to select documents. They do not intend to provide an application domain model nor to use documents as a way to fill in an application domain model. They work as follows (see fig.1):

- documents are written in a non structured, strongly structured or weakly structured manner, in accordance with the rules of the enterprise. The document file is stored in the documentary fund (often the file system of a server).
- the author or a librarian enters indexing information concerning the new document in the database using capture forms. These forms ask for contextual information on the document: location, author references, date, version number and key-words are the most usual data captured. The database contains this indexing information. It does not contain the domain model. We call it an indexing database.
- searching for a document is made by querying the indexing database. This query does not concern the document content but captured contextual information. The query returns the list of documents corresponding to it.
- some systems add to this manual referencing an automatic system. This indexing can be based on the frequency appearance of words in the document. In HTML, the

META section allows the document writer to define key-words on his document. Indexing systems on the Web use such techniques.

These systems do not consider documents as data sources for the computerized system, but as information sources for the end-user. The database is an indexing database. It permits the end-user to access the documents that contain information. But it does not allow a direct access to application domain information. The indexing database can be seen as a dictionary containing metadata on a documents fund. It contains tables and attributes like "document name", "document id", "author", "authoring date", "section title"... that are comparable to the dictionary of databases that contain tables and attributes like "table name", "table key", "author", "authoring date"... These elements do not depend on an application domain. The database is orthogonal to application domains and can be multi-domain.

The system we propose is totally different. It uses :

- a classical information system database (the model represents data of the application domain),
- weakly structured documents instead of classical forms to capture information,
- an analyzer to fill in the database through an automated analysis of the captured documents,
- queries can concern the database itself as it is classically done, or can ask for source documents.

Our system has to drop the fastidious manual step of database update through a capture form, by automating this step using an automatic analyzer (see fig.2). The database contains both contextual information and semantically extracted information. The extraction of information does not concern all information in the document, but only information that have a correspondence in the database. We propose to capture weakly structured documents, so that documents are quite easy to capture and the extraction mechanism is guided by semantic tags.

Figure 1 : Classical systems associating a documentary fund and an indexing database

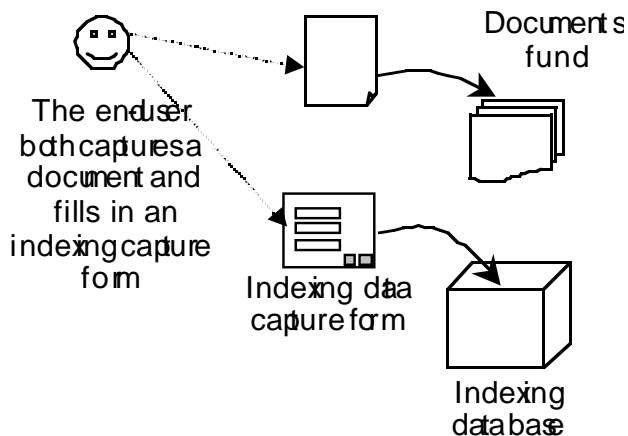
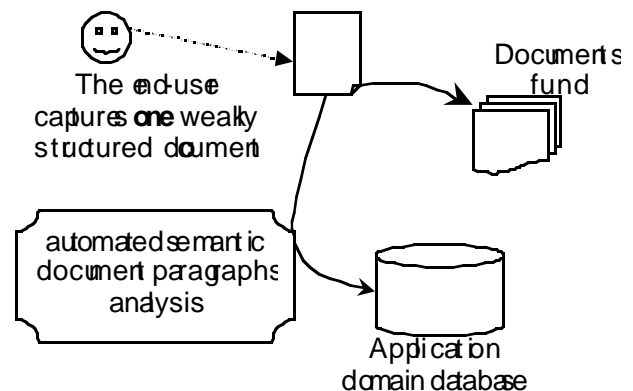


Figure 2 : Our proposal based on automated database update from weakly structured documents



The following paragraphs give an overview of the principles of the DRUID system. They are detailed in the next section throughout the meta-model and the software architecture used.

How Documents Are Built Within DRUID

We have chosen the XML [W3C, 1998b] language to build weakly structured documents. With this language, the user builds documents by including free pieces of text quoted by tags. A DTD provides the definition of tags allowed and of their composition relationships. Tags depend on the application domain, and on the related database.

One can find 2 types of XML documents (see fig.3) :

- Strongly structured documents tightly tag each information piece that have to be inserted in the database. Such documents can be classified as strongly structured documents, since all information for the database has to be tagged and correctly formatted. They can also be seen as flexible forms, as they allow to put free text around structured data.
- Weakly structured documents that tag paragraphs, so that a tag does not quote a data, but a piece of free text that needs to be analyzed to get data for the database. DRUID follows this case.

In this work, the application domain is the electronic medical record. The database was previously defined. We built the DTD manually, in accordance with the dedicated domain users' habits and with the database structure. We noticed that documents are divided into paragraphs that can be classified according to their semantics, and we noticed tables or subsets of attributes corresponding to each paragraph. We thus decided to define one tag for each paragraph type. We provide an example in the prototype section.

Figure 3. A strongly structured document followed by a weakly structured document (information for the database in bold)

```
<patient id='245'> <name> Dupont </name>
<first_name> Henri </first_name>
<prescription> Give <dose> 3 </dose> <dose unit>
pills </dose unit> of <medication id='12'> aspirin </
medication> <frequency> 3 </frequency> times a day
during <duration> 10 </duration> <duration_unit>
days </ duration_unit > </prescription></patient>
```

```
<patient id='245'><name> Dupont </
name><first_name> Henri </first_name>
<prescription> Give 3 pills of aspirin 3 times a day
during 10 days</prescription>
</patient>
```

How DRUID Goes From Information Pieces To Data

We need a correspondence between the tagged paragraphs and the attributes that may be filled in the database from these paragraphs. Correspondences are stored in a meta-model containing the list of attributes to be filled for each tag. The correspondence between a tag and an attribute is not limited to a simple link. It defines translation rules from the text in the paragraph into the attribute in the database.

A first step is based on pattern matching rules. Some pieces of text correspond to patterns, from which a value may be extracted. For example, the duration of a prescription is searched in the following patterns : <during x days> or <during x months>. In fact, patterns are composed of possibly coupling pieces of text from thesauri. In our example the pattern looks more like <"during" x duration-unit> where duration-unit takes its values in a thesaurus containing all possible values.

A formatting phase then translates the detected text into the appropriate attribute format.

How DRUID Fills In The Database

When a user finishes writing a document and validates it, the analyzer is launched. It parses the document and uses the mapping model to search for the values to put into the database. Once attributes have been detected and formatted, SQL queries are built and sent to the database.

Some additional information is necessary to identify the database table key values for the new attribute value. Most of the time, these values are given at the beginning of the document (a first Id paragraph is often written), or in the file system (name, location, creation date). Recognition of these key values is determinant for the efficiency of our system. At the moment, we define compulsory tags in each DTD to include key information that is not managed by the file system.

DRUID architecture

Meta-Model

The schema on figure 4 provides a UML class diagram of the meta-model used to extract information from the document and to insert them into the database.

We have divided this meta-model into 3 packages : the database world, the document world, and the mapping world to associate the two previous worlds.

The document world contains documents produced, DTD used to produce the documents and tags defined in the DTDs. Documents and DTDs are located with their URLs. Each tag belongs to one DTD, has a label, may be compulsory in the documents produced. There may be composition relationships between tags.

The database world refers to a simplified relational database dictionary. It represents tables and their attributes, noticing keys and foreign key relationships. A counter key is

an attribute whose value is incremented for each new occurrence without any semantic meaning. A specific URL attribute is added to each database table that may be filled from documents. It refers to the document from which data have been extracted.

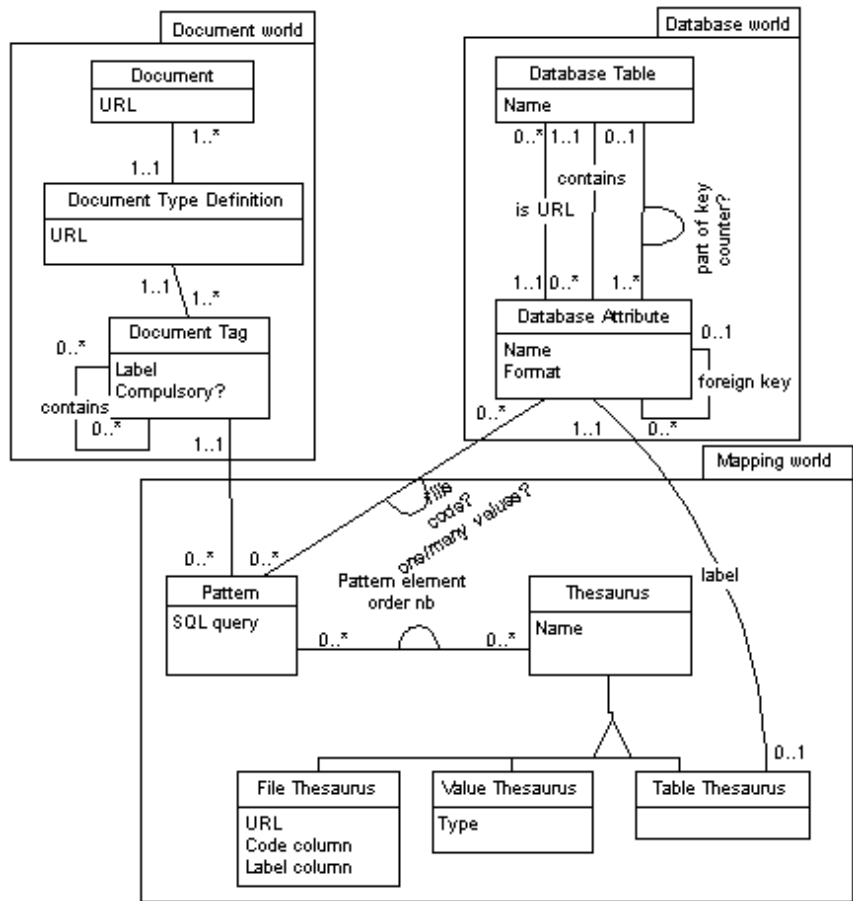
The mapping world is the center of the system. It requires more attention: it contains all information necessary to determine which information to search in which tagged paragraph and to fill which database attribute.

- A pattern is composed of an ordered list of pattern elements, each of which is extracted from a thesaurus (detailed later). A pattern also contains an SQL query to be built using the data found. For example, to insert the name of a medication from a “prescription” tagged paragraph, the pattern refers to a drug thesaurus and the SQL query is something like *insert into PRESCRIPTION values (\$patientcode, \$date, \$1)*. *\$patientcode* and *\$date* are key data extracted from the document and kept in mind to store other data in tables that use these key data as key attributes. *\$1* refers to the drugs thesaurus value found in the paragraph. Some attributes may be filled many times from one paragraph, it is allowed when the *one/many values?* attribute is set to MANY.

- We have defined 3 types of thesauri:
 1. File thesauri concern thesauri stored as files. These files contain one or two columns. One column contains patterns to be searched, the second (when existing) provides the code of the pattern. For example, in a medication file, the *pattern column* contains medications names while the *code column* contains medications identifiers. To search information in such thesauri, the system has to search for a pattern in the paragraph that corresponds to a label in the file. Most of the time, the database does not contain the label, but the identifier. When needed, the *code?* attribute of the *fills* association is set to true and the code of the detected label is put into the database rather than the label.
 2. Table thesauri concern thesauri stored as tables in the database. Such thesauri are similar to file thesauri. A label is given as an attribute of the table, code is the key of the table.
 3. Value thesauri refer to values. These are virtual thesauri that correspond to numbers, dates or strings. They cannot be extracted from any thesaurus but have to be put into the database. For example, the number of pills to be taken, the duration of the prescription, the name of the patient...

Here is an example of a composite pattern from the medical domain. The duration of a prescription contains 3

Figure 4 : Meta model to connect a database to a document capture system



parts: the pre-duration expression (during, to be taken...), the duration value (number) and the duration unit (hours, days, weeks, months...). Each part comes from a different thesaurus. All the possible values for the pre-duration expression are stored in a file. The duration value comes from a value thesaurus of integer type. The possible expressions for the duration unit are stored in a second file. The pre-duration expression is used to point at the text corresponding to the duration value but it will not be stored in the database. On the other hand the duration value and the duration unit are stored in the database. The duration value is stored in the database as it is. The duration unit is located using the label column of the duration units file; it is stored in the database using the code column of the same file.

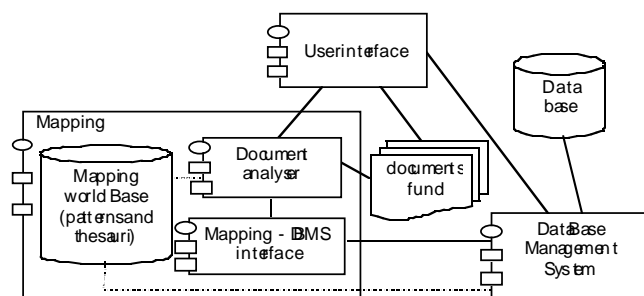
Software Architecture

The schema on figure 5 presents the 3-tiers architecture of our system, using the UML components diagram.

Our system contains three main components plus a database and a documents fund:

1. the database is the information database to be filled and queried;
2. the documents fund contains all documents created by

Figure 5 : Software architecture



end-users as well as DTD files;

- the user interface manages all interaction with the end-user: creating documents, querying documents, reading documents are the three main interaction scenarios. It has connections with the Mapping component for the integration of new documents, with the DBMS for querying data and existing documents (get URL), with the documents fund for reading documents;
- the database management system is a classical relational DBMS without any modification. It manages the information database, the dictionary of this database. It can manage the mapping world database if stored as a relational database;
- the Mapping component is the core of our system and we have divided it into three sub-components :
 - the Mapping world base refers to the mapping world part of the meta-model mentioned above. It contains references to all thesauri available, and classes containing the patterns to search and the SQL queries to generate. This mapping world base can be stored as a relational database (and thus can be managed by the DBMS), or as structured files.
 - the Document analyzer has to analyze documents and find new values for data. It may have to ask for help from the end-user if some compulsory information is not automatically detected in the document. It sends its results to the Mapping-DBMS interface component under the form of a list of SQL queries. It also communicates with the Mapping-DBMS component to get information about the database dictionary and the mapping world if stored in the DBMS;
 - the Mapping-DBMS interface manages the communication between the document analyzer and the DBMS. It asks for information concerning the Database dictionary and the Mapping world (as part of the meta-model). It transfers detected information to the database via SQL queries. It manages connection sessions with the DBMS. It also ensures that queries sent to the database are sent in an order compatible with foreign key references and other constraints on the database.

Document Analyzer Algorithm

The document analyzer algorithm provides the steps followed by the document analyzer component each time a new document is validated. Its algorithm is divided into two main parts : paragraphs location and queries generation. We present here a macro-algorithm for each of these two parts.

Prototype

Implementation

We have developed a prototype of this system. This prototype has been developed with the Java language and XML as major tools. We have used the IBM Alphaworks XML parser [Alphaworks, 2000] to parse the XML documents and identify paragraphs. This parser follows the DOM specification [W3C, 1998a] and the SAX 1.0 specification [Megginson, 1998].

- The document base contains XML documents and DTDs on the file system of the computer.
- The database is simulated by XML documents containing parts of the database dictionary. We have defined a XML

Figure 6 : Paragraphs location macro-algorithm

```

get the meta-model information - if inconsistencies detected, stop
parse the document into a DOM representation
for each paragraph of the document
  if this paragraph contains sub-paragraphs, recur on sub-paragraphs
  else
    get the tag and verify that it exists in the meta-model
    if it does not exist ignore it
    else provide the paragraph to the queries generation algorithm
    get the resulting queries
if a compulsory tag has not been encountered, generate an error
if not, transmit all queries to the Mapping-DBMS interface component
  
```

Figure 7 : Queries generation macro-algorithm

```

get the pattern information corresponding to the tag under processing
prepare a new tuple to be filled by this process according to the table it
refers to
get key attributes values:
  if it is a counter, generate a new value for this counter
  if it is a value coming from another paragraph,
    get the tag name and copy the value it tags
    verify that the key value does not already exist in the table
get an attribute value:
  get the related thesaurus
  find in the paragraph all expressions corresponding to the thesaurus
  if many expressions found and not many values possible generate an
  error
  if no expression found but compulsory, generate an error
  for each expression found, code it if necessary
if an insert query has to be generated, generate the query - if many,
generate as many as needed
if an update query has to be generated, verify that the occurrence to be
modified exists in the database ; if it exists generate the update query ;
otherwise generate an insert query - if many, generate as many as needed
return the query / queries
  
```

DTD to store the parts of the database dictionary our system needs. The database we used is presented in the following section. It is an excerpt of a real medical record database.

- The Mapping world base is stored in XML documents. We have defined a XML DTD to define the meta-model of the Mapping world base. Thesauri are all stored as file thesauri (except value thesauri) containing one column (label) and sometimes a second column (code).
- The DBMS component is not used, as all databases are represented using XML documents. Queries are presented on the screen rather than sent to a DBMS.
- The User interface component has not yet been implemented. At the moment, we create XML documents manually based on real documents. Much work has to be done for the user interface. The collaboration between the database, the thesauri and the user interface could efficiently improve the composition of documents. We are currently working on this problem, which may impact on the Mapping component (for example, if medications are captured using a “medication” combo box including the medication thesaurus, medications can be quoted by a specific tag and their analysis is thus highly simplified). We have just started an implementation of the user interface, shown on figure 11.
- The Mapping component is the core of our prototype. It analyzes XML documents and generates SQL queries in accordance with the macro algorithms provided above. Today, it can only do exact matching. We are studying existing systems that allow non-exact matching.

We have chosen to implement the mapping world as XML documents, mainly to test XML. We have defined a DTD for thesaurus description. Thanks to this DTD, we can create documents describing the thesauri used in each particular application.

We have also defined a DTD for mapping documents. Each DTD for the end-user is associated to a mapping document that explains the mapping for each tag in the end-user DTD. Each tag in the end-user DTD that has a link to the database is associated a pattern in the mapping document.

We do not provide these DTDs in this paper because they are quite long and necessitate a lot of explanations.

Example On Medical Patient Record

This paragraph presents an example of the work of our prototype on a medical patient record. We do not provide all XML documents produced (thesauri XML description and mapping documents) because they are long to explain.

The medical record database used is a simplified version of a real relational medical records database. It contains the following tables :

A Patient is identified by a number. We also store his name and family name. The URL attribute refers to the documents used to provide the occurrence.

Figure 8 : Database model

Underlined attributes take part of the key of the table.
Italic attributes are foreign keys.

patient (<u>patientId</u> , name, <i>familyName</i> , URL)
encounter (<u>encounterId</u> , <i>patientId</i> , date, URL)
disease (<u>diseaseCode</u> , <i>diseaseLabel</i>)
medication (<u>medicationCode</u> , <i>medicationLabel</i>)
problem (<i>encounterId</i> , <i>diseaseCode</i>)
prescription (<i>encounterId</i> , <i>medicationCode</i> , quantity, duration, durationUnit, frequency, frequencyUnit).

An Encounter occurrence gathers all information noticed during an encounter between the patient and a practitioner. It references the patient encountered (patient Id), provides the date of the encounter and the URL of the document produced concerning this encounter. Other encounter information are stored in two other tables : the problem table and the prescription table.

The Problem table contains all the diseases detected during encounters. The diseaseCode is a reference to the diseaseCode attribute of the disease table.

The Prescription table refers to the encounterId of the corresponding encounter. It also refers to the medication prescribed, and provides posology information (duration, frequency, dose)

Based on this database model and on hand-written documents, we have defined the following DTD for encounter documents. Each time a physician meets a patient, he writes a document of this type.

The following figure presents an encounter document written by a physician. It follows the encounter DTD given above.

This document is not captured as is: the end-user captures it in an interface that hides the XML structure of information. The window on figure 11 provides a skeleton of the user interface we have just started to implement. The left part of the window contains the tree structure of the document, i.e. paragraphs and their content. The right part of the window presents information as captured by the end-user. He

Figure 9 : DTD for encounter documents

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!ELEMENT encounter (patient_id, date, problem?,
prescription*, comment)>
<!ATTLIST encounter mappingBase CDATA
#FIXED "encounterMapping.dtd">

<!ELEMENT patient_id (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT problem (#PCDATA)>
<!ELEMENT prescription (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
```

Figure 10 : An encounter document

```

<?xml version='1.0' encoding='ISO-8859-1' standalone='no'
?>
<!DOCTYPE encounter SYSTEM "encounter.dtd">
<encounter>
<patient_id> 2 </patient_id>          <date> 11-06-1999 </
date>
<Problem>The problem concerns the left eye. eye pain. red
eye. Afraid of eye illness.</problem>
<prescription>Visiodose every 2 days during 7 days</
prescription>
<prescription>1 pill A Vitamin (1 week) </prescription>
<comment>verify the eye does not change aspect</comment>
</encounter>

```

captures free-text paragraphs. When a paragraph is written, he selects its tag using a pop-up menu. Patient id and date are the identifiers of the document. They are not captured here by the end-user. They are taken from previous windows that allow to select a patient in a patient list and to propose the current date as encounter date.

The analysis of this document by our prototype provides six SQL queries. They are given in figure 12. Diseases are coded using the ICPC classification [WONCA, 1997] and medications using an excerpt of a French drugs databank [Flory, 1983].

Conclusion

This paper provides a description of the DRUID system. This system can be used to structure information in a database using weakly structured documents. We show how it is possible to provide a system associating documents capture freedom with database storage structure. A prototype allowed us to see that this system can be used with interest by end-users. In particular we use it for medical records and doctors show a great interest for this system.

Figure 11: Skeleton of the end-user interface to capture weakly structured documents

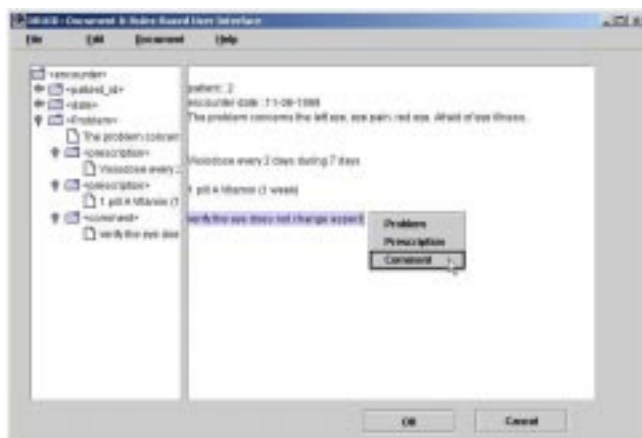
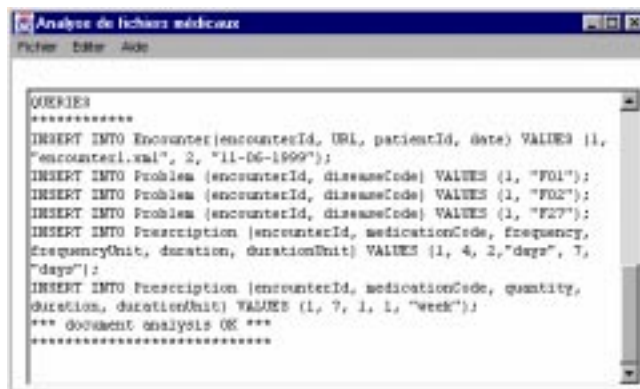


Figure 12 : Prototype results on the document



The system we propose here provides much capture freedom to end-users, without limiting data management capabilities. Information captured in documents are transformed into data sent to a traditional domain-oriented database. Exact queries and statistical studies can be performed in the same way as if data were directly captured in form-based windows. As opposed to systems found in the literature, capture techniques do not impact data management power. Our proposal can be seen as a package including a user interface and an analyzer that should not replace the capture interface of the existing application. We see it as an alternative capture module, without any impact on host system data management policy.

The proposal we presented here does not intend to be a universal capture system. It cannot be adapted to any application domain. It can only deal with domains that have classifications, thesauri and databanks that refer the domain technical terms. Matching rules are much based on specific terms and cannot apply to any free speech domain. Another drawback of the current system concerns the necessary time to build extraction rules. In the future, we intend to study automatic definition of DTDs according to an existing database model, so that the fastidious step of rules description could partially be assisted.

This prototype has to be improved. Currently rules can only do exact matching. We have to release this huge drawback in order to make the system usable in end user daily work. For example, we have to make the analyzer misspelling-resistant.

Moreover, we are studying a more complete analysis mechanism based on transformation rules. Transformation rules contain 4 steps to select, extract, structure information and to build queries to the database. We also will have to take into account command sentences, that are at a meta-level. How to manage sentences like "replace prescription of Aspirin made yesterday by..."

We are also currently working on the user interface to capture documents. As presented in the previous figure, we

try to simplify XML tagging at the end-user view. We also want to include tools like end-of-word prediction, so that the end-user stating to write “aspi” is proposed to have directly “aspirin”. This end-of-word prediction should be context-driven: according to the paragraph type, some thesauri words have a high probability to appear.

Extraction validation is also a great issue, especially in domains like medicine: we imagine that the analyzer builds first a strongly structured documents that can be validated by the end-user before sending data to the database.

We are currently developing a robust prototype including all specifications we presented in this article, as well as a validation mechanism and improved transformation rules. This prototype is due next summer and will allow experiments in daily work dimension.

References

- [Abiteboul, 1997a] Abiteboul S. and al (1997). Querying Documents in Object Databases. *Int. Journal of Digital Libraries*, 1(1), 5-19.
- [Abiteboul, 1997b] Abiteboul S. and al (1997). The Lorel query language for semistructured data. *Int. Journal of Digital Libraries*, 1(1), 68-88
- [Abiteboul, 2000] Abiteboul S., Buneman P., Suciu D. (2000). *Data on the Web : From relations to semi-structured data and XML*. Morgan-Kaufmann, San Francisco. 257 p.
- [Alphaworks, 2000] Alphaworks (2000). *XMLAJ - XML parser for Java*. Retrieved from the World Wide Web: <http://www.Alphaworks.ibm.com/tech/xml>
- [Atzeni, 997] Atzeni P., Mecca G., Merialdo P. (1997). Semistructured and structured data in the web : going back and forth. *ACM SIGMOD Workshop on Management of semi-structured data*, Tucson, Arizona, may 1997, 1-9
- [Baeza, 1996] Baeza-Yates R. and Navarro G. (1996). Integrating contents and structure in text retrieval. *ACM SIGMOD record*, 25(4), 67-79
- [Biezunski, 1995] Biezunski M. (1995). Modeling hyperdocuments using the topic map architecture. *Proc. Int. Hytime Conference. Vancouver*, august 1995. Retrieved from the World Wide Web: <http://www.techno.com>
- [Blanquet, 1999] Blanquet A., Zweigenbaum P. (1999). A Lexical Method for Assisted Extraction and Coding of ICD-10 Diagnoses from Free Text Patient Discharge Summaries. *J. Am. Medical Informatics Association*, 6(suppl.)
- [Bohm, 1997] Bohm K., Aberer K., Neuhold E.J. and Yang X. (1997). Structured document storage and refined declarative and navigational access mechanisms in HyperStorM. *VLDB journal*, 6(4), 296-311
- [Buneman, 1996] Buneman P., Davidson S. and Hillebrand G. (1996). A query language and optimization techniques for unstructured data. *ACM SIGMOD'96*, 505-516
- [Christophides, 1994] Christophides V., Abiteboul S., Cluet S. and Scholl M. (1994). From structured documents to Novel query facilities *ACM SIGMOD Records*, 23(2), 313-24
- [Cluet, 1999] Cluet S., Deutsch A. et al. (1999). XML query languages : experiences and exemplars. Retrieved from the World Wide Web: <http://www-db.research.bell-labs.com/users/simeon/xquery.html>
- [Cohen, 1998] Cohen WW (1998). A Web-based Information system that reasons with structured collections of text. *Proc. Autonomous Agents AA'98*, 400-407. Retrieved from the World Wide Web: <http://www.research.att.com/wcohen/postscript/agent-98.ps>
- [Delcambre, 1997] Delcambre L. et al. (1997). Structured Maps : Modeling Explicit Semantics over a Universe of Information. *Int. Journal of Digital Libraries*, 1(1), 20-35
- [Deutsch, 1999a] Deutsch A., Fernandez M., Florescu D., Levy A. and Suciu D. (1999). A Query Language for XML. *Int. WWW Conference*. Retrieved from the World Wide Web: <http://www.research.att.com/~mff/files/final.html>
- [Deutsch, 1999b] Deutsch A. Fernandez M. Suciu D. (1999). Storing Semi-Structured Data in Relations. *Proceedings Workshop on query Processing for Semi structured data and non-standard data formats*. Retrieved from the World Wide Web: <http://cheops.cis.upenn.edu/~adeutsch>
- [Dobson, 1995] Dobson SA, Burrell VA (1995). Lightweight databases. *Proc. Int. WWW conf.*, 282-288
- [Embley, 1998a] Embley D.W., Campbell DM., Smith R.D., Liddle S.W. (1998). Ontology-based extraction and structuring of information from data-rich unstructured documents. *Int. Conference of Information and Knowledge Management -CIKM'98*, 52-9
- [Embley, 1998b] Embley D.W., Campbell D.M., Jiang Y.S. et al (1998). A conceptual modeling approach to extracting data from the Web. *Int. Conf. on Conceptual modeling (ER'98)*, 78-91
- [Flory, 1983] Flory A., Paultre C., Veilleraud C. (1983). A relational Databank to aid in the Dispensing of Medicines. *MedInfo'83*, 152-5
- [Frenot, 1999] Frenot S. and Laforest F. (1998). Medical Records Management Systems : criticisms and new perspectives. *Methods of Information in Medicine*, 38: 89-95
- [Goldman, 1997] Goldman R., Widom J. (1997). Data Guides : Enabling Query Formulation and Optimization in Semi structured Databases. *23rd VLDB*, 436-45
- [Goldstein, 1999] Goldstein J. and al. (1999). Summarizing Text Documents: Sentence Selection and Evaluation Metrics. *ACM SIGIR'99*, 121-8
- [ISO, 1986] International Standard. (1986). *Information processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*, ISO 8879.
- [Jackson, 1998] Jackson P. and al.(1998). Information Extraction from Case Law and Retrieval of Prior Cases by Partial Parsing and Query Generation. *ACM CIKM'98*, 60-7
- [Kaskiel, 1997] Kaskiel M., Zobzl J. (1997). Passage retrieval revisited. *ACM SIGIR'97*, 178-85
- [Kilpelainen, 1993] Kilpeläinen P. and Mannila H. (1993). Retrieval from hierarchical texts by partial patterns. *ACM SIGIR*, 214-22
- [Konopnicki, 1998] Konopnicki D. and Shmueli O. (1998). Information Gathering in the World Wide Web : The W3QL Language and the W3QS System. *ACM Transactions on Database Systems*, 23 (4), 369-410
- [Laforest, 1999] Laforest F., Tchounikine A. (1999). A model for querying annotated documents. *Proc. ADBIS'99, LNCS 1691*, 61-74
- [Lakshmanan, 1996] Lakshmanan L., Sadri F. and Subramanian I. (1996). A declarative language for querying and restructuring the Web. *IEEE Workshop on research issues in data engineering (RIDE'96)*, 12-21

- [LeMaitre, 1998] Le Maitre J., Muriasco E. Rolbert M. (1998). From annotated Corpora to databases : the SgmlQL language. *Linguistic Databases, CSLI Lecture Notes n°77*, J. Narbonne (Ed.), 37-58.
- [McHugh, 1997] McHugh J., Abiteboul S. and al. (1997). Lore : a database management system for semistructured data. *SIGMOD Records*, 26(3), 54-66
- [Megginson, 1998] Megginson Technologies (1998). *Sax 1.0 (Java): Road Map*. Retrieved from the World Wide Web: <http://www.megginson.com/SAX/SAX1/roadmap.html>
- [Mendelzon, 1997] Mendelzon A.O., Mihaila G.A. and Milo T. (1997). Querying the World Wide Web. In: *Journal Of Digital Libraries*, 1(1), 54-67
- [Neven, 2000] Neven F., Schwentick T. (2000). Expressive and efficient pattern languages for tree-structured data. *Proc. ACM PODS 2000*, 145-56
- [Riahi, 1998] Riahi F. (1998). Elaboration automatique d'une base de données à partir d'informations semi-structurées issues du Web *Actes du congrès INFORSID 98*, 327-43
- [Salton, 1986] Salton G. (1986). Another look at automatic text-retrieval systems. *Communications of the ACM*, 29(7), 648-656
- [Stairmand, 1997] Stairmand M. (1997). Textual Context Analysis for Information Retrieval. *ACM SIGIR'97*, 140-7
- [Stonebraker, 1983] Stonebraker M., Stettner H., Lynn N., Kalash J., Guttman A. (1983). Document processing in a relational database system. *ACM TOIS journal*, 1(25)
- [W3C, 1998a] World Wide Web Consortium (1998). *REC-DOM-level-1-19981001* Retrieved from the World Wide Web: <http://www.w3c.org/TR/REC-DOM-Level-1>.
- [W3C, 1998b] World Wide Web Consortium (1998). *Extensible Markup Language (XML)* Retrieved from the World Wide Web: <http://www.w3.org/XML/#9802xml10>
- [WONCA, 1997] The WONCA International Classification Committee (1997). *The International Classification of Primary Care*. Retrieved from the World Wide Web: <http://www.ulb.ac.be/esp/wicc/table-en.html>

Frederique Laforest received the title of Engineer in Computer Science from the National Institute of Applied Sciences of Lyon (INSA Lyon - France) in 1993. She received a PhD degree in Computer science at INSA Lyon in 1997 with a thesis entitled "Generic models and user interfaces – application to medical information systems". From 1996 to 1998, she was assistant professor at the Department of Informatics of the Technological Institute of Lyon 1 University. Since 1998, she is associate professor at the Telecommunications, Services and Uses Department of INSA-Lyon. Her teaching fields are Algorithms and Programming, Databases, Operating Systems and Information Systems. Since 1993, she is working in the Information Systems Laboratory (LISI). Her major research topics concern information systems, documents and databases. She has always applied her research studies to the medical domain and has thus acquired a certain knowledge of this application domain problematics.

Andre Flory is a full professor of databases and information system in the department of computer sciences at the national institute of applied sciences (INSA) of Lyon (France) since 1989. He is currently the director of the computer sciences resources center of the institute. From 1970 to 1979 he was associated professor at the Claude Bernard University (Lyon) and from 1979 to 1998 professor in the MIS department of Jean Moulin University (Lyon). He has served as a visiting professor at the MIS department of the university of Minnesota in 1984. He has worked on database design for many years and his current research interest include medical information system. He is author of more than 100 papers published in conference proceedings, books and scientific journals. He has served in the program committees of many international conferences and has participated in several research projects sponsored by the French government. He also served as expert at the European economic communities.