What can XML do for textual information retrieval on the WEB? *

Emmanuel Nauer, Rim Al Hulou and Amedeo Napoli
LORIA – UMR 7503
B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France
Email: {nauer,alhulou,napoli}@loria.fr

Abstract

In this paper, we propose an analysis of the links existing between (1) semi-structured data or SSD, i.e. rough and heterogeneous data without fixed structure (2) XML, the language of description of textual documents (3) object-based representation systems or OBRS. The manipulation of SSD comes from several needs such as data integration, information retrieval on the Web and text mining. Due to its characteristics, XML may be used for describing SSD and can be considered as a bridge between SSD and OBRS, OBRS being mainly used for their reasoning capabilities. We conclude the paper with a discussion enlightening the benefits of using XML and OBRS for manipulating SSD and solving problems involving SSD.

1 Introduction

In this position paper, we are interested in the role that can be played by XML for reasoning and problem solving on textual data. Examples of problems posed on textual data are the merging of different textual sources (from the WEB for example), guided navigation on the WEB, text mining and scientific and technical information analysis. Information retrieval, e.g. document retrieval based on keywords, and reasoning, are of main importance for solving these problems. Information retrieval and reasoning can be enhanced if semantic aspects can be taken into account, e.g. semantic relationships between textual data or documents. One approach is to take advantage of a model of the domain of textual data using a knowledge-based system, whose knowledge base represents this model. In our work, we are mainly interested in object-based representation systems and classification-based reasoning.

Content-based information retrieval on the WEB is not an easy task since the information accessible through the WEB is distributed among many sources, each source providing a different interface and exporting the data in different format. The major portion of the WEB data is known to be semi-structured data (SSD), data which are irregular and not conformed to a global fixed schema [1]. The problem of representing

*This paper also appears in the proceedings of XOT'2000 (ECOOP workshop on XML and object technology).

SSD is important in many application areas such as multimedia data management, biological databases, digital libraries, and data integration. Our goal is to represent textual SSD–using a knowledge representation formalism– and to reason with these data to solve problems on the data domain [9][3] such as text semantic querying, text mining, text analysis, reasoning on queries (classification of queries [8], querying by analogy).

Using an unified format for describing heterogeneous data selected from different sources facilitates the link between these data and the knowledge representation formalisms. In our approach, we use XML as a bridge between SSD and knowledge representation formalism. XML (eXtensible Markup Language) [7] is an emerging standard for describing textual data, developed by the World Wide Web Consortium. It is believed to become the format for textual data exchange on the WEB. Moreover, textual SSD can be taken into account and described within XML structures, which are then transformed as object in an object-based representation system (OBRS).

This paper is organised as follows. In section 2, we present a simple example to illustrate the need of reasoning with textual SSD and we present the use of an OBRS for supporting such a reasoning task. In section 3, we motivate the use of XML as an intermediate format between SSD and OBRS. The cooperation between XML and OBRS is discussed in section 4. Finally, section 5 concludes the paper.

2 Why do we need reasoning for manipulating textual SSD?

2.1 Example of textual SSD

In this section we introduce a motivating example, using the sample databases presented in Fig. 1. These data were selected from three different sources: `Food`, `Wine` and `French Wine`, which respectively contain data about different dishes, data about wine characteristics and data about some French wines.

The data illustrated in our example can be considered as SSD, which can be characterised as follows:

- Data may have different formats.
- Data are not strongly typed, meaning in particular that attributes with the same name may have different types.
- Data have their own structures (and thus their own characteristics).
- Data evolve with a high frequency.

Due to data heterogeneity and the absence of a global schema for describing data, the manipulation of SSD with conventional query languages is rather difficult [1].

French Wine

```
Wine{ Name      : Sylvaner
      Year      : 1996
      Productor { Name    : Gustave Lorentz
                  Contact { Address: 68750 Bergheim
                            Tel    : 03 89 73 22 22 }}
      Price     : 30.00 FF }

Wine{ Name      : Riesling Cuvée Particulière
      Type      : vin blanc alsacien
      Year      : 1997
      Productor : G. Ruhlmann
      Price     { FF     : 45.00
                  Dollar : 8.02 } }

Wine{ Name      : Chablis Grand Cru Bougros
      Year      : mil-neuf-cent quatre-vingt quinze
      Origin    : Bourgogne
      Productor : William Fevre }
```

Wine

```
Name   : Riesling
Flavor : delicate
Color  : white
Sugar  : dry
Region : Alsace

Name   : Tokay pinot gris
Color  : white
Sugar  : dry
Region : Alsace

Name   : Sylvaner
Color  : white
Sugar  : dry
Region : Alsace

Name   : Chablis
Body   : medium
Color  : white
Sugar  : dry
Region : Bourgogne
```

Food

Dish	Category	Color	Companion wine
Trout	fish	red	Sylvaner
Salmon	fish	red	Riesling
Pikeperch	fish	white	Riesling Tokay pinot gris
Devilfish	fish	white	dry white wine
Chicken	meat	white	Tokay pinot gris

Figure 1: Example of semi-structured data

2.2 Object-based representation systems

In order to take into account and to manipulate SSD and to draw inferences on these data –for knowledge discovery and problem-solving purposes– we exploit the notions of *semi-structured objects* and *polythetic concepts*. In our approach, we work with an *object-based representation system*, that is based on a hierarchy $\mathcal{H}$ of *concepts* (or *classes*), where concepts are related by a *specialisation* (or *subsumption*) relation. The hierarchy $\mathcal{H}$ corresponds to the knowledge base of the system.

OBRS are used for representing knowledge and for reasoning in order to solve problems in a given domain. Reasoning is based on inheritance and classification. Inheritance is used for knowledge sharing in the hierarchy $\mathcal{H}$, where concepts inherit all the properties of their super-concepts, while classification is used to insert a new concept in $\mathcal{H}$, to determine the possible concept of an *object* (or *individual*), to handle queries and to draw inferences [6].

In order to represent SSD in the context of an OBRS, the concepts of the hierarchy $\mathcal{H}$ are no more considered as *monothetic* concepts but as *polythetic* concepts [19][14]. For a monothetic concept, the possession of a unique set of features is both necessary and sufficient for an object to be a member of the extension of the concept. For a polythetic concept, this is no longer the case: a polythetic concept is considered as a combination of a conjunction and a disjunction of attributes (rather than a conjunction of attributes). More precisely, a concept `C` can be formalised as $\texttt{C} = a_1 \sqcap a_2 \sqcap ... \sqcap a_n$, where a_i denote an attribute; some of the a_i are "simple" or primitive attributes, i.e. not decomposable attributes, while some of the a_i can be of the form $a_i = b_{i_1} \sqcup b_{i_2} \sqcup ... \sqcup b_{i_n}$, where b_{i_j} denote primitive attributes.

Relying on the notion of polythetic concepts and following the ideas and the formalism introduced in [4], a *semi-structured object* is defined as an object without any concept, i.e. an instance of the root of the hierarchy $\top$. In order to discover regularities in a set of semi-structured data, these data are represented as semi-structured objects and then classified in polythetic concepts. The hierarchy $\mathcal{H}$ of polythetic concepts is then used as a *model* of the domain of the data. The hierarchy $\mathcal{H}$ plays a role similar to that of the *data guide* –a kind of *global schema* for data– introduced in [1]. In that way, SSD can be represented as semi-structured objects and hierarchically organised in order to be manipulated for reasoning purposes (e.g. query answering and reasoning on queries).

For example, the SSD presented in Fig. 1 can be represented by the two hierarchies of concepts given in Fig. 2.

The classification process can be used for conceptual clustering purposes. In particular, the concept definitions and the specialisation relationships in the two hierarchies given in Fig. 2 have been built using the classification process as a conceptual clustering algorithm applied on XML structures describing SSD (the use of XML for describing SSD is explained in section 3).

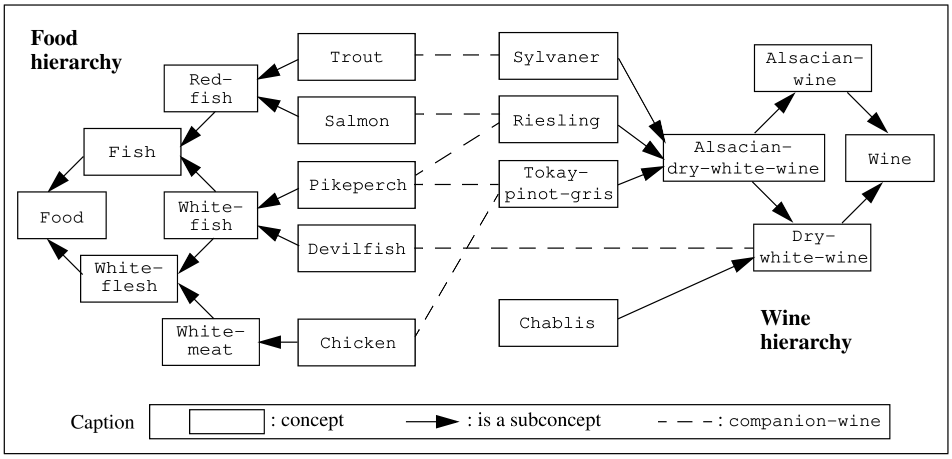

Figure 2: Two hierarchies of concepts representing SSD. The `companion-wine` relationship of the base `Food` is associated with the concepts in the bases `Wine` and `French Wine`.

2.3 Examples of reasoning with SSD in an OBRS

In this section, we show how to take advantage of reasoning capabilities in an OBRS for querying a set of objects representing SSD. We illustrate this reasoning capabilities to answer the four following queries:

Q_1. Existential query: what is the set of wines which are companion-wines of *at least* one Fish? (i.e. What can I drink with Fish?)

Q_2. Specialised existential query: what is the set of wines which are companion-wines of Devilfish (i.e. What can I drink with Devilfish?)

Q_3. Specialised existential query: what is the set of wines which are companion-wines of Devilfish and which are not Alsacian-wine (i.e. Knowing that I do not like the Alsacian-wine, what can I drink with Devilfish?)

Q_4. Universal query: what is the set of wines which are companion-wines of *all* White-flesh? (i.e. What can I drink with any type of White-flesh?)

Answering these questions using a relational database management system is rather difficult. With such a system, there is no way to find objects sharing a set of properties, neither to qualify the type of the objects answering a query, e.g. if the instances of a concept Y provide an answer to a query Q, then the instances of subconcept X of Y can be also considered as answers for Q.
For answering the above queries, we can take advantage of the hierarchical structure of the object base.

A_1. For Q_1, a specialisation operation will provide Chablis, Sylvaner, Riesling and Tokay pinot gris (all wines which are subconcepts of Dry-white-wine) as companion-wine of Devilfish. Sylvaner, Riesling and Tokay pinot gris are the companion-wine of Trout, Salmon and Pikeperch. So the set {Chablis, Sylvaner, Riesling, Tokay pinot gris} can be the first answer to Q_1. This set is the union set of four subqueries.
Dry-white-wine is a more abstract answer that can be given by a generalisation operation. The three companion-wine of Trout, Salmon and Pikeperch are all subconcepts of Alsacian-dry-white-wine which is in turn a subconcept of Dry-white-wine. Moreover, Dry-white-wine is the companion-wine of Devilfish. So the type of the companion-wine of Fish is Dry-white-wine.

A_2. For answering Q_2, a specialisation operation will provide a complete answer. The concept base tells us that Dry-white-wine is the type of the companion-wine of Devilfish. Thus, Chablis, which is a Dry-white-wine is an answer, but the three Alsacian-dry-white-wines are candidate too: thus, the complete answer is the set {Chablis, Riesling, Sylvaner, Tokay pinot gris}.
Answering Q_2 needs to examine the concept Devilfish and the range of attribute companion-wine of this object. Every instance of this range may be an answer to Q_2.
One can note that the attribute companion-wine of an object may be filled with a set of individual objects. In this case we can also induce the type of the values of the attribute companion-wine which is given by the most common specific super-concept of all concepts from which these individual objects are instances.

A_3. Q_3 can be answered by taking into account the fact that Q_3 is more specific than Q_2: this is an example of reasoning on queries, i.e. recognising that a

query is more specific than another query (see for example [8]). So the answer of Q_3 can be obtained using the result of Q_2 and adapting the answer to the constraint associated with Q_3. In this case, concepts depending of the concept `Alsacian-wine` will be pruned and the answer will be `Chablis`.
Building the answer to Q_3 needs to split the set of concepts of Q_2 into a subset verifying the constraint "not `Alsacian-wine`" and a subset including concepts that do not verify the constraint.

A_4. For answering Q_4, it is necessary to explore the specialisations of the concept `White-flesh` and the respective ranges of the attributes `companion-wine` of these specialisations. In our case, the concepts `Chicken`, `Pikeperch` and `Devilfish` have respectively as `companion-wine` {`Tokay pinot gris`}, {`Riesling`, `Tokay pinot gris`} and {`Chablis`, `Tokay pinot gris`, `Riesling`, `Sylvaner`} (as subconcepts of `Dry-white-wine`). The intersection of these three sets is {`Tokay pinot gris`} which is the answer to Q_4. Note that, by contrast with Q_1, the answer here depends on an intersection of sets while the answer of Q_1 depends on a union of sets.

Answering these four queries needs not only a classification operation but also generalisation and specialisation operations for finding complete answers in the concept hierarchy.

3 XML as a bridge between SSD and objects: why?

Using an intermediate format between SSD and OBRS provides interoperability facilities. The only constraint is the transformation of SSD coming from multiple sources into common description format. Thanks to this common description format, it is then possible to apply different engineering facilities on the same data without any further transformations.

The choice of XML as this intermediate format allows the use of all the XML engineering tools (parser, query languages, etc.) and facilitates the publications of results directly on the WEB, thanks to XSL (XML Stylesheets Language) for example.

We will first describe the advantages of using an intermediate format. In a second step, we detail in which cases XML will have an important role to play.

3.1 Why using an intermediate format?

There are many advantages of using an intermediate format:

- Having a common format of information simplifies the data interoperability and the data exchange. From the information integration point of view, this corresponds to the homogenisation of the textual data at the character and structural levels. This provides facilities to manage data because the same engineering facilities can be applied to these data. For example, the `Wine` and `Food` hierarchies

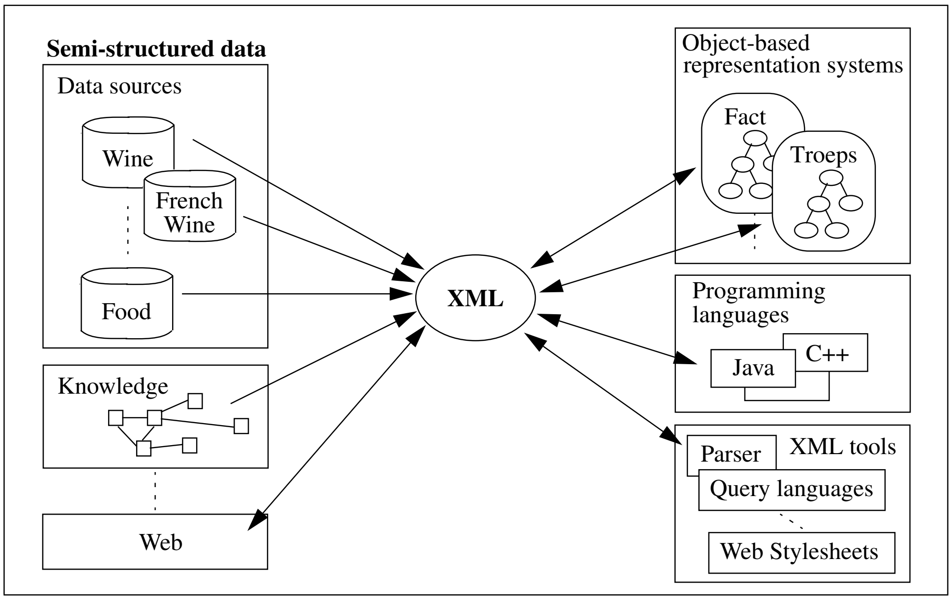

Figure 3: Using XML as an intermediate format between SSD and tools.

presented in Fig. 2 can be obtained by applying the same conceptual clustering tool [18] to the XML data describing the raw data of the `Wine` and `Food` databases.

Moreover, a single format for representing data allows the use of the most adapted tool for solving a given problem. For example, all OBRS do not have the same possibilities of representation and the same possibilities of data processing. To solve complex problems, it can be convenient or even necessary to use several representation and reasoning facilities, and to combine the advantages of each of them.

- A common formalism for encoding the inputs and the outputs of tools simplifies the communication between tools. Such a choice allows to solve a problem by decomposing it into simpler sub-problems. The tools solving each sub-problem will be linked one after the other. This principle of the modular programming approach is detailed in [15].

- An intermediate format reduces the number of transformations from one format to another. The transformation from `M` source formats to `N` target formats needs only `M+N` transformation operations (by contrast, `M*N` operations are necessary when direct transformations are processed). This shows that a common *bridge* format is of main importance when data provided by multiple sources have to be processed.

3.2 Why XML?

Choosing XML as an intermediate format may be justified by a number of arguments:

- XML is easy to read, to understand and to use.
- XML is well adapted to describe textual SSD.
- XML is becoming a standard for document description and has a (very) rich environment.

3.2.1 Simplicity of the XML format.

The main principles of the XML language are simple: (1) a beginning and an ending mark (i.e. tags) identify the logical units, (2) the composition of elements is done by including logical units one into each other. These two principles allows the description and the structuration of data in a simple and natural way: XML format is easy to read and to understand.

The clear distinction between the container and the contents (given by the tagged structures) favours the use of lexical and syntactic tools.

3.2.2 Adaptability of XML to other existing formats.

The conversion of raw textual data into XML data can be done most of the time by a standard lexical analysis. It is easy to define a XML structure for describing the SSD of our example (see Fig. 1). For example, we can transform the first descriptions of the `French Wine` and `Wine` databases into:

```
<Wine>
    <Name>Sylvaner</Name>
    <Year>1996</Year>
    <Productor>
         <Name>Gustave Lorentz</Name>
         <Contact><Address>68750 Bergheim</Address>
                  <Tel>03 89 73 22 22</Tel>
         </Contact>
    </Productor>
    <Price currency="FF">30.00</Price>
</Wine>
```

and

```
<Wine>
    <Name>Riesling<Name>
    <Flavor>delicate</Flavor>
    <Color>white</Color>
    <Sugar>dry</Sugar>
    <Region>Alsace</Region>
</Wine>
```

In the same way, many of the formats currently used in document applications can be transformed into XML [12].

3.2.3 The appropriateness of XML to SSD characteristics.

In the XML description of the SSD given above, we can notice that XML allows to take into account the characteristics of textual SSD illustrated in 2.1, briefly:

1. The translation of data into XML can be done using a lexical analysis (here, the use of LEX, a lexical analyser generator, is, for example, very convenient)

2. In XML, it is possible to describe data without defining a schema and this is important for taking SSD into account.

3. Data and structure are described together in nested tags, where tags define the structure and data are located between tags.

4. It is possible to describe the same data element using different structured types: data in XML are grouped into elements delimited by tags designing data structure, each XML element may contain data or other XML elements.

3.2.4 XML: a standard for textual data description.

XML was developed to facilitate the description and the exchange of data on the WEB. At present, XML has gained interest from a large number of communities. Many proposals are made to describe all kinds of information in XML: the TEI (Text Encoding Initiative) is one of the most known, but we can also notice CML (Chemical Markup Language), BSML (Bioinformatic Sequence Markup Language), BIOML (BIOpolymer Markup Language), OCKML (Ontology and Conceptual Knowledge Markup Languages), etc. (see [10] for more application examples). This collective *passion* for XML is giving to XML the status of a standard for document description. Indeed, the main objective for creating XML was to describe and to structure textual data and documents on the WEB.

3.2.5 The XML environment.

XML has a rich and attractive environment. In this environment, we can for example mention:

- The possibility to define a data schema through a DTD (Document Type Definition). The DTD is a grammar which defines the data structures through composition rules. For example, the element definition `<!ELEMENT Wine (Name, Year, Productor, Price*, Type?, Origine?)>` states that an element `Wine` is composed of one `Name`, one `Year`, one `Productor`, `0` or more `Price`, with the two optional fields `Type` and `Origin`. A more expressive way to define a schema is given by the XML-SCHEMA specification [20][5]. An interesting aspect in XML is that the schema, i.e defining a DTD, is optional. This is essential for manipulating SSD that have no schema. On the other hand, it is possible to build a schema and to exploit it in some cases such as query optimisation for example.

- The DOM (Document Object Model) [2] which provides manipulation facilities for XML data. Information about the DOM will be detailed further, in the part explaining the role that the DOM can play to link SSD with objects.

- The query languages (XML-QL [11], XQL [17]) which allow information retrieval or data extraction. These query languages take into account the characteristics of SSD: irregularity of the structure, multiple types which can correspond to an element, incomplete knowledge about the structure.

4 XML as a bridge between SSD and objects: how?

4.1 Manipulating XML data as objects

As mentioned before, an OBRS provides the ability to classify XML data. For specific processing, it is also interesting to handle XML objects because it is more expressive to manage objects such as `Wine` or `Productor` (with their own properties and methods) instead of their corresponding XML tree. We will now present two approaches to link XML data with objects. The first one consists in using the DOM to manage XML data and to convert them into specific application objects. The second one is to exploit the data schema (given by a DTD or another schema specification like XML-SCHEMA) to build the corresponding concepts automatically.

4.1.1 Converting XML data into classes using the DOM (Document Object Model).

The Document Object Model [2] is an API designed to manage XML data. This interface defines how a program can dynamically have an access to the content and the structure of XML data. The DOM defines an abstract data type which describes XML data as a tree. In this model, a tree is composed of various node types (root, element, textual data, attribute, etc.) for which the DOM defines management functions. So, thanks to the DOM, the representation of XML data and the access to these data can be done in a standardised way, for which a parser can be built. Such a parser may, for example, take XML data as inputs and transform them into another data model (see Fig. 4). This is helpful for building a concept hierarchy taking XML structures as input.

4.1.2 Converting XML data into concepts thanks to a schema.

The transformation of XML structures into objects may be done automatically according to certain representation choices and constraints. For example, see Fig. 5, the schema can be represented by concepts and the XML data can correspond to instances of these concepts but other representation choices may be done [13].

Thus, if we use a definition of schema given by a DTD, the declaration `<!ELEMENT Wine (Name, Year, Productor, Price*, Type?, Origine?)>`
will define the following concepts:

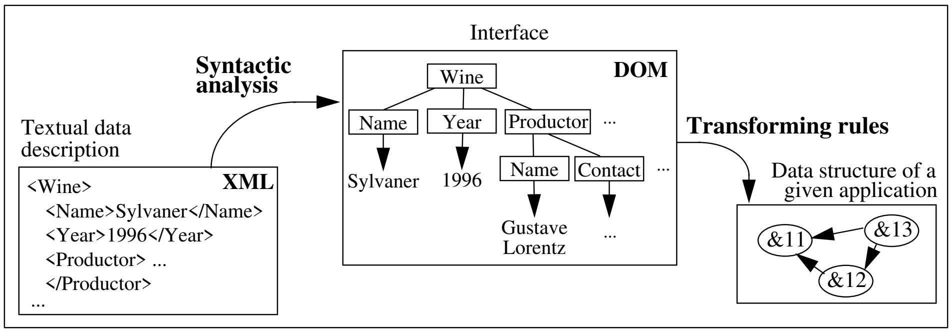

Figure 4: Using the DOM to transform XML data into another data structure.

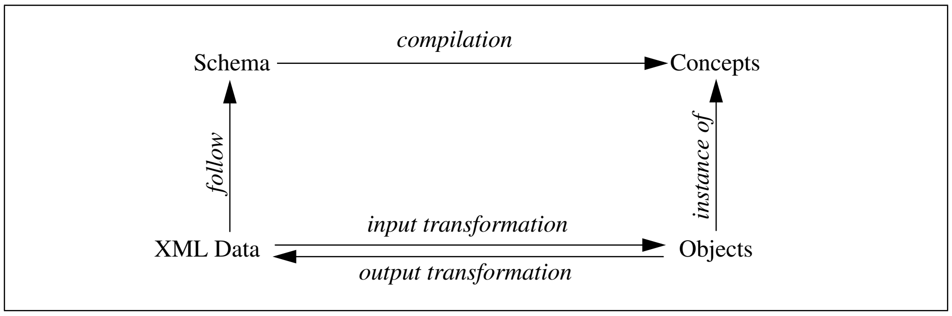

Figure 5: Links between schema, XML data, classes and objects (adapted from [16]).

```
concept Wine {
// attributes
   Name name;
   Year year;
   Productor productor;
   Price[] price;
   ...

// methods
  ...
}
```

In order to provide an automatic input and output of XML data into objects classes, the instantiation methods of these classes can be defined. The principle is to traverse depth-first the tree corresponding to the XML data and to call successively the constructors corresponding to the nodes. The use of the DOM is essential here because it gives directly access to the nodes. Following this principle, a compiler has been defined (see [16]) that generates class definitions and access methods in Java, starting from XML structures.

The modular programming approach allows the use of the most adapted tool or language for a given task, e.g. using an object-oriented language.

4.2 Linking objects and raw data: an example

Let us consider the two preceding wine databases (`Wine` and `French Wine`). These two databases give general and more specific characteristics on the wines.

The `Wine` database with general characteristics provides the basis for the wine hierarchy in an OBRS (see Fig. 2). It can be now possible to exploit the wine hierarchy to guide the access to data in the `French Wine` database. Indeed, each concept of the wine hierarchy can be associated, thanks to its properties, to instances of the `French Wine` database, using for example a query language, as those existing in the XML environment.
The query can automatically be built from the properties defining each concept of the wine hierarchy. Building this query requires two steps :

1. The first one associates to a given concept `C` all the concepts which are the most specific, i.e. the leaf concepts of the sub-hierarchy rooted in `C`: the data in the `French wine` database correspond to the leaf concepts in the concept hierarchy while the data in the `Wine` database correspond to the intermediate concepts in the concept hierarchy. For example, `Sylvaner`, `Riesling` and `Tokay pinot gris` are associated to the `Alsacian-wine` concept,

2. The second step retrieves the corresponding instances of these specific concepts in the `French Wine` database. For `Alsacian-wine`, it will provide the individuals `Sylvaner` from `1996` and `Riesling Cuvée Particulière` from `1997`.

5 Conclusion

In this position paper, we have briefly presented a work in progress on the role of XML as a abridge between SSD and OBRS. A system based on this idea is currently under development. There are still many points to be clarified and many ways to be explored. However, the basic ideas of this work seem to be valid, coherent and promising, in particular to carry out a research work on the intelligent manipulation of the WEB data and the design of intelligent information systems.

Acknowledgement

This research work is supported by the INRIA ARC action ESCRIRE and is done in collaboration with INRIA Rhône-Alpes and INRIA Sophia-Antipolis.

References

[1] S. Abiteboul. Querying Semi-Structured Data. In F. Afrati and P. Kolaitis, editors, *Database Theory – ICDT'97, 6th International Conference, Delphi, Greece*, Lecture Notes in Artificial Intelligence 1186, pages 1–18. Springer, Berlin, 1997.

[2] V. Apparao, M. Champion, A. Le Hors, T. Pixley, J. Robie, P. Sharpe, C. Wilson, and L. Wood. Document object model (DOM) – level 2 – specification version 1.0, 1999. http://www.w3.org/TR/WD-DOM-Level-2.

[3] S. Bergamaschi, S. Castano, M. Vincini, and D. Beneventano. Intelligent techniques for the extraction and integration of heterogeneous information. In *IJCAI-99 Workshop on Intelligent Information Integration*, Stockholm, July 1999.

[4] E. Bertino, G. Guerrini, I. Merlo, and M. Mesiti. An approach to classify semi-structured objects. In R. Guerraoui, editor, *Proceedings of ECOOP'99, Lisboa (Portugal)*, Lecture Notes in Computer Sciences 1628, pages 416–440. Springer, Berlin, 1999.

[5] P.V. Biron and A. Malhotra. XML-schema part 2 : Datatypes, 2000. http://www.w3.org/TR/2000/WD-xmlschema-2-20000225/.

[6] A. Borgida and D.L. McGuinness. Asking Queries about Frames. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts*, pages 340–349, 1996.

[7] T. Bray, J. Paoli, and C.M. Sperberg-McQueen. Extensible markup language (XML), version 1.0, 1998. http://www.w3.org/TR/1998/REC-xml-19980210.

[8] M. Buchheit, M.A. Jeusfeld, W. Nutt, and M. Staudt. Subsumption Between Queries To Object-Oriented Databases. *Information Systems*, 19(1):33–54, 1994.

[9] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description Logics Framework for Information Integration. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy*, pages 2–13, 1998.

[10] R. Cover. The XML Cover pages. http://www.oasis-open.org/cover/, 2000.

[11] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML (Submission to W3C), 1998. http://www.w3.org/TR/1998/NOTE-xml-ql-19980819.

[12] J. Ducloy, J.-C. Lamirel, and E. Nauer. A workbench for bibliographic or factual data handling. In *The Information Revolution : Impact on Science and Technology, 14th CODATA International Conference on Computer Aided Systems and Communication*, pages 63–70, Chambéry, France, 1994. CODATA, Springer-Verlag.

[13] J. Euzenat. XML est-il le langage de représentation de connaissance de l'an 2000 ? In *Actes du Colloque Langages et Modèles à Objets (LMO'2K), Mont Saint-Hilaire, Canada*, 2000.

[14] N. Jardine and R. Sibson. *Mathematical Taxonomy*. John Wiley & Sons, London, 1971.

[15] B. Meyer. *Object-oriented Software Construction*, chapter 2, pages 12–18. Prentice Hall International (UK) Ltd., 1988.

[16] M. Reinhold. An XML data-binding facility for the java platform, 1999. http://java.sun.com/xml/docs/bind.pdf.

[17] J. Robie, J. Lapp, and D. Schach. XML Query Language (XQL), 1998. http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[18] A. Simon and A. Napoli. Building Viewpoints in an Object-Based Representation System for Knowledge Discovery in Databases. In *1st International Conference on Information Reuse and Integration, IRI-99*, 1999.

[19] R.R. Sokal and P.H.A. Sneath. *Principles of Numerical Taxonomy.* Freeman, San Francisco (CA), USA, 1963.

[20] H.S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML schema part 1: Structures, 2000. http://www.w3.org/TR/2000/WD-xmlschema-1-20000225/.