International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:4, 2009

A Materialized Approach to the Integration of XML Documents: the OSIX System

H. Ahmad, S. Kermanshahani, A. Simonet, and M. Simonet

***Abstract*—**The data exchanged on the Web are of different nature from those treated by the classical database management systems; these data are called semi-structured data since they do not have a regular and static structure like data found in a relational database; their schema is dynamic and may contain missing data or types. Therefore, the needs for developing further techniques and algorithms to exploit and integrate such data, and extract relevant information for the user have been raised. In this paper we present the system OSIX (Osiris based System for Integration of XML Sources). This system has a Data Warehouse model designed for the integration of semi-structured data and more precisely for the integration of XML documents. The architecture of OSIX relies on the Osiris system, a DL-based model designed for the representation and management of databases and knowledge bases. Osiris is a view-based data model whose indexing system supports semantic query optimization. We show that the problem of query processing on a XML source is optimized by the indexing approach proposed by Osiris.

***Keywords*—**Data integration, semi-structured data, views, XML.

I. Introduction

IN the recent years, the number of data distributed in multiple information sources as well as the number of potential users that use these information sources has been continually increasing. These sources are often heterogeneous. In other words they use different models for the representation of data such as the relational model, semi-structured models on the Web, text files, etc. Accordingly, medical data related to a patient are found in several kinds of support. Beside this multiplicity of supports and formats, the semantics of the various data sources is also heterogeneous, which reflects the diversity of the points of view of the system designers. As a consequence, the languages used for programming or querying these data sources are different.

This raises severe problems to the users who try to combine - or "integrate" - information from various data sources.

In this context, companies have to meet two challenges in order to ensure the quality of their data: the fast availability of the inter-sources information and the discovery of tendencies from the data stored in the company over time. These challenges have led companies to realize the integration of data, which means proposing a more global homogeneous and coherent vision of their data.

Data integration approaches are classified into two main approaches. The first one is the mediator approach [16], where the integration of data is based on the exploitation of abstracted views describing the contents of the various data sources. The data items are not stored at the mediator level and they are accessible only from the original data sources. The second one is the Data Warehouse approach [17], where a Data Warehouse contains a selective extraction of the relevant information stored in diverse sources. After the construction of the Data Warehouse, the user can formulate his queries over a single database, the Data Warehouse. The Data Warehouse (also called materialized) approach is well adapted when the local sources are frequently modified and the query response time must be fast, whereas the mediation approach is preferable if the modifications in the local sources are frequent [9].

In this article we present the OSIX system (Osiris-based System for Integration of XML documents). This system is a materialized framework for XML data integration. This framework is based on the P-type model, implemented by the Osiris system. The P-type model allows defining a collection of real world objects which have the same semantic and susceptible to be perceived according to several points of view, called views, which satisfy various categories of users. We also present the OSIXT (OSIX Tool) which allows querying a XML data source. This tool uses the object-based indexing model of the Osiris system in order to allow the optimization of the query processing over the XML source.

II. XML Data Integration

Because of the simplicity and the flexibility of XML, it plays a growing role in the publication of data on the Web. However, contrary to structured data, which are data with a regular structure which can consequently be easily stored in relational tables, semi-structured data, and XML in particular, do not have a priori defined schema. When used, the model of underlying data sources can be seen as a relaxation of the traditional relational model, in which a less rigid and less homogeneous structure of the "attributes" is permitted.

The semi-structured model is very useful to represent different kind of documents: multi-media, hypertextual, scientific data, etc. As a consequence to the popularity of XML an enormous amount of XML data of various origins and thus structured in various ways has become available. Heterogeneity is detrimental to the massive exploitation of these data. Indeed, in the absence of specific tools, querying XML data of various origins requires that the different DTDs/schemas and their underlying semantics be known,

Authors are with TIMC-IMAG Laboratory, 38700 La Tronche, France.

which is impossible in practice. Consequently, it is necessary to find techniques and methodologies making it possible to question in a simple and effective manner the enormous amount of XML data.

Several projects have dealt this problem like TSIMMIS [7], MIX [5], Xyleme [1][10], VIMIX [4], XyView [13]. The integration process in these systems is usually based on a view mechanism [6]. This mechanism allows defining an abstract (global) schema which presents a unified view of heterogeneous data sources. There are mainly two approaches to build the global schema of an integration system [8]: The *Global As View* (*GAV*) approach defines the global schema as a collection of views over the sources. On the contrary, with the *Local As View* (*LAV*) approach the global schema is built regardless of sources. The sources are defined as views over the global schema.

Each one of these two approaches presents advantages and disadvantages. Querying data sources throughout the global schema is easier with *GAV*, while adding new sources is easier with *LAV*.

Our approach uses G*AV* as the method for schema integration; we define the Osiris global schema as a subset of views over the sources and then we transform each document in the source satisfying a concrete schema into a document satisfying a global schema. We use the indexing system proposed by Osiris to optimize the query processing over a XML source. This indexing system will make possible the automatic determination of the views which a document satisfies. This way, the search space for the document answering a query can be contracted to the space of the views satisfying the query.

III. OSIRIS

A. P-types and Views

Osiris is a view-based database and knowledge base system where views are similar to concepts defined by logical properties, as in Description Logic approaches [11]. The main concept of the Osiris model is the P-type concept, which supports the specification of viewpoints on a domain [14]. For example, STUDENT and TEACHER can be viewpoints of the P-type PERSON in the university domain, To specify a P-type one first gives its minimal (root) view, then its other views by simple or multiple specialization. When specializing a view new attributes and assertions (logical constraints) may be added. The minimal view is the root of the hierarchy of views of a P-type. Thus, in Osiris a P-type is defined from its views, which are object-preserving [12]. Such a top-down approach is contrary to that of relational systems where views are defined as restrictions of a set of existing relations, and may themselves be used as relations in order to define other views.

The type of a P-type is derived from the views declarations (including the minimal view). The type PERSON contains all the attributes and methods which appear in its views. The domain of an attribute in the type PERSON is the union of its domains in the views where it is declared.

To express that a person may be seen as a student, a teacher, a sportsman, one will create the views PERSON, STUDENT, TEACHER, … as subtypes of the P-type PERSON (see Fig. 1 and Fig. 2). The set of interest of the minimal view PERSON is identical to that of the P-type PERSON. The domain of another view is a subset of the domain of the view it specializes, or of the intersection of the domains of the views it specializes in case of multiple specialization.

In OSIRIS, a P-type is given the name of its minimal view. All the objects of a P-type are models of its minimal view. Access to an object under a viewpoint provides access to the attributes of the viewpoint. Thus accessing an object from the minimal view only provides the attributes of the minimal view while accessing it in the viewpoint of the P-type gives access to the whole set of attributes of the type.

An object belongs to one P-type, for example PERSON, if and only if it satisfies the requirements of its minimal view. This means that its assertions are valid. An object can belong to only one P-type, which means that P-types are disjoint concepts if considered in a DL perspective.

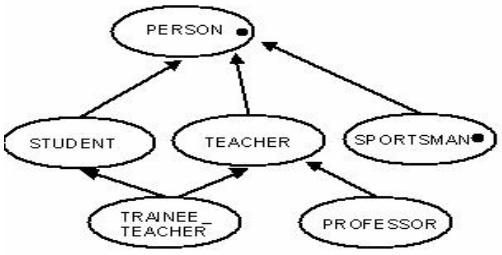

Fig.1 Graph of the P-type PERSON

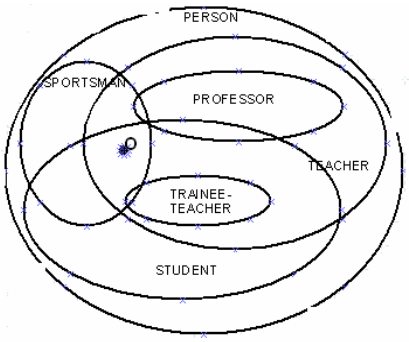

Fig. 2 Inclusion Set of the P-type PERSON

We present the main features of the P-type description language through a very simple OSIRIS example. The modeled universe is that of persons and vehicles. Persons may be STUDENT, TEACHER, TRAINEE-TEACHER, etc, or some of them simultaneously. A given person is a model of the minimal view and may belong to none, any or several other views.

```
type ADDRESS: (street: STRING; postcode: STRING;
                city: STRING);
view PERSON        -- Minimal view of P-type PERSON
attr name : P_NAME;          -- P_NAME is declared elsewhere
  children: setof PERSON ;
  sex : CHAR ;
  age: INT;
  ad_pers: ADDRESS;
  military_Service: STRING;
  incomeTax: REAL calc;    -- procedural attachment
  carsOwned: setof CAR;    -- CAR is a view of a P-type
                              VEHICLE
key  Name                  -- External key; not mandatory
methods
        -- other functions specification
assertions  -- Domain Constraints
  sex in {"f", "m"};
  0 ≤ age ≤120;
  military_Service in {“yes”, “no”, “deferred”, “exempt”};
end PERSON ; -- Note that the minimal view automatically
              contains a private attribute  OID : toid.

view STUDENT: PERSON -- STUDENT specializes PERSON
attr  studies: STRING in {"graduate", "postgraduate",
                           "doctorate"};
  year : INT;
end STUDENT;


view TEACHER : PERSON      -- TEACHER specializes PERSON
attr diplomas : setof STRING in {"degree", "B.A.", "BSc",
                                  "M.A.","MSc","PhD"} ;
  status: STRING in {"trainee", "lecturer", "professor",
                     "instructor", "doctor"};
end TEACHER;

view PROFESSOR: TEACHER --  specializes TEACHER
assertions  Diplomas contain "PhD";
            Status = "professor";
end PROFESSOR;

view TRAINEE-TEACHER: STUDENT, TEACHER
                    --specializes STUDENT and TEACHER
 assertions
  age ≤ 27;
  status = "trainee";
  studies = "graduate";
  diplomas contain "degree";
end TRAINEE-TEACHER;
```

B. Classification Space

Most innovative features of the system come from the use of a classification space, which is distinct from the original set of users' views.

The classification space is a partitioning of the object space into equivalence classes named Eq-classes, according to the relation "have the same truth values according to the (entire set of) Domain Predicates of the type". As a consequence all objects of a given Eq-class are models of the same assertions (Domain Constraints and Inter-Attribute Dependencies) [14].

Construction

In a P-type T, one considers for each attribute Ai the set $P_T(Ai)$ of predicates over Ai which appear in the assertions (Domain Constraints and Inter-Attribute Dependencies) of the views of T. Elementary predicates in these constraints are of the form $Ai \in Dik$ where Dik is a subset of the domain of definition Δi of Ai. A predicate $Ai \in Dik$ defines a partitioning of Δi into two (disjoint) sub domains: Dik and Δi - Dik. The product of all the partitions [16] defined by the predicates of P_T (Ai) constitutes a partition of Δi whose blocks dij, called Stable Sub Domains (SSDs), have the following property:(Stability of an attribute) When the value of an attribute Ai of an object varies within the same stable subdomain dij, Ai continues to satisfy the same set of predicates of P_T (Ai).

Considering the above definition of the P-type PERSON, and considering only the predicates on the attributes *age*, *military_Service* and *sex*, we obtain the following partitioning of the attributes:

SSDs of age: d11 = [0, 18[, d12 = [18, 27],
d13 =] 27, 65[, d14 = [65,140]
SSDs of sex: d21 = {"f"}, d22 = {"m"}
SSDs of military_Service: d31 = {“yes”},
d32 = {"no", “deferred, “exempt”}

This partition can be extended to the space of objects (which is restricted here to the three dimensions considered) and constitutes the classification space of the P-type.

Each element of the classification space is called an Eq-class. It is represented by a tuple with n elements, where n is the number of classifying attributes of the P-type.

Let SSD_{Attr1}, $SSD_{Attr2,}$ … , SSD_{AttrN} be the set of the stable sub domains of the attributes ATTR1, ATTR2, …, ATTRn of the P-type T respectively.

ClassificationSpace $T \subseteq$ {<d1i, d2j, …, dnk> |
d1i ∈ SSD_{Attr1} and d2j ∈ SSD_{Attr2} and …
dnk ∈ SSD_{AttrN}}

Partitioning the object space into Eq-classes is central to the implementation of the Osiris system. Although the actual partition is not represented in its totality (its size is exponential to the number of classifying attributes) it underlies most runtime processes such as object classification, view classification (subsumption), integrity checking and object indexing.

C. Indexing Structure Descriptor

An indexing structure called ISD (Indexing Structure Descriptor) is defined for each P-type [14]. Its main components are:

1) A vector of SSDs representing one or more Eq-classes indexing a set of objects. Two values have been added to represent the unknown and undefined (null) states of an attribute.

2) A vector of views that provides the status (Valid, Invalid or Potential) of each view of the P-type for the set of objects indexed by this ISD.
3) A reference to the actual set of objects of the ISD.
4) The total number of objects indexed by the ISD.

An object, even if only partially known, belongs to one and only one ISD, as an ISD denotes all its possible Eq-classes. The sets of Eq-classes corresponding to different ISDs may not be disjoint in the case of incompletely known objects. Only actual ISDs, i.e., ISDs containing actual objects, are represented. When an unknown attribute becomes known the corresponding object changes its ISD (a new ISD is created if necessary). When an attribute changes its value it remains within the same ISD iff none of its attributes has changed its SSD.

D. Query Evaluation in Osiris

Queries are evaluated in three steps:

1) Determination of the ISDs corresponding to the query when rewritten in terms of the SSDs of its attributes.
2) Determination of the ISDs indexing objects that are valid for the query
3) Projection of the resulting objects onto the attributes of interest of the query

IV. OSIX (Osiris-based System for the Integration of XML Documents)

The architecture of our system OSIX for the integration of XML documents is described in [3] (see Fig. 3). The system OSIX consists of:

1) A description processor, which describes an Osiris schema by an XML schema.
2) A Correspondence processor for every concrete XML schema, which makes the mapping between this concrete schema and the global Osiris schema. This mapping between schemas is a path-to-path mapping [2].
3) A transformation processor, which transforms each XML document in the data source with a concrete schema into a document satisfying the global schema.
4) An extraction processor, which parses each XML document, obtained after transformation, and stores it in a temporary memory in order to extract atomic values.
5) A classification processor, which classifies objects after their extraction.
6) A Data Warehouse, which stores the data extracted from the local sources.
7) A query processor, which uses the information of Stable Sub Domains (SSDs) to obtain the ISDs satisfying the query.
8) An extraction processor which searches the objects which satisfy the query using the corresponding ISDs of step 7 and extracts the relevant data for the query.

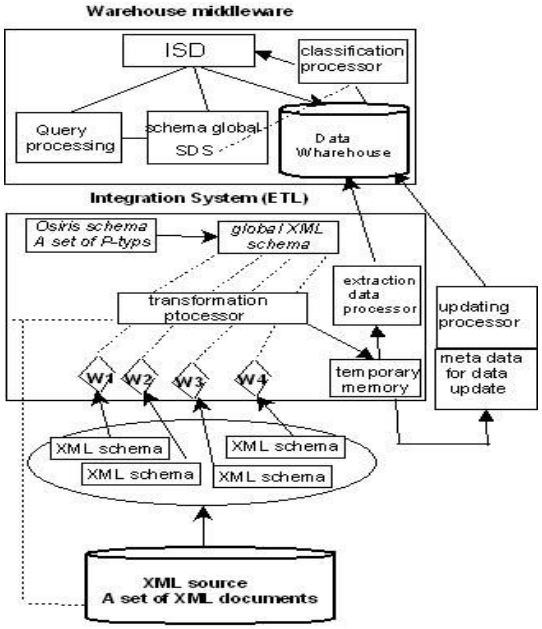

Fig. 3 Architecture of the OSIX System

In the following section we present OSIXT (OSIX Tool), which is the tool implementing the OSIX system. This tool allows querying a XML data source and provides an answer to the query. This tool uses the object-based indexing model of the Osiris system in order to allow the optimization of the query processing over the XML source.

V. The OSIX Tool (OSIXT)

OSIXT (OSIX Tool) allows querying a XML data source by using the indexing model of the Osiris system in order to reduce the response time. The OSIXT tool consists of three essential models:

1) The transformation model: This model allows making the transformation of a group of XML documents satisfying a concrete schema into a group of XML documents which satisfies the Osiris global schema.
2) The extraction model: This model allows parsing XML documents obtained after transformation in order to extract the atomic values (the leaves of the DOM tree of an XML document) and store these values in a dynamic relational table.
3) The querying model: This model allows typing a query over a source of XML documents and giving the appropriate answer.

A. Functional Architecture

The functional architecture of the OSIXT tool is presented in Fig. 4. The system consists of three interfaces which allow making the transformation (“convert” button), the extraction (“Expert’’ button) and the querying of a XML documents source (“Query” button).

The user interface allows to:

1) Enter a set of XML documents already contained in a folder; these documents satisfy a concrete schema, in order to perform the transformation. The user will get as result a folder of XML documents satisfying the global schema.
2) Select a folder containing XML documents after transformation in order to execute the "Extraction'' model. The result of this step is a relational table containing the atomic values of XML documents representing the data source.
3) Type a SQL query who will consult the table in order to get the appropriate answer. The query processing uses the classification mechanism of Osiris characterized by use of views and Stable Sub Domains (SSDs) in order to optimize the response time of the query.

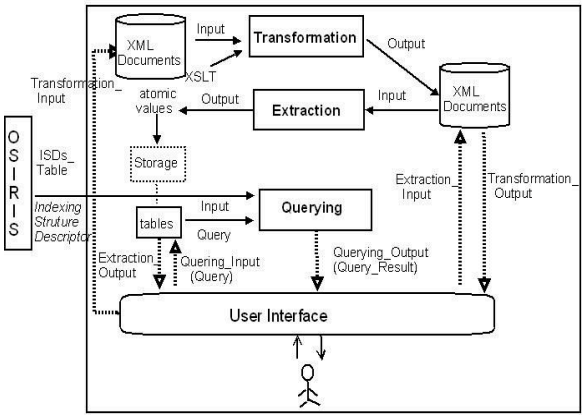

Fig. 4 Functional architecture of OSIXT

In the following section we give a detailed description of the three main modules of OSIXT, their algorithms and their graphic interfaces.

B. OSIXT Models

Transformation Model

To be able to take advantage of the indexation offered by Osiris, The role of this model is to transform every concrete XML document satisfying a concrete schema into a document which satisfies the Osiris global schema. This transformation is made in a semi-automatic manner by using the correspondence information described by a XSLT file. We describe below the algorithm of the "Transformation Model".

Transformation Model Algorithm

```
    TypeDocFile: <doc: Document;
                  pathFile: String;
                  nameFile:String>
transformDirectoryFiles (nameDirectory: String;
nameFileXSLT: String): List(DocFile)
begin
   Result: List (DocFile);
   fileSource: File;
   listeF: List (File);
   listeF ← ListeFile(nameDrectory) ;
for every fileSource in listeF   do
      doc: Document ;
      docF: DocFile ;
      doc ←  parseXMLFile (fileSource. (), nameFileXSLT);
      docF← create DocFile(doc,filePath(), fileSource.path(),
fileSource.name());
      result.add(docF);
endFor
 return (result)
end
```

transformDirectoryFiles is a function of the type List (DocFile) , it has two parameters: the directory name "nameDirectory", containing the set of XML documents , and the XSLT file name "nameFileXSLT" defining the mapping information.

For each source file, the transformation defined in the file "nameFileXSLT" is applied on all source files contained in the directory "nameDirectory". We obtain as result a list of XML files after transformation.

Fig. 5 presents a part of the OSIXT graphic interface called "Convert" which allows converting a directory of XML documents satisfying a local schema into a directory of XML document satisfying the schema global.

The main window is composed of several buttons:

1) The "source" button, which allows selecting the directory name containing the set of XML documents satisfying a local schema.
2) The "XSLT FILE" which allows selecting the XSLT file name which makes the mapping between the local and the global schema.
3) The "target" button which allows specifying the target directory name which will contain the set of XML documents after transformation.
4) The "Convert Source to Target" button is the key button of this interface; it allows making the transformation by using the mapping between the local schema and the global schema. This mapping is described by the selected XSLT file.

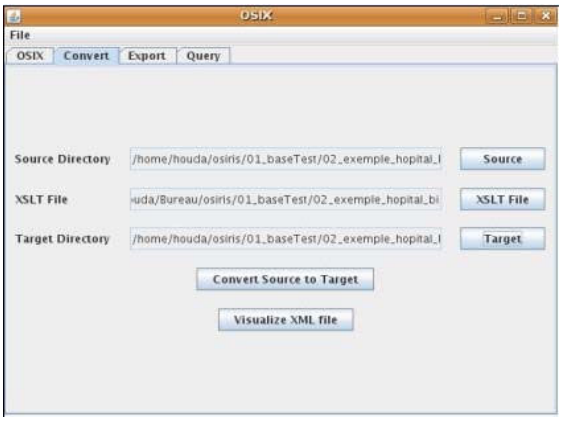

Fig. 5 Main window of the convert processor

1. Extraction Model

Following to the previous processor, all the documents of the source are transformed into documents satisfying the Osiris global schema. The extraction model parses these documents in order to extract the OSIRIS objects and store these objects in the Data Warehouse. We also store the Oids of these objects and the identifier of the document which contains this element in the source. We store this information in a relational table; the schema of this table has the following form:

Data _Store (oid, docID, $attr_1$, $attr_2$, …, $attr_m$) where:

1) Oid is the object identifier in the Osiris system.
2) $attr_1$, $attr_2$, … , $attr_m$ are the classifying Osiris attributes and all the simple elements having atomic values in the source base.
3) docID is the identifier of the document which contains the corresponding element in the source base.

Extraction Model Algorithm

```
Type AttributeValue: < AttributeName: STRING;
                       dfSource: DocFile;
                       value: STRING;
                       Xpath: STRING>
getAttributeValues (docF: DocFile): List (AttributeValue)

Begin
result: List (AttributeValue);
doc: Document; Xpath: String;
doc ← docF.getDocument();
Xpath ← "//*[not (*)]";   (it allows to get all elements without a child element)
NodeList: List (Node);
NodeList ← XPathAPI.selectNodeList(doc, Xpath);

For each node n Do
    name: String;
    value: String;
    name ← n.getName();
    value ← n.getValue();
    path ← getXPath(n, doc);
    Attr: AttributeValue;
    Attr ← create AttributeValue(name, value, path, docf);
    Result.add (attr)
EndFor
return (result);
End
```

The function "getAttributeValue" returns a list of "AttributeValue" where each element is a leaf element of the document object contained in Docfile. Each leaf node is defined by its name, its value, its path and its document name.

Fig. 6 presents a part of the OSIXT graphic interface called "Export" which allows extracting the atomic values of XML files after transformation; it also allows choosing the name of the relational table where the atomic values will be stored.

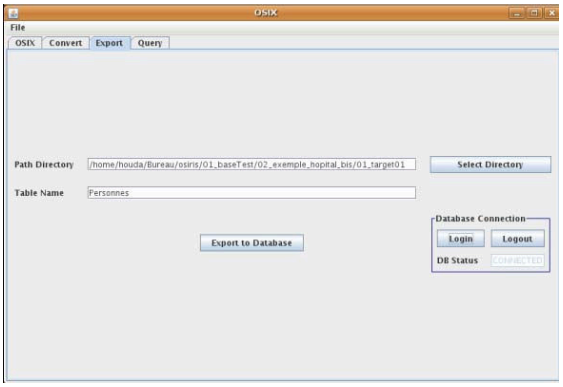

Fig. 6 Main window of the Export processor

The "Export to DataBase" button is the key button of this interface; it allows to extract the atomic values and to store them it in a relational table.

2. The Querying Model

This model allows typing a query over a source of XML documents and giving the appropriate answer. To do that the system uses SSD information in order to evaluate the query as explained in section 3 by performing the following steps:

1) The system rewrites the query in terms of SSDs and determines the potential ISDs corresponding to the query.
2) Among these ISDs, the system determines the ISDs which contain the objects validating the query (valid ISDs) and those which contain the objects that could potentially participate in the response of the query (potential ISDs). In these cases there are other conditions to verify. If an object satisfies these conditions, it participates to the response.

After these two steps, we determine the effective ISDs of the query, i.e., those stored in the ISD space of the system because they have at least a stored object. The valid ISDs give the valid objects for the query, whereas the potential ISDs give the objects for which a supplementary test must be made to determine if it is an actual answer to the query. For the potential objects we define the conditions to verify, so that an object of the ISD satisfies the query.

In the ISD (Indexing Structure Descriptor) of our system, for every object we have a reference towards the corresponding object stored in the Data Warehouse. For the objects validating the query we use their reference to be able to recover the atomic values of the attributes asked in the query. For the possibly valid objects, we need to verify that these objects satisfy the supplementary conditions that are necessary to confirm whether or not the object satisfies the query.

Fig. 7 presents a part of the OSIXT graphic interface called "Query" which allows choosing or writing a SQL query over a XML source and provides the appropriate answer.

The main window is composed of several buttons:

1) The “Load Query File” button allows selecting the Query name that we want to execute. The query text is displayed in the “Query text” filed.
2) The “Execute” button allows executing the selected query and displaying the result in the “Query Result” field.
3) The “chronometer” button allows showing the query execution’s time.

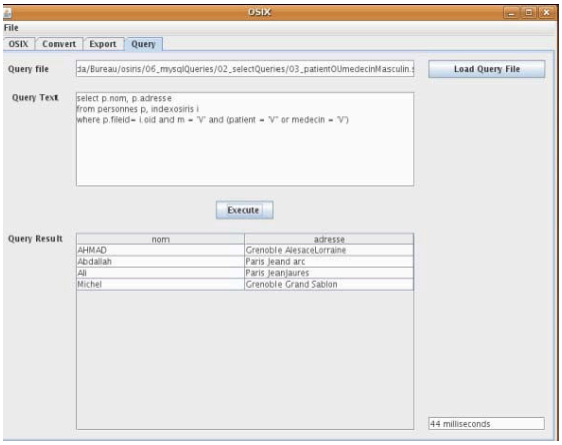

Fig. 7 Main window of the querying processor

VI. Conclusion

In this paper, we have presented the OSIX system (Osiris-based System for the Integration of XML sources). This approach is based mainly on the three following processors: transformation, extraction and querying. Our system uses the ISD (Indexing Structure Descriptor) provided by Osiris in order to achieve semantic query optimization. This indexing makes possible the automatic determination of the views that a document satisfies. This way, the search space for the documents answering the query can be reduced to the ISDs which satisfy the query.

We also have presented the OSIX tool which allows querying a XML data source and provides an answer to the query.

In future work we will focus on the maintenance of the modifications of data sources in our materialized approach.

References

[1] S. Abiteboul, S. Cluet , G. Ferran and M-C. Rousset: “The Xyleme Project”. Gemo Repot 248, INRIA, 2001.
[2] H.Ahmad, S. Kermanshahani, A. Simonet and M. Simonet: “A View-Based Approach to the Integration of Structured and Semi-structured Data”, IEEE International Baltic Conference on Databases and Information Systems-Communication of Baltic DBIS , 2006.
[3] H.Ahmad, S. Kermanshahani, A. Simonet and M. Simonet: “Data Warehouse based Approach to the Integration of Semi-structured Data”, WCMT The 1st International Workshop on Web-based Contents Management Technologies, Suzhou, China 2009
[4] X. Baril: “Un modèle de vues pour l’intégration de sources de données XML: VIMIX ”. PHD thesis, Languedoc University of Science and Techniques, 2003.
[5] C. Bornhovd: “MIX – A Representation Model for the Integration of Web- Based Data”. Technical report, Dep.CS, Darmstadt University of Technology, Germany, 1998.
[6] M. Cannataro, S. Cluet, G. Tradigo, P. Veltri and D. Vodislav:” Using views to query XML. In Encyclopedia of Database Technologies and Applications”, pp.729-735 , 2005.
[7] H. Garcia-Molina: “The TSIMMIS approach to mediation: Data Models and Languages”. Journal of Intelligent Information Systems. 8(2) pp 117-132, 1997.
[8] A. Halevy: “Answering queries using views: A survey”. The VLBD Journal, 10(4), 270-294. 2001.
[9] S. Kermanshahani: “Semi-Materialized Framework: a Hybrid Approach to Data Integration”, CSTST Student Workshop, Paris, October 2008.
[10] I. Manolescu, D. Florescu and D. Kossman: “Answering XML Queries Over Heterogeneous Data Sources”. In proceedings of the 27 th International Conference on VLDB, 2001.
[11] M. Roger, A. Simonet, M. Simonet, "Bringing Together Description Logics and Databases in an Object-Oriented Model", DEXA 2002, Database and Expert System Applications, Toulouse, Sept. 2002.
[12] M. H. Scholl, C. Laasch, M. Tresch, “Updatable Views in Object-Oriented Databases”, Proc. 2nd DOOD conf., pp 187-198, Dec. 1991.
[13] I. Sebi : “Interrogation de Documents XML à Travers des Vues”. PhD thesis, EDITE, CEDRIS Laboratory, 2007.
[14] A. Simonet, M. Simonet, "Classement d’instance et Evaluation des Requêtes en Osiris", in BDA’96 : Bases de Données Avancées, Cassis, France, pp 273-288, Aug. 1996.
[15] D. Stanat, D. McAllister: “Discrete Mathematics in Computer Science”, Prentice Hall, 1977.
[16] G. Wiederhold. “Mediators in the architecture of future information systems”. IEEE Computer Magazine, 25(3), 38-49, 1992.
[17] M.-C. Wu, A. P. Buchmann. “Research issues in data warehousing”. In Datebanksysteme in Buro, Technik and Wissenschaft, pp. 61-82, 1997.