

An Efficient Data Extraction and Storage Utility For XML Documents

Ismailcem Budak Arpinar¹, John Miller², and Amit P. Sheth³

LSDIS Lab
Computer Science Department
University of Georgia
Hardman Hall, Athens, GA – 30602

¹budak@ainge.cs.uga.edu, ²jam@cs.uga.edu, ³amit@cs.uga.edu

Abstract. A flexible filtering technique and data extraction mechanism for XML documents are presented. A relational database schema is created on the fly to store filtered and extracted XML elements and attributes. Building an XML based workflow process repository provides a motivation. Dynamic XML technology combined with Java reflection provides for an efficient traversal method for XML hierarchies to locate the elements/attributes to be filtered.

1 Introduction

In this paper, a mechanism to provide selective extraction of data objects from XML documents, the storage of these documents in a relational database, and retrieval and reconstruction of XML documents from extracted data objects is discussed. The motivation is provided by a need for a Workflow Process Repository in a Workflow Management System (WFMS) [5], namely METEOR WFMS, to store meta-data about workflow designs, organizations, informational resources and computational resources. Thus meta-data is composed of different XML documents representing different components of a workflow process.

The repository, involving the Data Extraction and Storage Utility (i.e., Extractor), has the following main capabilities:

- Filtering of XML objects that need to be extracted,
- Generating relational schemas for on-the-fly storage of XML documents,
- Loading data from XML documents into relational tables,
- Re-creating original XML documents as needed,
- Querying, browsing, and versioning.

Our scheme has superiority over other storage alternatives for XML documents in terms of practicality and flexibility. Practicality arises because of the obvious acceptance and wide use of Relational Database Management Systems (RDBMSs); flexibility is provided by selective extraction mechanism (i.e., filtering) employed by the Extractor, which is not available in similar approaches [3] using a RDBMS. Other approaches, such as XML databases (e.g., Lore [4]), might have superiority over our approach in terms of efficient storage and querying XML documents [1,2].

2 Motivation and Repository Architecture

The motivation behind building an XML based workflow repository is providing for multi-organizational workflow processes, and supporting reusability, adaptability and

survivability of both intra- and inter-organizational workflows. Multiple organizations on the Web can post their services into the repository as workflow steps, and these steps can be incorporated into other organizations' workflow processes using repository's querying and browsing capabilities. Even the organizations can use their own local repositories to reuse existing workflow components, eliminating design of new workflows from scratch. Finally, the survivability is supported by replacing failed workflow components with functionally equivalent components at run-time, thus changing workflow schemas on the fly. These new components are searched in the repository using the graphical query composer and placed into the new workflow schema by a "drag-and-drop".

The workflow repository architecture is depicted in Figure 1. Dynamic XML¹ (DXML) provides ability to access and manipulate XML objects like regular Java objects. This provides a greater flexibility in traversing the XML documents over other methods involving Document Object Model (DOM). The Document Type Definitions (DTDs) are processed to generate classes for each element defined in a DTD. XML objects that need to be extracted and mapping between relational tables/attributes, and XML elements/attributes are defined in a *spec* file. Relational schemas are created dynamically using this specification, and Extractor uses generated classes and Java reflection to "extract" XML objects to be stored in the corresponding relations.

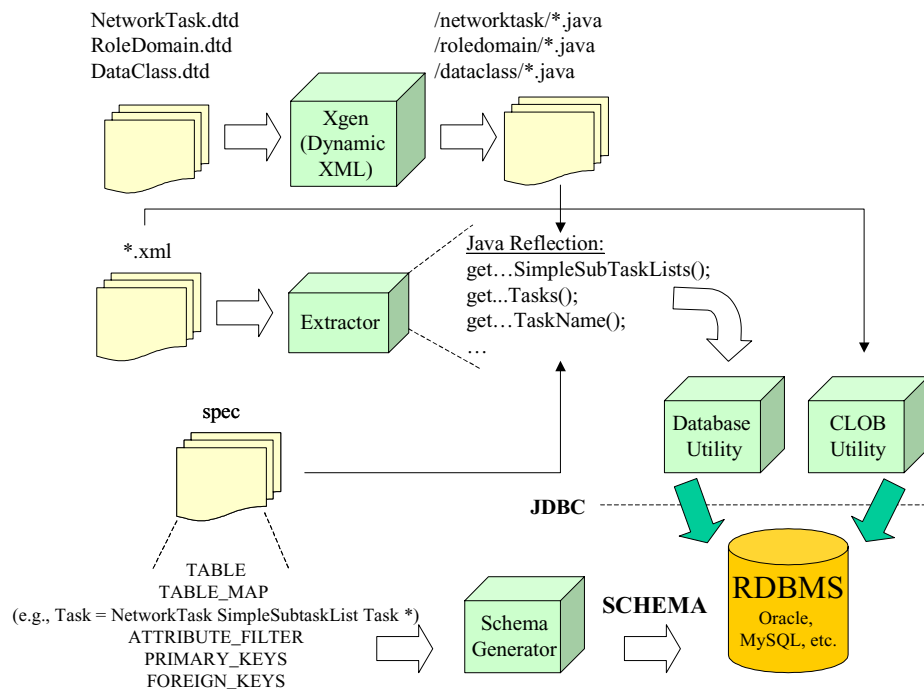


Figure 1. An Overview of Repository Architecture

The XML documents themselves are stored as Character Large Objects (CLOBs)², and the extracted information is correlated with corresponding CLOBs using foreign keys. In this way, extracted information can be queried (or browsed through), and once desired

¹ DXML is a trademark of ObjectSpace, Inc.

² Large Data Object (LOB) or similar data types are used where CLOB is not supported by a RDBMS.

object is located, original XML documents (e.g., workflow schemas) can be reproduced from the database.

3 Data Filtering and Extraction From XML Documents

An XML-relational mapping scheme is used to create a relational schema corresponding the “filtered” hierarchy of an XML document. Actually, both an XML document and a relational database can be viewed as trees. In Figure 2, a tree representation of the XML document below, namely *SampleFlow.xml*³ describing workflow steps (i.e., tasks) is depicted.

```
<?xml version="1.0"?>
<!DOCTYPE NetworkTask SYSTEM "NetworkTask.dtd">

<NetworkTask id="1">
  <Task id="2">
    <Name>SampleFlow</Name>
    <TaskType>Non-transactional Workflow</TaskType>
  </Task>
  <SimpleSubTaskList>
    <Task id="3">
      <Name>Start</Name>
      <TaskType>Human</TaskType>
    </Task>
    <Task id="4">
      <Name>Close</Name>
      <TaskType>Transactional</TaskType>
    </Task>
  </SimpleSubTaskList>
</NetworkTask>
```

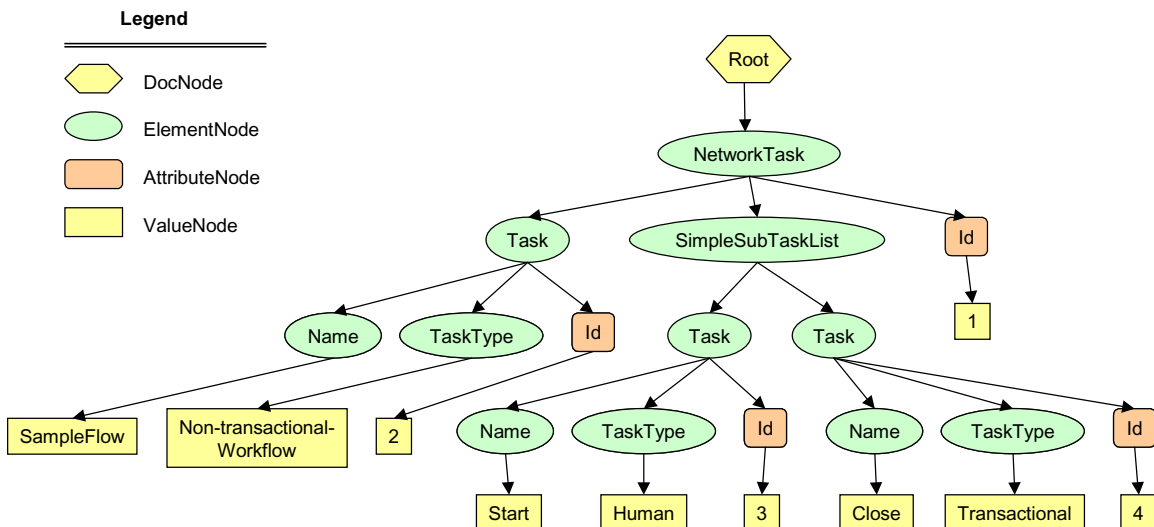


Figure 2. A Tree Representing the XML Document Hierarchy

³ A highly simplified version of the actual XML document is presented here.

Filtering is realized in terms of (1) mapping only “selected” *elements* to *relations*, and (2) mapping only “selected” *elements* or *attributes* to *relational attributes*. To achieve (1), TABLE and TABLE_MAP structures are used to define relations and paths to find corresponding elements in the XML tree respectively.

```
public static final String [ ] TABLE = {
    "xml doc",
    "networktask",
    " task",
    "dataobject",
    "role domain",
    "wfrole"};

public static final String [ ] TABLE_MAP = {
    "", // special root table (no data from
    XML document itself)
    "NetworkTask Task",
    "NetworkTask SimpleSubTaskList Task *",
    "DataClass",
    "RoleDomain",
    "RoleDomain Role *"};
```

For example, the path for a networktask is "NetworkTask Task"⁴ (the orange path in Figure 3), and the path for a task is "NetworkTask SimpleSubtaskList Task *" which is depicted as a turquoise path. Star (*) means task is multivalued. Extractor uses these specifications to recursively traverse XML tree structure from the Root downward using the interfaces and classes generated by DXML. The elements that will be extracted, which are underlined in Figure 3, constitute a filtered subset of elements in the tree and are not necessarily leaf nodes of the paths (e.g., NetworkTask). Note that complete extraction algorithm is not discussed here, because of space limitations.

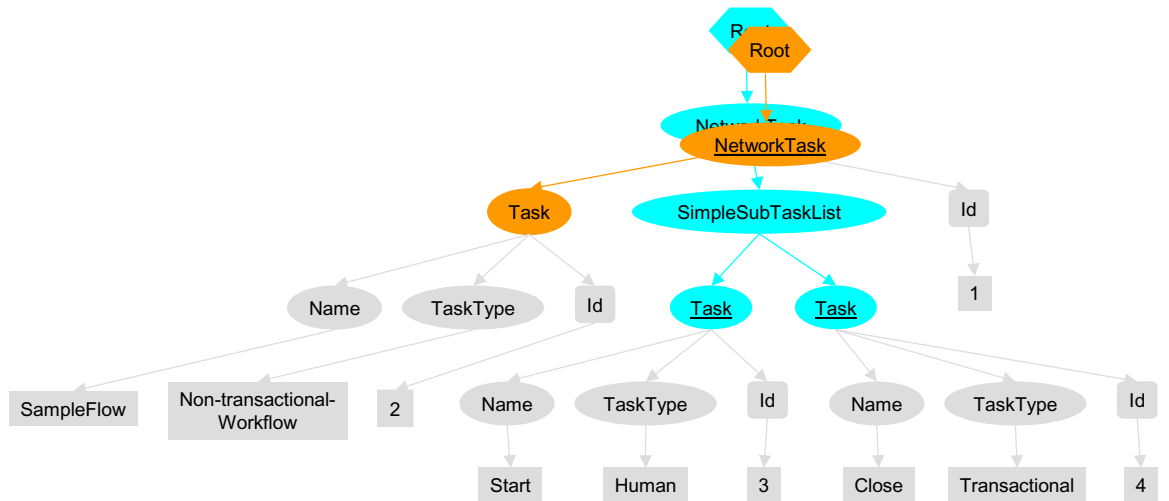


Figure 3. Specification of Tables and Table Map

The property (2) above is realized by filtering only a selected subset of attributes associated with the leaf nodes of the paths. Actually, these paths constitute lattices with a single start node and a single termination node if we keep only one of the multivalued elements. Thus, every attribute/element of a termination node is actually associated with the start node⁴, which represents a relation. For example, Name and TaskType are children of a leaf node (i.e., Task) and constitute filtered elements and attributes for NetworkTask

⁴ The Root can be eliminated.

relation (Figure 4). Notice that these elements/attributes are not associated with their direct parent (i.e., Task), because it is not filtered as a relation at the previous step.

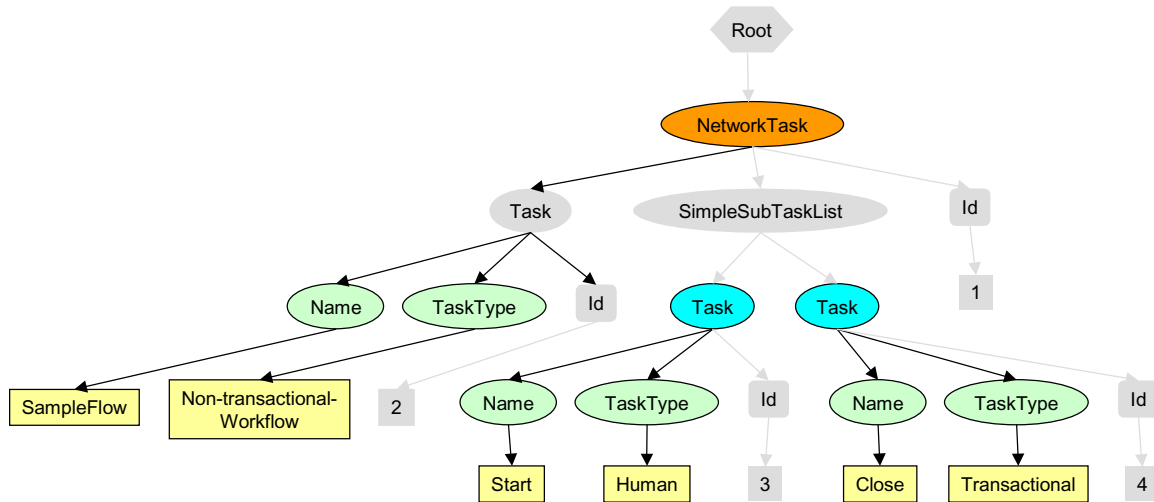


Figure 4. Filtering Elements and Attributes

The employed two filtering techniques ((1)&(2)) allow users to control creation of database schemas and provide an efficient storage mechanism for the selected components, because instead of whole document important parts of the document for a user are specified and extracted.

The resulting relational database is depicted in Figure 5. Filtered elements (i.e., NetworkTask and Task) are mapped to relations, and filtered elements/attributes (e.g., Name and TaskType) are mapped to attributes. A special relation, namely Xml Doc, is used to store XML documents themselves in DocumentText attribute as CLOBs. A VersionNo attribute is placed to support different versions of the same XML document.

Remaining entities that are needed to complete a relational schema are primary and foreign keys. Simply the relation(s) referenced are indicated for each relation, and foreign key(s) are produced to reference that relation(s)'s primary keys. A generated name for a foreign key is "<relation-name>_<primary-key>" of what it references. For example, for Task "Xml Doc_DocumentId" references the XML Document. In general, a filtered XML element in the tree can reference one of the primary keys of a parent element, which is also filtered, or XML document to which it belongs. Finally, the primary keys are specified and they are underlined in Figure 5.

4 Conclusions

Recently, XML gains a great acceptance as a data interchange format on the Web. Thus, providing storage and querying capabilities for XML attains interests of many researches. However, a broadly accepted solution is still missing. We believe that our approach provides for a flexible and practical solution until XML DBMSs are improved and standardized. Furthermore, the XML based workflow repository provides easy exchange of workflow process definitions between companies, and an integration tool to enable coordination of companies' business processes.

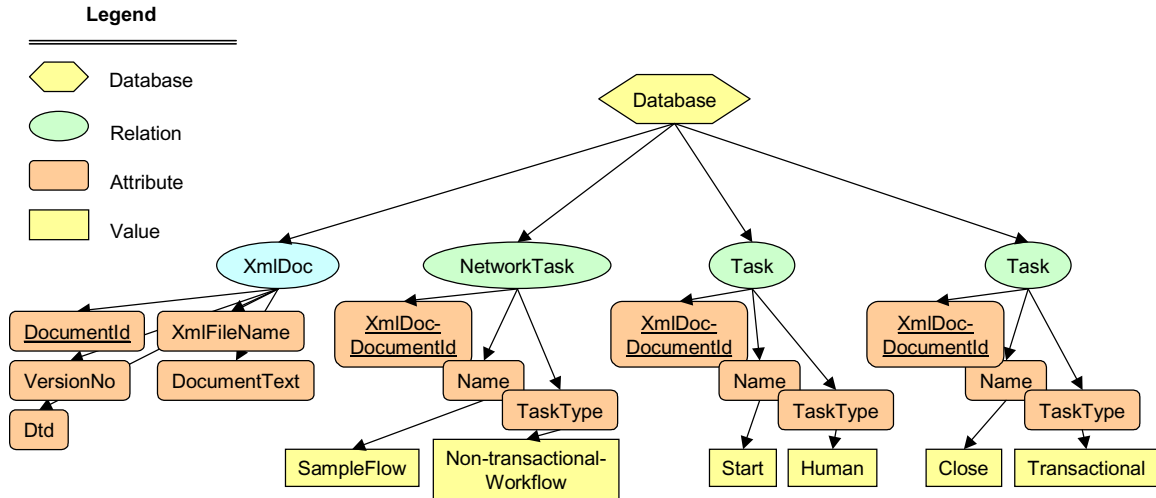


Figure 5. Dynamically Created Relational Database

References

- [1] "The XML Query Algebra", W3C Working Draft, December 2000.
- [2] S. Abiteboul, P. Buneman, and D. Suciu, "Data on the Web", Morgan Kaufmann, 2000.
- [3] R. Bourret, C. Bornhovd, and A. Buchmann, "A Generic Load/Extract Utility for Data Transfer Between XML Documents and Relational Databases", WECWIS 2000, Milpitas, CA.
- [4] L. McHugh, S. Abitebul, R. Goldman, D. Quass, and J. Widom, "Lore: A Database Management System for Semistructured Data", ACM SIGMOD Record, 26(3), 1997.
- [5] A. Sheth, W. Aalst, and B. Arpinar, "Processes Driving the Networked Economy: Process Portals, Process Vortexes, and Dynamically Trading Processes, IEEE Concurrency, July-September 1999.