NO MATTER WHO USES A CORPORATE memory or how it is constructed, information search through that memory should be efficient and effective. In particular, it should adapt to the users’ needs, activities, and work environments. For a document-based corporate memory distributed through the Web, which is our research area, these requirements raise two main questions: How will we describe the documents that will be part of this corporate memory? How can we improve their retrieval on the Web, particularly by taking advantage of knowledge models?

To answer these questions, we’ve developed a strategy that applies the XML (Extensible Markup Language) metalanguage to knowledge management for an enterprise. (In this case, an enterprise is an actual company or a virtual enterprise made of distributed organizations and teams of people who work together online.) We’ve implemented this strategy in the Osirix (ontology-guided search for information retrieval in XML documents) system. Because Osirix performs search based on the semantics of the documents, it can find more relevant documents than classic search engines can.

XML & knowledge management

HTML would seem the natural choice for Web documents because it is the most popular language for them. However, because of HTML’s drawbacks (see the sidebar “From HTML to XML”), we prefer XML for a document- and Web-based corporate memory.

XML, which derives from SGML, defines a standard for document formation. A DTD (document type definition) is a syntactic specification used as a model for XML documents. It contains definitions of the elements and their attached attributes. A document is considered valid if it conforms to its associated DTD.

XML offers major advantages for corporate-memory management, combining the benefits of SGML and the Web:

An information-exchange standard. To facilitate communication and exchange of information or knowledge, a community (that is, a department, an enterprise, a group of companies of the same domain, a company and its related providers and clients, and so on) can use a DTD to define a standard domain-oriented or application-oriented vocabulary. Community members can then express documents or data with the defined XML markups and exchange them using these markups. An enterprise can thus define one or several DTDs for the documents that will constitute its corporate memory.
Documents built from heterogeneous data.
A corporate memory might include heterogeneous information sources such as documents, knowledge bases, relational or object-oriented databases, and case bases. The answer to a user’s request might require fusion of information from some of these sources. The XML format enables document description as well as arbitrary data description. This lets you mix data and documents to build virtual documents issued from several sources. Data might come from a technical database, while text might come from a document management environment.

Multiple views of the same data. XML supports information and knowledge management in a uniquely structured way (that is, one standard format—XML—manages heterogeneous types of data) and enables different types of processing. Knowledge servers retrieve information, and clients present it to users through adapted interfaces. So, the information search system can take users and context into account, presenting different views of the same data—for example, a graphical view, a table of contents, or the actual data. Furthermore, data are loaded into the client (the browser) and can be processed locally; for example, Java applets can process XML data. Hence, XML could represent for data what Java represents for programs: transparent portability through machines and operating systems. The presentation of different views of the corporate memory could greatly assist user-friendly diffusion of the corporate memory among different classes of users.

Document formatting and presentation.
XML has a companion formalism called XSL (Extensible Style Sheet Language) to define the document-oriented presentation format. Because XML separates a document’s logical structure and its presentation, you can define several style sheets for the same XML document. So, XSL can present a document in HTML, PDF, and so on. It can also generate a generic format that can be postprocessed to generate a standard output format. XSL also enables document processing such as sorting, generating tables of contents, creating tables, and reorganizing the document structure.

So, you can use XSL to define several output formats for the same document structure. You can write once and publish many times, from the same source, to different digital and paper-based media. This allows customized presentation of a document. For example, if a corporate memory has several user classes (for instance, managers, engineers, administrative employees, and commercial employees), the presentation of the answer to a user’s request can be adapted to that user.

Hypertext. XML will eventually offer tools to build powerful hypertext documents: XLink (XML Linking Language) and XPointer. XLink will implement the major hypertext functionality found in dedicated tools: links between more than two documents, external links, links with semantics, and so on. With external links, documents can be annotated from the outside without modifying the source. XPointer is a language that enables addressing into documents according to their structure.

Information search. XML facilitates information search because documents are structured and hence can be considered as a database. You can rely on standardized markups to search for information in a structured way. Moreover, the database community is integrating XML with database technology and query languages.

Annotation with metadata. In the spirit of the Resource Description Framework (RDF), SHOE, and OntoBroker, with XML you can annotate a document with metadata that describe the document abstractly and synthetically, according to predefined standard ontologies. (See the “Related work” sidebar for more on SHOE and Ontobroker.)

You can also mark up parts of a text document with tags whose semantics establish a relationship with ontologies and knowledge models. In some cases, the markup can be interpreted as a hypertext link to ontologies. Markup-driven search engines can also exploit it.

Memory management. XML as a structured-document open standard might be a good candidate to facilitate migration to new systems or software over a long time period. With XML, documents exist by themselves, independently of the processing tools. There are no more proprietary-format documents—just XML documents.

From HTML to XML

One major drawback of HTML is that information loses its structure when translated into HTML. Because HTML is a presentation-oriented markup language, information embodied in it is difficult, if not impossible, to process. Information and knowledge servers become overloaded because they have to search information and perform format processing. Furthermore, the servers often answer the same request many times if users request several views on the same data.

HTML has three other main drawbacks. First, it lacks extensibility: it does not let users create their own tags or attributes to parameterize or semantically qualify their data. Second, it lacks structure: it does not support the specification of deep structures needed to represent database schemas or object-oriented hierarchies. Finally, it lacks validation: it does not support language specification that lets applications check imported data’s structural validity.

To eliminate these drawbacks, a working group of the World Wide Web Consortium created XML (Extensible Markup Language) as a standard for creating markup languages. They designed it for distributing structured documents over the Web. It is a kind of “light” SGML (Standard General Markup Language), simplified to meet Web requirements.

Unlike HTML, XML lets users

- extract data from a document,
- define their own tags and attributes,
- define data structures and nest document structures to any complexity level, and
- make applications that validate a document’s structure. Any XML document can contain an optional description of its grammar for use by applications that perform structural validation.

References


Related work

Several research projects have attempted to exploit the power of ontologies for both knowledge management and Web searches.

Knowledge management

Two types of ontologies that could be useful for knowledge management are domain ontologies (for example, a vehicle ontology for an automobile manufacturer) and enterprise ontologies for describing an enterprise’s model. An enterprise’s corporate memory could contain both types of ontologies.

Two examples of enterprise ontologies are the Enterprise Ontology and the TOVE (Toronto Virtual Enterprise) ontologies. The Enterprise Ontology, proposed by the University of Edinburgh’s Artificial Intelligence Applications Institute (www.ai.ed.ac.uk/project/enterprise/enterprise/ontology.html), has five sections: activity, organization, strategy, time, and marketing. The TOVE ontologies, proposed as part of the University of Toronto’s TOVE project (www.ie.utoronto.ca/ EIL/tove/toveont.html), comprise five core ontologies: activity, organization, product, resource, and service.

Web searches

Ontobroker offers an extension of HTML to annotate documents with information relating to a particular ontology, to guide search by this ontology. In addition, Ontobroker enables representation of ontologies and offers formalisms for request formulation. It consists of a request interface, a search engine (Ontocrawler), and an inference engine. Ontobroker’s authors have used its current version on an ontology for the knowledge acquisition community.

The SHOE (Simple HTML Ontology Extension) search tool employs an extension of HTML grammar to let users create and use ontologies. SHOE aims at both simple textual arrangement and semantic organization of documents. It has an inference engine.

WebCokace is a server of CommonKADS expertise models. Starting from a user’s request, it delivers expertise models and associated documents over the Web. WebCokace offers hypertext navigation within an expertise model, information search within an expertise model through a query language, and navigation between expertise models and documents. An expertise model’s entities are annotated with links to documents, and vice versa. So, a knowledge model guides the search for document-level information.

Osirix

Some aspects of our Osirix system (see the main article) are comparable to research on ontology-guided information search on the Web. Like WebCokace, Osirix relies on the CommonKADS method and CML. The main differences between Osirix and WebCokace, Ontobroker, or SHOE are that Osirix uses XML (Extensible Markup Language) instead of HTML and does not use axioms. As with classic search engines, information search in Osirix is keyword-based, but Osirix also offers semantics-based information search. However, Osirix searches are less powerful than those permitted by XML-QL (XML Query Language). After Osirix undergoes enterprise wide evaluation, we could extend it to enable more complex requests and to integrate new search criteria that might be needed in a real enterprise.

References


3. V.R. Benjamins, D. Fensel, and A. Gomez Perez, “Knowledge Management through Onto-


Implementing Osirix

Our main objective is to enable information search in Web documents, guided by knowledge models. The search’s result should include only relevant answers—that is, Web documents that correspond semantically to the request. The knowledge models guiding the search will be CommonKADS expertise models. Figure 1 shows the global architecture for the Osirix search process.

Our implementation of Osirix involves two main phases: creation of annotated XML documents and information search through those documents.

Creating annotated XML documents. This phase consists of six steps:

1. Build and implement an ontology in the enterprise.

2. Define the core DTD.

3. Generate an annotation DTD from the ontology.

4. Create the complete DTD.

5. Create ontologically annotated XML documents.

6. Validate the documents.

Figure 2 diagrams this phase.

Step 1: Building the ontology. An ontology for a corporate memory can be an enterprise-developed ontology or one imported from the external world on which the enterprise members agree. In our case, we used the Cokace programming environment to implement in CommonKADS Conceptual Modeling Language (CML) an extension of the Artificial Intelligence Applications Institute’s Enterprise Ontology. (For more on the Enterprise Ontology, see the “Related work” sidebar.) Cokace comprises a set of tools that let us build and verify CML programs—that is, CommonKADS expertise models. We built it above Centaur, which generates a programming environment for a language when given the specifications of that language’s syntax and semantics. For each parsed CML program, Centaur builds an abstract syntax tree. The parsed program is then manipulated through its abstract syntax tree, which enables complex processing.

Steps 2 through 4: Dealing with DTDs. The enterprise members must first agree on the format of the XML documents that will constitute the corporate memory. This leads
Osirix automatically translates the CML expertise models into the annotation DTD. The annotation DTD indicates the (optional) elements that can be used as ontological annotations in the documents. This annotation lets a document have a semantic value enabling its retrieval. If necessary, an annotation DTD can be modified—for example, to simplify the kinds of ontological annotations.

We built Osirix above Cokace and implemented Osirix using both Centaur’s Pretty-Printer Meta-Language tool and an object-inheritance-management mechanism. To pretty-print abstract syntax trees in a readable format, PPML lets the programmer specify pretty-printers. So, Osirix exploits PPML to generate, from the CML program’s abstract syntax tree, adequate elements constituting the annotation DTD.

Steps 5 and 6: Creating and validating documents. At this point, the document author or corporate-memory builder creates new ontologically annotated XML documents or adds ontological annotations to already existing XML documents.

Osirix’s validation parser then checks if the documents follow the syntax specified in the complete DTD. In particular, this parser checks whether the ontological annotations of the documents conform to the DTD specification. Validation ensures that this document can be retrieved for an answer. This validation will also prevent the Osirix filtering engine from jamming during data extraction.

For a validation parser, we chose IBM’s XML for Java, which is one of the most developed validation parsers. XML for Java includes the W3C’s API DOM (Document Object Model), which describes the interface of XML basic objects, and the SAX (Simple API for XML) interface.

Searching for information. To fulfill the user’s requests, Osirix searches the ontological annotations for an answer. If the system does not find exact answers, it can seek

```
concept(*name, con_body(*descr, *super, *prop_list, *axioms)) ->
  [v]
  [v]  "<!ELEMENT" *name "(" inhslotvrg(*name) *prop_list ")">"
  [v]  "<!ATTLIST" *name "name_id" "ID #IMPLIED""
  def_child::*prop_list;
```

Figure 3. Part of the translator of a CML concept into a part of a DTD.
Information search follows these steps:

1. The user makes a request.
2. Osirix sends the request to a search engine.
3. The search engine searches for appropriate XML documents.
4. Osirix produces the URLs of the most likely candidate documents.

Figure 5 diagrams this phase; Figure 6 details a typical scenario.

**Step 1: Making a request.** Initially, the user expresses the request in a browser such as Netscape or Explorer, using the Osirix query interface. Requests have two basic types. In a general request, the user constrains no attribute value—for example, “Find the pages of all the projects of the enterprise.” In a specific request, the user constrains some attribute values—for example, “Find the pages of the project named GENIE.” Users can submit a hybrid request consisting of several basic requests—for example, “Find all the deliverables of the project named GENIE.” Osirix will decompose the hybrid request into basic requests.
To focus a search on only a few sites, the user can specify a URL that he or she considers to be the root of the URLs of the sites to be browsed. For a company’s intranet, this is quite realistic. However, for a virtual enterprise, such a root URL will probably not exist. In this case, the user needs to indicate explicitly the URLs of the sites of the organizations in the virtual enterprise.

When a user expresses a request, the Osirix query interface lists the concepts of the corporate memory’s ontology. Figure 7 shows the query interface for the request, “Find the participants of the project named GENIE.”

**Step 2: Sending the request.** When the user clicks on the Search button to submit the information he or she entered in the query form, Osirix launches a Java servlet (see the “Servlets” sidebar for more information). The servlet recovers the information in the query form and sends the requests to IBM’s Xcentral, a classic search engine, which accepts a logical AND among the request’s keywords.

**Step 3: Searching for documents.** The search engine visits the specified sites, identifies the documents that follow an XML format, and seeks the occurrences of the keywords in the user’s request. For the hybrid request “Find all the deliverables of the project named GENIE,” the engine will search for the keywords “project,” “name,” and “deliverable” and find XML documents where all these keywords occur. The servlet puts URLs of these candidate documents in a vector for the Osirix ontological filtering engine to process later.

**Step 4: Producing the answers.** Osirix’s ontological filtering engine processes the documents sent back by the search tool. It identifies only those documents that have the appropriate ontological information and returns their URLs. For example, for the hybrid request mentioned previously, Osirix will find documents having ontological information such as in Figure 8a.

To distinguish which candidate documents are really the answers, the ontological filtering engine identifies the XML elements in an XML document. It then determines the semantic presence of a keyword (concept or property) that corresponds to the request in that document. Semantic presence means that the keyword is a tag in the document’s ontological part. Ontological information can be regarded as meta-information and need not be visible through a browser.

For a general request, the engine determines whether the keyword is semantically present in the document, without taking into account its value. For a specific request, the engine determines whether the keyword and its value indicated in the request are semantically present.

To gather information from the XML documents, the engine uses SAX. SAX sends back events to the ontological filtering engine each time that it meets an element, an attribute, a document, and so on. The event type depends on the data type SAX encounters. SAX requires a parser, so we chose IBM’s parser.

We integrated SAX in an Osirix servlet.

---

**Servlets**

A servlet is a Java program that uses additional modules belonging to the API of Java servlets. It executes on a Web server inside a Java-compatible server. It enables extension of server functions through an environment offering query–answer services through the Web. A client sends a request to the server, and the server transmits to the servlet the information related to the request. The servlet then creates an answer that the server sends back to the client.

During the answer’s creation, a servlet can exploit all the Java functions. To create the answer and, if needed, save the information related to the request–answer interaction, the servlet can also communicate with external resources such as files, databases, and other applications. The answer sent to the client can thus be a dynamic answer designed for a particular interaction and not an existing static HTML page.

Servlets have several advantages. They are independent of the operating system and the Web server, and they rely on the standard Java language. Moreover, because they remain in memory after they are first called, they execute more quickly.
that calls the IBM parser and the SAX package's predefined methods. The ontological filtering engine tests if the names of the elements indicated in the request appear among the events emitted by SAX. The ontological filtering engine then sends the candidate documents' URLs to the servlet.

For example, for the request “Find all the deliverables of the project named GENIE,” suppose that an XML document contains the information in Figure 8a. Figure 8b shows the SAX-generated events. The ontological filtering engine follows these steps:

1. While analyzing the XML document, it determines if SAX emitted the event **Start element: project**. The answer is yes.
2. It tests whether it encounters the events **Start element: name**, **Characters: GENIE**, and **End element: name**. The answer is yes (because SAX generated these events for the XML element `<name> GENIE </name>`).
3. Then the engine tests whether it encounters **Start element: deliverable**. The answer is yes. The value after the event **Characters: GENIE** constitutes the answer about the deliverables the user requested.
4. The engine sends back this value as answer, along with the document's URL.

The servlet uses hypertext links to display selected XML documents in the user's browser. Because no browser supports XML, documents must be formatted in HTML to enable their display. So, we used XSL to specify the style sheets for presentation of the XML documents in HTML.

```xml
<annotation>
  <project>
    <name> GENIE </name>
    <starting-time>1, January 1995 </starting-time>
    <end-time>30, June 1996 </end-time>
    <company> INRIA </company>
    <company> Dassault-Aviation </company>
    <deliverable> Rapport final du projet GENIE, Thème 3, Lot L3.3.2.1 </deliverable>
  </project>
</annotation>
```

(a)

Start element: annotation
Start element: project
Start element: name
Characters: GENIE
End element: name
Start element: starting-time
Characters: 1, January 1995
End element: starting-time
Start element: end-time
Characters: 30, June 1996
End element: end-time
Start element: company
Characters: INRIA
End element: company
Start element: company
Characters: Dassault-Aviation
End element: company
Start element: deliverable
Characters: Rapport final du projet GENIE, Thème 3, Lot L3.3.2.1
End element: deliverable
End element: project
End element: annotation

(b)

**YOU MIGHT WONDER WHETHER**

the administrative overhead for supporting the structure imposed by XML outweighs XML's advantages. XML-dedicated editors will let corporate-memory builders build and edit "friendly" XML documents (in the same way HTML editors avoid programming directly in HTML). So, handling XML documents instead of HTML documents will impose no additional charge.

Moreover, a company usually has standardized formats for its documents (internal documents or documents published outside). Also, some companies routinely handle SGML documents. So, conforming to a given XML format would not require tremendous effort. Moreover, companies usually have a department responsible for updating the corporate memory.

The requirement for ontologically annotated documents would seem to add an extra layer of complexity to Osirix. However, because XML's philosophy encourages a document's author to adhere to an associated DTD (if one is imposed), inserting annotations will just be similar to adhering to the complete DTD. The authors or a dedicated department can be responsible for annotations. For example, company documentalist could perform this work, which is similar to their usual task of indexing documents to facilitate their retrieval. Automatic indexing resulting from automatic generation of the ontological annotations would simplify keeping the annotations consistent with the ontology and documents. It would also enable regeneration of the annotations when the ontology or the documents evolve. But such automatic indexing seems utopian at present.

Once XML browsers are available, we will integrate Osirix in them. Also, now that the W3C has officially recommended RDF's final version, we will exploit PPML to build an automatic translator of a CML ontology into an RDF schema. We will use the same principle as for Osirix translation of CML into an annotation DTD. Finally, we will evaluate Osirix in an actual company.

**Acknowledgments**

This article is an extension of a paper published at EKAW '99, with the gracious permission of Springer-Verlag.10

Figure 8. (a) Part of an XML document; (b) the SAX-generated events for that part of the document, for the request “Find all the deliverables of the project named GENIE.”
References


Auguste Rabarijona was a trainee in the Acacia Project at Inria while working on his DEA degree at the University of Nice, Sophia Antipolis.

Rose Dieng’s biography appears on page 17.

Olivier Corby is a researcher on the Acacia project at Inria. His main research interest is software environments for knowledge modeling. He helped develop the Smec software system shell. He also developed NMS, a blackboard system, and Cokace, an environment dedicated to CommonKADS. He received his PhD in computer science at the University of Nice, Sophia Antipolis. Contact him at Inria, ACACIA Project, 2004 Route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex, France; olivier.corby@inria.fr.

Rajae Ouaddari was a trainee in the Acacia Project at Inria as part of her studies at the EMI engineering school in Morocco.