Ontology driven websites with Topic Maps TUTORIAL

José Carlos Ramalho[1], Giovani Rubert Librelotto[1⋆], and Pedro Rangel Henriques[1]

University of Minho, Computer Science Department
4710-057, Braga, Portugal
{jcr, grl, prh}@di.uminho.pt

Abstract. Website development has always been an hard task: it consumes time and resources.
What is new today is normally taken as granted tomorrow by users. This is to say that users always want more. Today they want up to date information and they want to access it according to their point of view or particular preferences.
To cope with these demands, websites must be dynamic and must be able to reconfigure automatically their structure, content and appearance.
This scenery has favored the creation of tools for website automatic generation and management.
In this tutorial we propose not a new tool of this kind but a new approach to the problem.
In our approach we consider two layers. A physical layer that we call the resources layer, composed by databases, XML documents, directory subtrees, and the whole sort of files you can think of to represent your information. A metadata layer called the ontology layer, that provides a view to those resources. We will be using this metadata layer to navigate, query and access information resources.
This tutorial has the following structure: it starts by giving an overview of XML and related technologies (XML is used in every component of our architecture); than it details the available XML languages to specify ontologies (in this case web ontologies): RDF, OWL, DAML; after these introductory topics it starts focusing on XTM, showing how one can use it to create web ontologies; at this point the attendee will be prepared to discuss the optimization of Topic Map specification; a component for automatic Topic Map creation will be presented and the attendee will be taught to use it with the case studies; the tutorial ends introducing two navigational components: one is static – it generates a set of static web pages, the other is dynamic – in each moment the website presented to the user is dynamically created from the ontology specification document.

1 Introduction

Everyday thousands of new information resources are linked to the web. This way the web is growing very fast what makes search tasks more difficult. To

⋆ Sponsored by CNPq grants 200339/01-0 - Brazil

solve the problem several initiatives were undertaken and a new area of research and development emerged: the one called Semantic Web.

When we refer to the semantic web we are thinking about a network of concepts. Each concept has a group of related resources and can be related to other resources, thought we can use this concept network (also called ontology) to navigate among web resources or simply among information resources.

The main idea behind this tutorial is to integrate the specification of these concept networks or ontologies with the navigation and their storage. We intend to do this using XML technology in every component of the system.

The idea of using XML to specify ontologies is not new and several markup languages have been developed. From the undertaken initiatives one became an ISO standard: Topic Map ISO 13250[17]. However there are other languages that should be considered like: RDF [18], RDFS [19], DAML [5], OIL [4] or OWL [20]. The most used ones are Topic Map (the XML version – XTM, is being widely used and became recently an appendix to the ISO standard) and RDF. People using them normally have the same goals in mind but they differ in their basic philosophy: XTM follows a top-down approach while RDF works in some sort of a bottom-up approach.

When one is developing a website thinks in a top-down perspective. So the XTM approach seems the best approach to specify an ontology from which the website will be generated.

There are many tools that use ontologies to derive websites or to navigate through a set of resources. Those tools are normally implemented with a traditional programming language like JAVA or PYTHON and work dynamically. Considering the dynamic version we achieved the same purpose with an XML transformation inside a CGI script ("Common Gateway Interface"). The idea behind our navigational component is quite simple: we have defined a view over a point of the ontology; that view is composed by a series of information data-sets; some of those data-sets are hyperlinks; each hyperlink is a CGI call; each time the CGI is called it just presents to the user the same ontology but positioned in another point (the one passed as a parameter). This way to have a website running we just need our CGI and an ontology specified in XTM. However this approach is not suitable for large information systems because the times to calculate the different views tend to grow. For these cases we developed another version, a static one. This static version takes the ontology and generates all the static HTML pages corresponding to all possible views represented in the ontology.

The documentation supporting this tutorial is structured as follows. In the next section we give a brief overview of the technical issues necessary to understand the rest of the tutorial: XML, CGI, and XML transformations. Section 3 presents the basic concepts in this tutorial: Semantic Web and Ontologies. In section 4 there is a detailed overview about Topic Maps. A brief introduction to the subject is given and some of its characteristics are exemplified. At the end of the section a case study is presented to help the reader understand how a Topic Map is created. Section 6 introduces the architecture of the environment

we have developed to work with web ontologies. Afterwards, section 7, discusses the implementation of the navigational component. Finally, section 8 presents two case studies that will be developed during the tutorial with the attendees. The tutorial ends with some conclusions about this approach.

2 XML and structured documents

Structured documents are documents written according to a predictable set of rules. There is a high probability that the majority of documents produced inside an organization are, or are intended to be, structured documents. That is, these documents are written according to a set of structural and organizational rules.

Standard Generalized Markup Language (SGML) [9] is an international standard that was created to provide us with a method for defining the rules of document structure in a way that is both interchangeable and enforceable. XML (eXtended Markup Language) is its descendant and it corresponds to a simpler and easier to process version.

Documents encoded in XML are encoded in a neutral format that can be interchanged with other organizations. In addition, XML enables us to create a sort of structural specification for the content and structure of our documents. This means that every software product that claims to be XML compliant can either impose the rules of structure or ensure that those rules were strictly followed.

SGML and its new and simplified version named "eXtensible Markup Language" (XML), are spreading quite fast. XML is used by groups as diverse as legal and financial publishers, US military, chemists, computer scientists, historians, and aircraft industry. Today, XML is the best-known vendor-independent format to represent document content and structure.

In this context, a document does not have a physical meaning as a set of printed paper sheets or a file. A document is an hierarchical structure, that has an element clearly identified as the root element and its child nodes. For example, a book has an element BOOK as the root element, this element has child nodes called CHAPTERS, each chapter ...

Elements are distinguished from each-other by tags that are inserted in the text. This kind of markup is called DESCRIPTIVE MARKUP. This way, an XML document has two kinds of information: content and markup. The second gives us information about the structure.

Example 1 (XML document).

This example shows how a text can be structured with descriptive markup (tags like BOOK, TOC, TITLE ..., are written between angle brackets) to document production and maintenance.

```
<?xml version="1.0"?>
<BOOK type="science">
  <TITLE>Ramalho's recipes</TITLE>
  <TOC>
```

```
  <SECTION>
    <TITLE>General Procedures</TITLE>
    <PARA>To go through this section you will need...
    </PARA>
    ...
  </SECTION>
  ...
</BOOK>
```

Although only one is presented in the example, an XML document has two parts:

the document instance this is the document marked up with the tags and following the structure of the respective DTD (Document Type Definition - set of rules that should be followed when creating a document). Example 1 shows an instance. A document instance contains the raw data plus the tags and optionally a reference to the DTD if it is not present at beginning of the instance. Note that an instance first line should always be the XML declaration stating that the document is in fact an XML document.

the DTD the Document Type Declaration [12] is the central piece of an XML system. Normally is a file containing the structural specification of a certain kind of documents.

In a DTD, documents are structured in terms of elements. An element may have subelements that also may have subelements and so on. An element also has attributes used to qualify it.

An element is marked with two tags, one marking the beginning and the other marking the end. This markup inside a document completely describes its structure showing what are elements are there and in what order.

In the next example we present a sample DTD that could be the DTD for the document instance shown in the previous example 1.

Example 2 (a simple DTD). Here is the DTD for the document instance presented above:

```
<!-- This DTD defines the structure for books. -->
<!-- Its scope is this tutorial...-->
<!DOCTYPE BOOK [
<!-- a book has a title, an optional table of contents, -->
<!-- and a sequence of one or more sections             -->
<!ELEMENT BOOK (TITLE, TOC?, SECTION+)>
<!-- title is raw text -->
<!ELEMENT TITLE (#PCDATA)>
<!--toc is empty, an application'll take care of its content-->
<!ELEMENT TOC EMPTY>
<!-- section has title and one or more paragraphs -->
<!ELEMENT SECTION (TITLE, PARA+)>
```

```
<!ELEMENT PARA (#PCDATA)>
<!-- book has an attribute named type that reflects its kind -->
<!ATTLIST BOOK TYPE CDATA>
]>
```

As we can see from the example 2, a DTD is a set of element and attribute declarations. This will allow parsers to validate a document instance against a DTD.

This issue, validity checking, is important and it has been one of the most relevant qualities of XML, together with platform independence and the fact of being a standard.

3 Ontologies

The component that represents and maintains these structured collections is the ontology. An ontology is a logical theory to relate the intended meaning of a formal vocabulary, i.e., it is a binding with a particular conceptualization of a world. The ontology includes structures that allow manipulating terms in a more efficient way; it is useful to the human understanding and validation mechanisms operating on inter-agents communication. The importance of its use is the ability to represent hierarchies of object classes (taxonomies) and their relationships.

In the same area, different ontology definitions and classifications can be found . In Artificial Intelligence, Guarino defines ontology as a logical theory accounting for the intended meaning of a formal vocabulary, i.e. its ontological commitment to a particular conceptualization of the world [7].

In the area of Information Systems, ontology is defined as a set of concepts and terms, that can be used to describe some area of knowledge, or construct a representation of the knowledge [16]. According to Chandrasekaran [3], ontologies are theories of content about the object types, object properties, and relationships between objects that are possible in a domain of specific knowledge.

The ontology's development will continue to provide the construction mechanism of the semantic part of Semantic Web. The model proposed by Berners-Lee[1] has been accepted mainly as representation to the architecture of the Semantic Web.

The development of these mechanisms depends of languages that express the information in a way that machines can understand. The challenge is to provide a language that will enable the manipulation, in an efficient way, of data and rules for queries about these data, and that will allow existing rules in any knowledge representation system to be exported to the Web.

The ontology's development will have to represent an important part of the effort in the development of any application in the future. That way, the development of environments to the construction and manipulation of ontologies is crucial and important. Such environment must be composed by an ontology deposit that can be manipulated by developers, users, and application programs,

[1] Available in http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

allowing the navigation, search and terms reuse. When new terms are added to the ontology, the environment must verify the deposit consistency and act accordingly.

3.1 RDF and RDFS (Resource Description Framework)

RDF – the Resource Description Framework – is a foundation for processing metadata; it provides interoperability between applications that exchange machine-understandable information on the Web. RDF emphasizes facilities to enable automated processing of Web resources. RDF metadata can be used in a variety of application areas; for example: in resource discovery to provide better search engine capabilities; in cataloging for describing the content and content relationships available at a particular web site, page or digital library; by intelligent software agents to facilitate knowledge sharing and exchange; in content rating; in describing collections of pages that represent a single logical "document"; for describing intellectual property rights of Web pages, and in many others. RDF with digital signatures will be key to building the *Web of Trust* for electronic commerce, collaboration, and other applications.

Before we move onto the details, we need to tell you, that a draft document, is a work in progress. The Working Groups working with RDF at the W3C has not yet reached full consensus on all parts of RDF, and is continuing to refine the draft.

What is metadata? Let's start of by talking about Metadata. Metadata is "data about data" or "information describing content." In HTML we have:

```
<meta name="keywords" content="rdf,xml,w3c">
<meta name="description" content="This page is about RDF">
```

These tags show that the keywords in this HTML-file is *rdf*, *xml* and *w3c*. Keywords are often used by search engines such as *AltaVista*, *Excite*, and *Lycos* to index sites.

In the context of RDF, metadata is "data describing web resources". RDF uses XML as the encoding syntax for the metadata. The resources being described by RDF are, in general, anything that can be named via a URI. The broad goal of RDF is to define a mechanism for describing resources that makes no assumptions about a particular application domain, nor defines the semantics of any application domain. The definition of the mechanism should be domain neutral, yet the mechanism should be suitable for describing information about any domain.

Why users want metadata If you are having trouble with "information overload" you should look into metadata, as this will give more control over content. A big problem with HTML, is that there are too many different interfaces to metadata information. On one page, an author might use the following piece of code:

```
<meta name="Author" content="Janus Boye">
```

and some other author, that wanted to display the same information, could instead use:

```
<meta name="AuthorName" content="Janus Boye">
```

This shows some of the current problems, that search engines are facing. There's no current standard. What is really meant by Author? or what is AuthorName? and what's the difference? And also, what Giovani Librelotto, are we talking about? The Giovani Librelotto from Brazil, or some other italian Giovani Librelotto?

Why publishers want metadata If you are a publisher (is not everybody on the internet a publisher), you would want to look into metadata, so that you could provide more information about your content. Today there is no complete and standard way to describe all aspects of website content.

There is currently also much redundancy, where description of site content requires multiple standards and multiple files. The current internet metadata is not widely used by publishers, which perhaps is caused by the lack of extensibility. The systems currently used are proprietary, incompatible, and are not widely supported by software vendors.

Another point that makes metadata interesting for publishers, is that RDF introduces a uniform query capability for resource discovery. This could give a publisher much more information about their competition, without drowning in the *information overload*.

Model, Syntax and Schemes Without trying to get too complicated, we will spend this chapter telling you about the different RDF components. We will finish off with a nice looking example, so please stay with us.

At the core of RDF we have the RDF Data Model for representing named properties and their values. These properties serve both to represent attributes of resources (and in this sense correspond to usual attribute-value pairs) and to represent relationships between resources. The RDF data model is a syntax-independent way of representing RDF expressions.

The RDF Syntax is for expressing and transporting this metadata in a manner that maximizes the interoperability of independently developed web servers and clients. The syntax uses the eXtensible Markup Language (XML).

Last, but not least, RDF Schemas are a collection of information about classes of RDF nodes, including properties and relations. RDF schemas are specified using a declarative representation language influenced by ideas from knowledge representation, e.g., semantic nets, frames, and predicate logic, as well as database schema representation models such as binary relational models, and graph data models.

RDF in itself does not contain any predefined vocabularies for authoring metadata. It is though expected that standard vocabularies will emerge, after

all this is a core requirement for large-scale interoperability. Anyone can design a new vocabulary, the only requirement for using it is that a designating URI is included in the metadata instances using this vocabulary.

Without going further into the core, or starting to talk about Nodes, PropertyTypes, or Triples, we will show you an example instead in figure 1, that illustrate all these things.

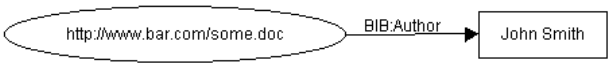

Fig. 1. RDF Triples.

What this picture tells you, is that *Department of Informatics is the Author of the document whose URL is http://www.uminho.pt/di.pdf*. In RDF syntax this would be:

```
<?xml:namespace name="http://uminho.pt/bib-info/" as="BIB"?>
<?xml:namespace name="http://w3.org/TR/WD-rdf-syntax#" as="RDF"?>
<RDF:RDF>
 <RDF:Description RDF:HREF="http://www.uminho.pt/di.pdf">
  <BIB:Author>Department of Informatics</BIB:Author>
 </RDF:Description>
</RDF:RDF>
```

The above syntax represents the named properties, and their values (the Data Model), using the schemas in the 2 first lines, that'll provide you with more information about the different classes (Author and Description).

The first 2 lines tells the user agent, that we will be using the schemas (or vocabularies) from the 2 URL's. The first URL is on our own server, and contains information about the tags, that we have created and added to RDF. The next line is the W3C schema, and it contains the tags, that are recommended by the W3C. The schemas tells the browser what tags are legal, and what they mean, and therefore they are very important. After that, you have the actual RDF code starting. The description line, tells the user agent, that we are describing the document at http://www.uminho.pt/di.pdf. The author line, tells the browser, that Department of Informatics wrote this thing. The last 2 lines closes the RDF, in the same way you would close HTML-code.

Dublin Core Example in RDF One obvious application for RDF is in the description of web pages. This is one of the basic functions of the Dublin Core (DC) initiative. The Dublin Core is a set of 15 metadata elements (such as Title, Subject, Publisher etc.) used to describe resources on the Web. Dublin Core has gathered experts from the library world and the networking and digital library

research communities. Dublin Core is intended to be usable by non-catalogers as well as by those with experience with formal resource description models.

Dublin Core is currently being used in many places, and is one of the foundations that RDF is building on.

We will now show you an example, that uses the RDF syntax to encode Dublin Core metadata within HTML documents.

3.2 DAML+OIL

RDF was developed by the W3C at about the same time as XML, and it turns out to be an excellent complement to XML, providing a language for modeling semi-structured metadata and enabling knowledge-management applications. The RDF core model is successful because of its simplicity. The W3C also developed a purposefully lightweight schema language, RDF Schema (RDFS), to provide basic structures such as classes and properties.

As the ambitions of RDF and XML have expanded to include things like the Semantic Web, the limitations of this lightweight schema language have become evident. Accordingly, a group set out to develop a more expressive schema language, DARPA Agent Markup Language (DAML). Although DAML is not a W3C initiative, several familiar faces from the W3C, including Tim Berners-Lee, participated in its development.

This section series introduces DAML[2] presenting basic DAML concepts and constructs, explaining the most useful modeling tools DAML puts into the designer's hands.

Introducing DAML RDF and provide a basic feature set for information modeling. RDF is very similar to a basic directed graph, which is a very well understood data structure in computer science. This simplicity serves RDF very well, making it a sort of assembly language on top of which almost every other information-modeling method can be overlaid. However, users have desired more from RDF and RDF Schema, including data types, a consistent expression for enumerations, and other facilities. Logicians, some of whom see RDF as a possible tool for developing long-promised practical AI systems, have bemoaned the rather thin set of facilities provided by RDF.

In response the DARPA Agent Markup Language (DAML) [5] sprang from a U.S. government-sponsored effort in August 2000, which released DAML-ONT, a simple language for expressing more sophisticated RDF class definitions than permitted by RDFS. The DAML group soon pooled efforts with the Ontology Inference Layer (OIL) [13], another effort providing more sophisticated classification, using constructs from frame-based AI. The result of these efforts is DAML+OIL, a language for expressing far more sophisticated classifications and properties of resources than RDFS. The most recent release was March 2001 [4], which also adds facilities for data typing based on the type definitions provided in the W3C XML Schema Definition Language (XSDL) [6].

[2] http://www.xml.com/lpt/a/2002/01/30/daml1.html

And the W3C itself is getting into the act. This DAML+OIL specification and its relationship to RDF and RDFS are also available as a series of W3C notes [21], which are used as the base specifications in this article and the following parts. Furthermore, the newly commissioned Web Ontology (WebONT) Working Group [22] has taken on the task of producing an ontology language, with DAML+OIL as its basis.

More Facilities for Properties Let's dive right in by looking at some examples of how DAML+OIL might be used in practice. We'll use an imaginary online sports store, Super Sports Inc. as an example.

RDF Schema allows you to define classes by direct declaration:

```
<rdfs:Class rdf:ID="Product">
  <rdfs:label>Product</rdfs:label>
  <rdfs:comment>An item sold by Super Sports Inc.</rdfs:comment>
</rdfs:Class>
```

You can make similar definitions of properties:

```
<rdfs:Property rdf:ID="productNumber">
  <rdfs:label>Product Number</rdfs:label>
  <rdfs:domain rdf:resource="#Product"/>
  <rdfs:range
  rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdfs:Property>
```

You define instances of these classes by defining resources to be of the relevant RDF type, and then give them relevant properties:

```
<Product rdf:ID="WaterBottle">
  <rdfs:label>Water Bottle</rdfs:label>
  <productNumber>38267</productNumber>
</Product>
```

Data typing, multiple ranges This is all very well, as far as it goes. The problem is that it doesn't go very far. For one thing, based on the name, and the example above, one would reasonably assume that the values of the productNumber property are numbers. We did specify that they must be literals, but literals can be any string, including those we can't interpret as numbers. DAML+OIL allows property values to be restricted to the data types defined in XSDL or to user-defined data types. One does this by using a specialization of RDF properties: DatatypeProperty.

```
<daml:DatatypeProperty rdf:ID="productNumber">
  <rdfs:label>Product Number</rdfs:label>
  <rdfs:domain rdf:resource="#Product"/>
```

```
  <rdfs:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
</daml:DatatypeProperty>
```

We use the "daml" prefix here to represent the DAML+OIL namespace http://www.w3.org/2001/10/daml+oil#. DAML+OIL adds primitives, like DatatypeProperty, as needed, and it defers to RDFS otherwise. There are some changes to the semantics of rdfs:domain and rdfs:range in DAML+OIL systems, the most important of which is that a property can have multiple ranges. Note that DAML+OIL effectively considers every literal used as the value of a property to have some data type. In DAML+OIL, any property that is not a daml:DatatypeProperty is an daml:ObjectProperty, whose range must be a class defined in DAML+OIL or RDF. This provides the sort of flexibility James Clark calls for.

A property can also be defined as identical to another. DAML+OIL provides two ways to express this: with either the daml:equivalentTo or daml:samePropertyAs property. We shall use only the latter because it has the advantage of being subclassed from rdfs:subPropertyOf, which allows some measure of backward compatibility. Suppose that, after Super Sports Inc. has deployed its RDF-based system, an online retailer consortium comes up with a standardized vocabulary for merchandise information. If this new vocabulary uses a property called productID to indicate product number, Super Sports does not have to hack all their code to use the new property. Thanks to DAML+OIL, they can simply extend the definition of productNumber as follows:

```
<rdf:Description about="#productNumber">
  <daml:samePropertyAs
rdf:resource="http://consortium-of-shoppers.org/vocab/productID"/>
</rdf:Description>
```

Broadening the Concept of Class The most important facilities provided by DAML+OIL are those that give designers more expressiveness in classifying resources. The class daml:Class is defined as a subclass of rdfs:Class, and it adds many new facilities.

One example of an added feature in daml:Class is built-in support for enumerations, a very common need in RDF design. An enumeration defines a class by giving an explicit list of its members. So for instance, "US State" is a class with 50 instances. It isn't necessarily hard to come up with a way to express enumerations in RDF. The RDFS spec even hints at the most common approach:

```
<rdfs:Class rdf:ID="MaritalStatus"/>
   <MaritalStatus rdf:ID="Married"/>
   <MaritalStatus rdf:ID="Divorced"/>
   <MaritalStatus rdf:ID="Single"/>
   <MaritalStatus rdf:ID="Widowed"/>
```

Basically, this is just the definition of the instances of the class in the schema. The main problem is that the enumeration is not closed. Anyone can merrily add

items to the enumeration without restriction. In a few cases, this might not be a problem, but usually there needs to be tight control over the items in an enumeration. DAML+OIL allows you to state that a class is defined by being one of a given set of instances.

```
<daml:Class ID="Availability">
  <daml:oneOf parseType="daml:collection">
    <daml:Thing rdf:ID="InStock">
      <rdfs:label>In stock</rdfs:label>
    </daml:Thing>
    <daml:Thing rdf:ID="BackOrdered">
      <rdfs:label>Back ordered</rdfs:label>
    </daml:Thing>
    <daml:Thing rdf:ID="SpecialOrder">
      <rdfs:label>Special order</rdfs:label>
    </daml:Thing>
  </daml:oneOf>
</daml:Class>
```

The daml:oneOf element defines an enumeration, using a DAML+OIL construct that allows us to define closed lists, the daml:collection parse type.

The daml:collection parse type allows a DAML+OIL agent to interpret the body of a property element as a special form of list, which is made up of each of the instances that appear in the element body. This list has a special representation in the RDF model: it is actually a set of statements that recursively breaks the list down into the first element and the sub-list consisting of the rest of the elements (when the last element is reached, the sublist of remaining elements is a special DAML resource daml:nil). Figure 2 illustrates this data structure. This approach should be familiar to functional language programmers, and it has the benefit that you cannot add items to this list without replacing one of the statements that make up the list, unlike RDF containers, to which elements can be added without disrupting the existing list statements.

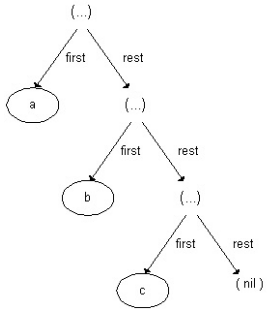

Fig. 2. Data structure.

Each item in the enumeration is defined as an instance of daml:Thing, a special DAML+OIL type which universally includes all instances of all Classes. We don't define each item in the enumeration as an instance of the Availability class directly, because this would be a circular definition.

3.3 OWL (Ontology Web Language)

This section describes the OWL Web Ontology Language. OWL is intended to be used when the information contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be presented to humans. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and their interrelationships is called an ontology. OWL has more facilities for expressing meaning and semantics than XML, RDF, and RDF-S, and thus OWL goes beyond these languages in its ability to represent machine readable content on the Web. OWL is a revision of the DAML+OIL web ontology language incorporating lessons learned from the design and application of DAML+OIL.

Why OWL? The Semantic Web is a vision for the future of the Web in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web. The Semantic Web will build on XML's ability to define customized tagging schemes and RDF's flexible approach to representing data. The first level above RDF required for the Semantic Web is an ontology language what can formally describe the meaning of terminology used in Web documents. If machines are expected to perform useful reasoning tasks on these documents, the language must go beyond the basic semantics of RDF Schema. The OWL Use Cases and Requirements Document provides more details on ontologies, motivates the need for a Web Ontology Language in terms of six use cases, and formulates design goals, requirements and objectives for OWL.

OWL has been designed to meet this need for a Web Ontology Language. OWL is part of the growing stack of W3C recommendations related to the Semantic Web.

- XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.
- XML Schema is a language for restricting the structure of XML documents.
- RDF is a datamodel for objects ("resources") and relations between them, provides a simple semantics for this datamodel, and these datamodels can be represented in an XML syntax.
- RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.

OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

The three sub-languages of OWL OWL provides three increasingly expressive sub-languages designed for use by specific communities of implementers and users.

- OWL Lite supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and OWL Lite provides a quick migration path for thesauri and other taxonomies.
- OWL DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computed) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). OWL DL is so named due to its correspondence with *description logics*, a field of research that has studied the logics that form the formal foundation of OWL.
- OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

Each of these sub-languages is an extension of its simpler predecessor, both in what can be legally expressed and in what can be validly concluded. The following set of relations hold. Their inverses do not.

- Every legal OWL Lite ontology is a legal OWL DL ontology.
- Every legal OWL DL ontology is a legal OWL Full ontology.
- Every valid OWL Lite conclusion is a valid OWL DL conclusion.
- Every valid OWL DL conclusion is a valid OWL Full conclusion.

Ontology developers adopting OWL should consider which sub-language best suits their needs. The choice between OWL Lite and OWL DL depends on the extent to which users require the more-expressive constructs provided by OWL DL and OWL Full. The choice between OWL DL and OWL Full mainly depends on the extent to which users require the meta-modeling facilities of RDF Schema (e.g. defining classes of classes, or attaching properties to classes). When using

OWL Full as compared to OWL DL, reasoning support is less predictable since complete OWL Full implementations do not currently exist.

OWL Full can be viewed as an extension of RDF, while OWL Lite and OWL DL can be viewed as extensions of a restricted view of RDF. Every OWL (Lite, DL, Full) document is an RDF document, and every RDF document is an OWL Full document, but only some RDF documents will be a legal OWL Lite or OWL DL document.

4 XML Topic Maps

A topic map is a formalism to represent knowledge about the structure of an information resource and to organize it in *topics*. These topics have occurrences and associations that represent and define relationships between them. Information about the topics can be inferred by examining the associations and occurrences linked to the topic. A collection of topics and associations is called a topic map.

Topic Maps can be seen as a description of what is about a certain domain, by formally declaring topics, and by linking the relevant parts of the information set to the appropriate topics [17].

A topic map expresses someone's opinion about what the topics are, and which parts of the information set are relevant to which topics. Charles Goldfarb [8] usually compares Topic Maps to a GPS (Global Positioning System) applied to the information universe. Talking about Topic Maps is talking about knowledge structures.

Enabling to create a "virtual map" of information, the information resources stay in its original form and so they are not changed. Then, the same information resource can be used in different ways, for different topic maps. As it is possible and easy to change the map itself, information reuse is achieved.

Topic Map architecture was also designed to allow merging between topic maps without requiring the merged topic maps to be copied or modified.

4.1 The characteristics of Topic Maps model:

Topic *Topics* are the main building blocks of topic maps[14]. In its most generic sense, can be anything. A person, an entity, a concept, really anything regardless of whether it exists or has any other specific characteristic. It constitutes the basis for the topic maps creation. It can be seen as a "multi-headed link, that points to all its occurrences". This "link" aggregates information about a given subject (the thing that the topic is about).

Each topic has a topic type or perhaps multiple topic types. *Topic Type* could be seen like a typical class-instance relationship. Types represent the classes in which topics are grouped in, i.e., the category of one topic instance. Topic types are also topics (by standard definition).

Occurrence A topic can have one or more occurrences. One or more addressable information resources of a given topic, constitutes the set of *Topic Occurrences*. It might be a monograph devoted to a particular topic, for example, or an article about the topic in an encyclopedia; it could be a picture or video describing the topic, a simple mention of the topic in the context of something else, a commentary on the topic (if the topic were a law, say), or any of a lot of other forms in which an information resource might have some relevance to a topic [15]. The occurrences can be an *Universal Resource Identifier* (URI). A topic occurrence represents the information that is specified as relevant to a given subject.

At this point it is very clear the separation in two layers of the topics and their occurrences, one of the great features of Topic Maps. Occurrences establish the routes from the topics to the information resources, enabling also to provide the reason why that route exist. Among all occurrences of a given topic, a distinction can be made among subgroups. Each subgroup is defined by a common role. *Occurrence role* can be used to distinguish graphic from text, main occurrences from ordinary occurrences, mentions from definitions, etc. "The occurrence roles are user-definable and therefore can vary for each topic map" [17]. The standard also defines occurrence roles as topics.

But to make the real distinction between different types of occurrences, Topic Maps uses also the concept of *occurrence role type*. This is different of the occurrence role in the sense that the last one is simply a mnemonic and the first one is a "reference to a topic in the map, which further characterizes the relevance of the role" [15].

Association *Topic associations* are almost ordinary links, except that they are constrained to only relate topics together. Because they are independent of the source documents in which topic occurrences are to be found, they represent a knowledge base, which contains the essence of the information that a someone is creating, and actually represents its essential value. An unlimited number of topics can be associated within "topic associations".

The power of topics maps increases with the creation of topic associations because that way, it is possible to group together a set of topics that are somehow related. This is of great importance in providing intuitive and user-friendly interfaces for navigating large pools of information.

As topic types group different kinds of topics and occurrences roles supports occurrences of different types, associations between topics can also be grouped according to their type (*Association Type*).

It is important to refer that each topic that participates in an association has a corresponding *association role* which states the role played by the topic in the association. Association roles are also regarded as topics in the topic map standard.

4.2 How to define a Topic Map

Before we start, we need to know exactly what we want to represent in the Topic Map. There are two phases to this: delimiting the scope of the TM (that is, deciding the extent of the domain it should cover); and designing the basic ontology. In TM terminology, an ontology is a precise description of the kinds of things that are found in the domain covered: in other words, the set of topics that are used to define classes of topics, associations, roles, and occurrences.

To illustrate all the ideas so far introduced and to describe the TM building process, we will present a case-study where the subject is a workshop. This workshop *XML: Aplicaes e Tecnologias Associadas (XATA)* took place at University of Minho, in February, 2003. Considering this example, we have four topic types: Institution, Paper, Participant, and Session.

In the following examples, we will assume that a participant's name is *Pedro Rangel Henriques*, and that he is *Participant* that belongs to *Department of Informatics*, and he wrote a paper called *Especificao XML de Aplicaes para WWW* that was presented in session *Dialectos XML*. The basic ontology therefore consists of the topic types *Participant*, *Institution*, *Paper*, and *Session*, the association roles *belongs-to/contains*, *is-in-the-session/contains the paper*, and *is-author-of/is-wrote-by*, and the association types *Participant and Institution*, *Paper and Session*, and *Author and Paper*.

First of all, we define the topics themselves (topic types and their instances), specifying their identifiers and base names. Below is an incomplete example:

```
<?xml version="1.0" encoding="UTF-8"?>
<topicMap xmlns="http://www.topicmaps.org/xtm/1.0"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <topic id="Participant">
    <baseName>
      <baseNameString>Participant</baseNameString>
    </baseName>
  </topic>
  <topic id="pedrorangelhenriques">
    <instanceOf>
      <topicRef xlink:href="#Participant"/>
    </instanceOf>
    <baseName>
      <baseNameString>Pedro Rangel Henriques</baseNameString>
    </baseName>
  </topic>
</topicMap>
```

We only show the definition of the topic *Pedro-Rangel-Henriques* and its type *Participant*. The other topics and their types are created in a similar way.

Next we add the occurrence definitions, using the element (*resourceRef*) that contains the URL (resource address) as the value for the *xlink:href* attribute. For instance, one occurrence for the topic *Pedro Rangel Henriques* is specified

in XPath writing the path to the XML file used as the resource, as illustrated below.

```
<topic id="pedrorangelhenriques">
  ...
  <occurrence>
    <instanceOf>
      <topicRef xlink:href="#website"/>
    </instanceOf>
    <resourceRef xlink:href="http://www.di.uminho.pt/~prh"/>
  </occurrence>
</topic>
```

Notice the use of *#xpath* as a *xlink:href*; it is possible because *xpath* is also a topic, more precisely it is a occurrence type.

The "..." in the code above stands for the definition of the topic *Pedro Rangel Henriques* as appeared in the previous specification fragment; we do not repeat it to keep the example as light as possible.

In the third step, we define the associations among topics, stating their type and their members (a topic with an explicit role). In the example below, we define the *Author-and-Paper* association between *Especificao XML de Aplicaes para WWW* and *Pedro Rangel Henriques*. The first one having the role *was-wrote-by* and the second *is-author-of*.

```
<association>
  <instanceOf>
    <topicRef xlink:href="#Author-and-Paper"/>
  </instanceOf>
  <member>
    <roleSpec>
      <topicRef xlink:href="#was-wrote-by"/>
    </roleSpec>
    <topicRef xlink:href="#especificacaoxmldeaplicacoesparawww"/>
  </member>
  <member>
    <roleSpec>
      <topicRef xlink:href="#is-author-of"/>
    </roleSpec>
    <topicRef xlink:href="#pedrorangelhenriques"/>
  </member>
</association>
```

The references *was-wrote-by* and *is-author-of* are association role types, and they are declared like a topic type, i.e., with a identifier and a base-name only. The reference *Author-and-Paper* is an association type, that defines the type of this association. The declaration of this topic is shown below:

```
<topic id="Author-and-Paper">
  <baseName>
    <baseNameString>Author and Paper</baseNameString>
  </baseName>
  <baseName>
    <scope>
      <topicRef xlink:href="#was-wrote-by"/>
    </scope>
    <baseNameString>was wrote by</baseNameString>
  </baseName>
  <baseName>
    <scope>
      <topicRef xlink:href="#is-author-of"/>
    </scope>
    <baseNameString>is author of</baseNameString>
  </baseName>
</topic>
```

The TM above explains that the *Pedro Rangel Henriques* is author of *Especificao XML de Aplicaes para WWW* and, at the same time, indicates that the *Especificao XML de Aplicaes para WWW* was wrote by *Pedro Rangel Henriques*.

The reasons of the use of topic maps are that, first of all, there are too few topic maps-based tools and programs out there to ensure widespread use, and this was first of all our contribution to the distribution process. Second, topic maps is an outstanding technology and standard that bridges several levels of professionals, from technicians to marketing to information architects. Third, a lot of our work is about creating sites and prototypes of various designs and layouts, so to ease the process, we made *DINavigator* aware of relations and contextual menus. To bind all these, topic maps seem the perfect technology.

Also, even if there is a growing amount of tools available, none of them are simply distributed and installed, most of them requiring some other technology, software, library, programming language or utility to work. This is fine for die-hard technologists, but not need a knowledge of how to setup *Tomcat* with *Omnigator*, *TM4J* or *Nexist*, or install a *Python* tool like *SemanText*, or even figure out how to add a plug-in to *Protg 2000*. Hence, *DINavigator* provides an out-of-the-box solution.

The XTM file is where all metadata about your site is stored, and the *DINavigator* framework uses this for all the structuring, relations and so forth you might have. The XTM file is not there for manual editing, but if you do, do note that the next build will overwrite any changes made to it.

5 Metamorphosis architecture

Although we are discussing *DINavigator*, the tool created to navigate among a semantic network. This navigation component is part of a bigger project called *Metamorphosis*.

The idea behind metamorphosis goes a little further than what was discussed so far. To give you a better idea let's pick a real example: suppose you several data resources like XML documents, relational databases or PDF documents; you want all these items to be accessible through the Web; at the first try you have built a complete index of all your data items; that index is huge and you know that there is no browser capable of displaying it; here is the point where you will start discussing the organization of your data resources; and it is here that we introduce you to *Metamorphosis*; you do not need to change anything in your data resources; you just need to create an ontology for your data resources.

Figure 3 further illustrates this idea and gives an idea of *Metamorphosis* architecture.

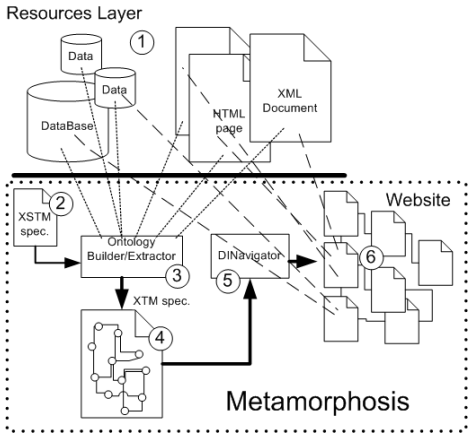

Fig. 3. Metamorphosis

This architecture can be described as follows:

(1) Resources layer This component is composed by our data resources: XML documents, Web pages, Databases, ... Metamorphosis does not interfere with any of it, it will only use part of the information to build the semantic network.

(2) XSTM spec This XML document gives precise information on where to go to extract the relevant information; this component is not being discussed in this paper.

(3) Ontology Builder/Extractor This component uses the XSTM spec and retrieves the information it needs to build the ontology from the data resources. It is implemented as a XSL stylesheet and it takes advantage of information organization: for instance, when processing a relational database it looks for the relations information and automatically generates all the associations between the relevant topics.

(4) XTM spec This is the topic map specification automatically generated by the ontology builder.

(5) DINavigator This is the component being discussed in this paper. Also implemented in XSL. It takes a topic map and produces a whole website according to some principles.
(6) Website The final generated website through which is possible to navigate semantically through the data resources.

In the next section we are going to discuss the navigational component and the website generation.

6 XTM: Automatic Generation

The reasons of the use of topic maps are that, first of all, there are too few topic maps-based tools and programs out there to ensure widespread use, and this was first of all our contribution to the distribution process. Second, topic maps is an outstanding technology and standard that bridges several levels of professionals, from techies to marketing to information architects. Third, a lot of our work is to create sites and prototypes of various designs and layouts, so to ease the process, we made *DInav* aware of relations, contextual menus, and the binding element that is perfect for doing this is topic maps.

Also, even if there is a growing amount of tools available, none of them are simply distributed and installed, most of them requiring some other technology, software, library, programming language or utility to work. This is fine for die-hard technologists, but not need a knowledge of how to setup *Tomcat* with *Omnigator*, *TM4J* or *Nexist*, or install a *Python* tool like *SemanText*, or even figure out how to add a plug-in to *Protg 2000*. Hence, *DInav* provides an out-of-the-box solution.

The XTM file is where all metadata about your site is stored, and the *DInav* framework uses this for all the structuring, relations and so forth you might have. The XTM file is not there for manual editing, but if you do, do note that the next build will overwrite any changes made to it.

6.1 The TM Builder

Looking at a TM we can think of it as having two distinct parts: an ontology and an object catalog. The ontology is defined by what we have been designating as topic type, association type, and occurrence role type. The catalog is composed by a set of information objects that are present in information resources (one object can have multiples occurrences in the information resource) and that are linked to the ontology. Figure 4 gives a schematic representation of this vision.

After creating some Topic Maps by hand, it is easy to conclude that such task is time consuming and very repetitive. Thus gave us the idea to develop an Extractor of topics and relations from a set of XML resources, or by other words, a TM Builder.

In our context, a TM Builder is a converter from one XML language to another XML language. The TM Builder is a XSL stylesheet that receives an

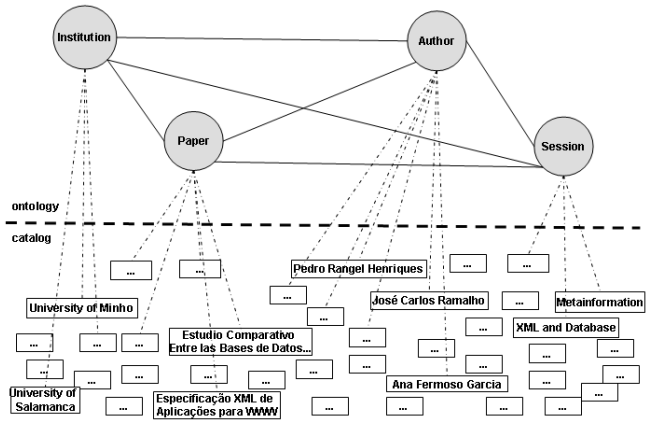

Fig. 4. The case-study's ontology and catalog

XML document as input and generates another XML file that contains a Topic Map. The reasons why the proposed TM Builder accepts XML documents as input are:

- XML is the current language to markup documents;
- XML is becoming the platform for information interchange;
- New data sources (non-XML) can be easily added to our extracting system just by using a translator to XML. Most of the actual information systems, like Database Management Systems, have facilities to dump their information in XML; so, for these cases the front-end is already there.

The main algorithm of the TM Builder to extract a Topic Map from an XML document is: initially, for the given ontology creates all the topics types, occurrences roles, occurrences types, and associations types; after, during a document tree traversal, for each association, define the: association type and association members; and for each element in the source that is seen as a topic, create the: topic ID, topic type, topic names, and topic occurrences. In our system, this algorithm was coded in a XSL stylesheet.

In the next section, we will introduce a language that can be used to specify the extraction process.

6.2 XSTM: specifying XTM extractors

The TM extractor discussed in the last section is tied to the structure of a specific XML source. The creation of a TM for an XML source with a different structure would imply the development of a new extractor. To solve this problem we have created an abstraction layer based on a new XML language: XSTM (XML Specification of Topic Maps).

The XSTM language supplies all the constructors that are needed to specify the extraction task, the Topic Map Builder process; it allows the definition of

topics and their types and occurrences, as well as associations and their types and occurrence roles.

In a more formal way, we show below the CFG (Context Free Grammar) for that language:

XSTM's Context Free Grammar

```
xstm ::= topicType+ topic+ assoc* assocType*
topicType ::= TTypeID InstanceOf TTypeName
topic ::= xpath TTypeID InstanceOf Resource*
Resource ::= resourceData | resourceRef
assocType ::= ATypeID ATypeName MemberAssoc*
MemberAssoc ::= Scope Description
assoc ::= assocClass ATypeID Members*
assocClass ::= "N2N" split=("true"|"false") |
      "one2one" type=("attribute"|"subelement") |
      "one2N" split=("true"|"false") | "all2all"
Members ::= ElemIndex Element*
Element ::= TTopicAssoc RoleID
```

Each XSTM specification is defined as an XML instance and the XSTM language is defined by a DTD and/or an XML-Schema.

Nowadays, XML-Schema has overcome the DTD approach for the definition of classes of the markup documents. We also made that upgrade; however, as XML-Schema is much more verbose than the correspondent DTD, we decided to include here the DTD.

Notice that the XSTM DTD listed above is obtained direct and systematically from the CFG shown.

7 DInav: website automatic generation

At this point we can say that ontologies, specified with XTM, are a set of records, each record represents a concept, it points to some resources (physical information records) and participates in several relations (associations). So, using these relations between concepts to query and navigate through information concepts (first) and then through information resources (secondly) seems the right way to do it.

The main idea about navigation can be described as: when you are positioned at a certain concept the navigation framework shows you a particular concept, the resources it points to, and all the concepts related to it; if you choose one of the related concepts the position changes to that concept and the view will change accordingly; if you choose one of the resources the system will show that resource view.

In terms of implementation we wanted web browsers to be our navigators. This way all the views discussed above would be HTML pages. So, the implementation of this navigational component is reduced to an XML transformation. However, there were two possibilities: runtime or batch transformations.

7.1 Runtime transformations

Runtime transformations were implemented with an XML transformation inside a CGI script. The initial page is a call to the CGI parametrized with the main concept. The CGI applies the transformation to the ontology and produces the HTML view. In this view all the links are CGI calls with a different parameter, the new position inside the ontology.

This is a nice solution because it only needs two files, the ontology (specified in XTM) and the CGI. However, runtime transformations are time consuming. This solution is being used in small prototypes and is still under development.

7.2 Batch transformations

This solution corresponds to the actual component being used in *Metamorphosis*. We have created an XSL stylesheet that transforms the ontology into a website. This website corresponds to all the possible views of the ontology.

Though we can think of *DINavigator* as a website generator, based on XSL and using topic maps for this purpose. It is made to be a simple way of creating full sites, with design, content and topical links. All in all; the words are "simple" and "powerful."

DINavigator was created as a direct consequence of creating a similar framework in XSLT for the XSTM [11] language. When this language was developed, and as the need for a good prototyping-tool emerged at his work, the time was right to look for a way of making all these come together. *DINavigator* was created out of three reasons;

- Be an effective prototyping tool in his work, and lowering the time from prototype to production.
- To create a tool that uses Topic maps, and that is easily distributed, installed and used by all.
- To bridge the technical languages and methods of technicians, programmers, designers and information architects.

DINavigator offers a framework for making rapid changes to a whole site, be it page setup, content, topics, relations or added/deleted pages. Since *DINavigator* is parsed through a standards-compliant XSLT parser, the result, through careful auditing of all accompanying HTML, is production ready. This simply means that you can quickly create prototypes, and simply expand them using the same tool to go into a production environment. This radically decreases development cost and time.

Topic maps is used in *DINavigator* for all structures, relations, content preparation, resources and occurrences, naming and ontological expression. This means that any other topic maps able tool out there can grab the topic map file from *DINavigator* and view its full metadata map.

8 Case Studies

8.1 Conference website specification and generation: XATA 2003

This section presents a case study about the workshop *XML, Aplicaes e Tecnologias Associadas* (*XATA*[3]), that demonstrates the use of TM-Builder to the ontologies extraction from a source XML. This workshop happens at University of Minho, in Braga, Portugal, at February 13-14, 2003. At XATA, the XML portuguese community. Researchers and users of XML from university or companies had participated. Thus allowing a sharing of information between the academic world and the professional world.

In this workshop, many papers are submitted to evaluation; the accepted ones were presented in the conference. The papers presentations were separated in sessions, each one with a associated theme, like *Tecnologia e Web Services* (Technology and Web Services), *XML e Base de Dados* (XML and Database), among others.

The event was all XML based, since its diffusion, until its production. Therefore, all the information referents to the *XATA* are stored in XML documents. The XML-Schema of the workshop is presented in the figure 5. Obviously this XML-Schema is incomplete, but the important to this case study is in this figure.

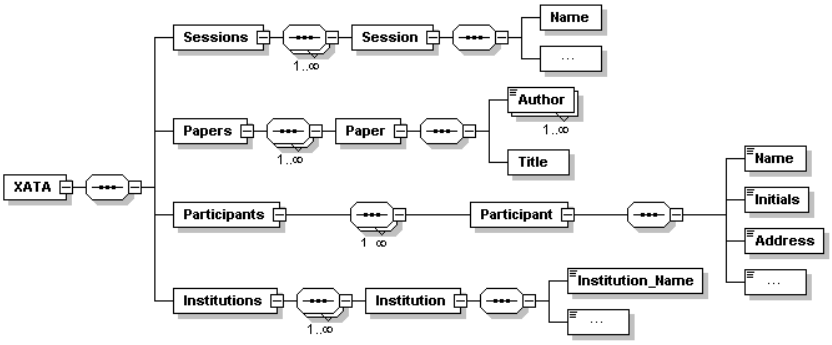

Fig. 5. XATA's XML-Schema.

As the XSTM language only depends of documents structure, and not the XML instance, the ontology specification of XATA can be defined from this XML-Schema. So, there are five steps: definition of the types of topics, of the topics itself, of the occurrence roles, of the association types, and finally, of the associations itself.

The XSTM specification to the XATA The XSTM root element is *xstm*, that has four sub-elements. Each sub-element references a piece of the ontology expressed by XTM. Its sub-elements are: *topicType*, *topic*, *assocType*, and *assoc*.

[3] http://www.di.uminho.pt/ jcr/XML/conferencias/xata2003/

First of all, it are defined the topic type. In this ontology, the topics are grouped in *Institution*, *Author*, *Participant*, *Session*, and *Paper*.

In XSTM, the topic types are declared by *topicType* element, that contains a *id* identifier – to be referenced at other moments in the specification – and a *name* name – to visualization in a XTM browser. For instance, the *Paper* topic type declaration is shown below:

```
<xstm>
  <topicTypes>
    <topicType>
      <id>ID-Paper</id>
      <instanceOf>XATA</instanceOf>
      <name>Paper</name>
    </topicType>
    <topicType>...</topicType>
  </topicTypes>
  ...
</xstm>
```

In XSTM, the topic type are abstract concepts defined by the ontology. The topics are real elements in the input XML documents. The topic element is used to its definition. This element has two sub-elements: the XPath path to the element itself (*xpath*) and its type (*type*). The attributes *id* and *name* mean the XPath path to the topic identifier and to the topic name. The XSTM specification to the topic definition referent to the Paper topic type is presented below:

```
<topics>
  <topic>
    <xpath id="@number" name="Title">//Paper</xpath>
    <type>ID-Paper</type>
  </topic>
  <topic>...</topic>
  ...
<topics>
```

Until here, all the topics and its types are declared. But in XTM, topic without any association has little functionality. The knowledge web is obtained from the associations among topics. Many associations can be inferred from the *XATA*; therefore, the example is the association among *Paper* and *Author* type topics.

Once defined the topics and its types, the next step is the definition of association types. They define the occurrence role for the association members. It is declared with the *assocType* element that has a *id* identifier, a *name* name, and its *membersAssoc* association type.

The *member* element means the member of the association and it has two child. The *scope* element specifies the reference to the topic that is the scope

of this occurrence role. The other one, named *description*, is a name that will be displayed in a Topic Map's application. Each association role will be a new topic, in the output XTM. An *association types* specification in XSTM, between *Author* and *Paper*, can be seen below:

```
<assocTypes>
  <assocType>
    <id>Author-and-Paper</id>
    <name>Author and Paper</name>
    <memberAssoc>
      <scope>is_wrote_by</scope>
      <description>is wrote by</description>
    </memberAssoc>
    <memberAssoc>
      <scope>is_author_of</scope>
      <description>is author of</description>
    </memberAssoc>
  </assocType>
  <assocType>...</assocType>
</assocTypes>
```

To finish the XSTM specification, the *assoc* element allows the specification of all the associations. This associations can involve two or more topics. They are found and extracted from the input XML document.

In the following, when we refer to relationships between tree nodes (XML elements and attributes) we are not talking about relations in the sense of the well-known entity-relationship model. So the usual names 1-to-1, N-to-1, M-to-N, and all-to-all, do not have exactly the same meaning used in that traditional perspective.

In our context, there are four kinds of relationships between elements, that are described by the three alternative children of the *assocs* element:

- associations between an element and its attribute. It is defined by the *one2one* element whose *type* attribute has the value *attribute*;
- associations between an element and a subelement referenced in the element's context. It is defined by the *one2one* element with the *subelement* value in the *type* attribute;
- associations one to N, defined by the *one2N* element;
- associations M to N, defined by the *M2N* element;
- associations between topics that are connected through another table, defined by the *all2all* element.

The *assocs* element contains specification of its type and all its members. The *type* always means the association type (see the previous subsection), i.e., a reference to the identifier of the topic that represents its association type.

The members are a choice of *one2one*, *one2N*, *one2N*, or *all2all*; all of these elements have a similar structure: a sequence of a *type* and *members* elements.

The *members* element is composed by one or more of *member* that defines the members of this association, and it is composed by three elements: the *xpath* means the XPath path to the element (or attribute) that is a member in this association; and the *role* element, that is a occurrence role of this member.

The one2one element expresses relationships that can be obtain from some connection among the topics found in XML document. For instance, in the specific association between *Author* and *Paper*, the authors of each paper can be identified by the content of XPath path *//Paper/Author*, which is a reference to the inicial letters of authors name found in *//Participant/Initial*. Thus, the association among the *Author* and *Paper* topic types, referent to the XATA, was specified in the way shown below:

```
<assocs>
  <one2one type="subelement">
    <type>Author-and-Paper</type>
    <members11>
      <element>
        <topicAssoc ref="Author">Paper</topicAssoc>
        <role>is_wrote_by</role>
      </element>
      <elementRef>
        <topicAssoc ref="Initial">Participant</topicAssoc>
        <role>is_author_of</role>
      </elementRef>
    </members11>
  </one2one>
</assocs>
```

Figure 6, shows the Topic Map visualization within DINavigator. The input topic map was created by TM-Builder. This navigator gives the total access to the extracted ontology, allowing the navigation through the concepts defined in the XSTM specification.

This figure shows the ontology described in XSTM, presents:

- the topic types in the item *Subject Indexes*;
- the associations in the item *Relationship Indexes*;
- the role associations in the item *Role Indexes*;
- the resources types in the item *Resource Indexes*.

8.2 Academic department website specification and generation

It will be developed in the tutorial session.

9 Conclusion

Metamorphosis is being used in several small to medium projects, to build web interfaces to information systems.

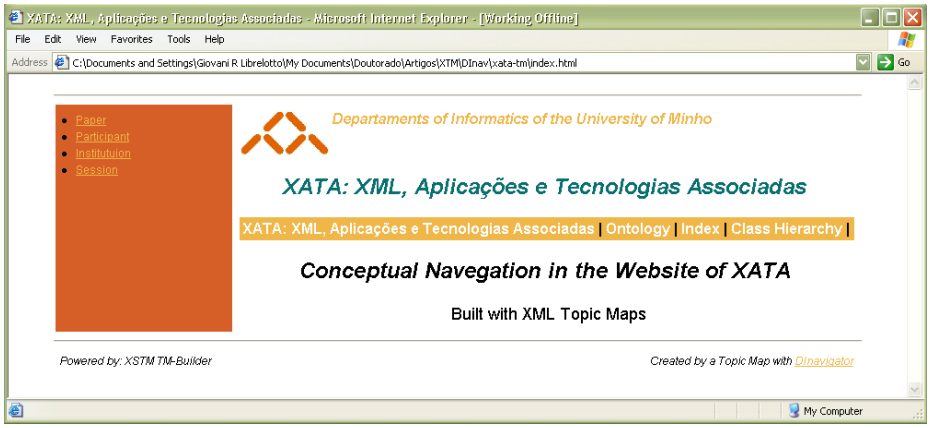

Fig. 6. Topic Map visualization in the DINavigator.

Until now it has proven to be a useful prototyping tool. The web interfaces are created quite easily and fast.

There are some situations where this system has helped a lot:

- suppose you have an heterogeneous information system;
- you want to make it accessible through the web;
- when you start creating your HTML index pages you end up with pages of about 5 Megabytes;
- web browsers can not hold pages above 1 or 2 Megabytes;
- you have a problem ...

We have been solving similar problems creating an ontology for the intended information system and using *Metamorphosis* to manage that interface.

However *Metamorphosis* is not finished:

- we are starting to test it with big XML documents (up to 25 Megabytes);
- we are adapting it to work with relational databases and other sources of information, besides XML documents;
- we are studying a relational model for ontologies (if your ontology grows the XML file would no longer be the proper way to store it);
- we are working in new versions for *Metamorphosis* components enabling them to work with new forms of representing information.

In the moment a web page is being created for *Metamorphosis* and should be announced soon.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. In: Scientific American, in http://www.sciam.com/2001/0501issue/0501berners-lee.html (2001)

2. Biezunsky, M.; Bryan, M., Newcomb, S.: ISO/IEC 13250 - Topic Maps. Available in: http://www.y12.doe.gov/sgml/sc34/document/0129.pdf (1999)
3. Chandrasekaran, B.: What Are Ontologies, and Why do We Need Them?. IEEE Inteligent Systems and their applications, Vol. 14, N. 1. (1999)
4. DARPA: Reference description of the DAML+OIL ontology markup language. Available in http://www.daml.org/2001/03/reference/ (2001)
5. DARPA: DAML. Darpa Agent Markup Language Program. Available in http://www.daml.org/ (2001)
6. Duckett, J., Griffin, O., Mohr, S., Norton, F., Ozu, N., Stokes-Rees, I., Tennison, J., Williams, K., Cagle, K.: Professional XML Schemas. Wrox Press (2001)
7. Guarino, N.: Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration. In: Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology, Springer-Verlag, Berlin Heidelberg New York (1997) 139–170
8. Goldfarb, C.F., Prescod, P.: XML Handbook. Prentice Hall. 4th edn. (2001)
9. Herwijnen, E.: Practical SGML. Kluwer Academic Publishers (1994)
10. Horrocks, I.: DAML+OIL. Available in http://www.daml.org/2001/03/daml+iol-index/ (2001)
11. Librelotto, G., Ramalho, J.C., Henriques, P.R.: XML Topic Map Builder: Specification and Generation. In: XATA: XML, Aplicaes e Tecnologias Associadas (2003)
12. Maler, E., Andaloussi, J.: Developing SGML DTDs: From Text to Model to Markup. Prentice-Hall (1996)
13. OIL: Welcome to OIL. Available in http://www.ontoknowledge.org/oil (2002)
14. Park, J., Hunting, S., Engelbart, D.C.: XML Topic Maps: Creating and Using Topic Maps for the Web. Prentice Hall (2003)
15. Pepper, S.: The TAO of Topic Maps - finding the way in the age of infoglut. Available in http://www.ontopia.net/topicmaps/materials/tao.html (2000)
16. Swatout, W., Tate, A.: Ontologies. In: IEEE Inteligent Systems and their applications. Vol. 14, N. 1 (1999)
17. topicmaps.org Specification: XML Topic Maps (XTM) 1.0. Available in http://www.topicmaps.org/xtm/1.0/ (2001)
18. World Wide Web Consortium: Resource Description Framework (RDF) Model and Syntax Specification. Available in http://www.w3.org/TR/REC-rdf-syntax (1999)
19. World Wide Web Consortium: Resource Description Framework (RDF) Schema Specification 1.0. Available in http://www.w3.org/TR/2000/CR-rdf-schema-20000327/ (2000)
20. World Wide Web Consortium: Web Ontology Language (OWL) Reference Version 1.0. Available in http://www.w3.org/TR/owl-ref/ (2002)
21. World Wide Web Consortium: A series of notes covering DAML+OIL as W3C technical reports. Available in http://www.w3.org/TR/daml+oil-reference and http://www.w3.org/TR/daml+oil-axioms (2001)
22. World Wide Web Consortium: WEBONT: The W3C Web-Ontology (WebOnt) Working Group home page. Available in http://www.w3.org/2001/sw/WebOnt/ (2003)