

## Modeling Relations Between Web Resources

Sabin Corneliu Buraga

Faculty of Computer Science, "A.I.Cuza" University of Iasi, Berthelot Street, 16 Iasi, Romania  
Phone: (+40) 32 201090, E-Mail: busaco@infoiasi.ro, WWW: <http://www.infoiasi.ro/~busaco/>

**Abstract** – *The paper proposes a high-level XML-based model to represent (spatial or temporal) relations between (fragments of) Web sites in order to make possible resource discovery, cataloging or easily describe information. This model will be based on Resource Description Framework (RDF) recommendation of the World-Wide Web Consortium, a general purpose technology that facilitates the description of resources on the Web.*

**Keywords:** *World-Wide Web, XML, Relationship, Data Representation, Distributed File Systems*

### I. INTRODUCTION

We can view the World-Wide Web space as a vast (virtual) distributed file system. A *distributed file system* is a file system whose clients, servers, and storage devices (disks) are dispersed among the interconnected computers of a distributed system [10].

In practice, the concrete configuration and implementation of a distributed file system may differ and it is not easy to decide the best implementation. A distributed file system can be realized as part of a distributed operating system or by a software layer whose key function is to supervise the communication between conventional operating systems and file systems. Some examples of distributed file systems are Sun's *Network File System (NFS)* – build on Remote Procedure Call model, broadly used on UNIX-like systems, Microsoft's *Active Directory* – implemented on Windows 2000/XP, *Prospero* – an Internet-compatible virtual system model based on Uniform Resource Identifiers (URIs), and Coda – an experimental file system developed at Carnegie Mellon University.

The paper proposes a high-level description of a virtual (distributed) file system using *Resource Description Framework (RDF)* – a model for processing metadata. RDF offers interoperability between applications that exchange machine-understandable information on the World-Wide Web.

The proposed RDF model can be applied for a particular distributed file system, not only for the Web. To point up some particular issues we prefer the UNIX file system structure. To state various file properties, we will define an

XML-based language called *Extensible File Properties Markup Language (XFiles language)*. The elements of *XFiles language* will be used to detail RDF statements about the components of a distributed file system or about the relationship between these components, as well as about different resources of a Web site.

Also, the proposed RDF description can be used to make high-level statements about main characteristics of a distributed file system in a consistent and platform- and implementation-independent way.

### II. FILE SYSTEMS

Most visible aspect of an operating system, the file system consists of two distinct parts: the collection of the actual *files*, each collection containing related information, and the *directory structure*, which provides information about all the files in the system. All modern operating systems have an acyclic graph directory scheme of logical file storage [10].

We can view a file like an *abstract data type*. To manipulate this data type we can define a minimal set of file operations (primitives) [5, 10]:

- *open()* is used to open a file. This primitive returns a special value called *handler* to be used in other file operations:  $h = \text{open}(f)$ ,  $f \in \text{FileNames}$ ,  $h \in \text{Handlers}$
- *close()* is used to close a file and to free the associated file handler:  $\text{close}(h)$ ,  $h \in \text{Handlers}$
- *seek()* is used to set the file pointer for the next input/output operation; this primitive gives the possibility to access the file in a sequential or direct manner:  $\text{seek}(h, p)$ ,  $h \in \text{Handlers}$ ,  $p \in \mathbf{Z}$
- *read()* is used to read from a file specific data into a memory buffer:  $\text{read}(h, m)$ ,  $h \in \text{Handlers}$ ,  $m \in \text{Memory}$
- *write()* is used to write to a file specific data stored into a memory buffer:  $\text{write}(h, m)$ ,  $h \in \text{Handlers}$ ,  $m \in \text{Memory}$

Formally, the *Handlers*, *FileNames*, and *Memory* sets are abstract data types. In practice,  $\text{Handlers} \subseteq \mathbf{N}$ ,  $\text{FileNames}$

$\subseteq Chars^+$  and  $Memory \subseteq \mathbf{N}$ , where  $Chars$  is a set of legal filename characters (subset of ASCII or Unicode codes).

In reality, there are many other helpful file primitives. These primitives are frequently implemented by the operating system kernel.

Each particular file system presents a similar interface with the programmer (or user). Each file is represented like an abstract data structure called *vnode* (*virtual information node*). The *vnodes* are data structures used by a virtual file system. For each particular file system – e.g. Linux *ext2*, *ext3* and *proc* file systems, IBM's *High Performance File System (HPFS)*, or Microsoft's *Virtual File Allocation Table (VFAT)* or *Windows NT File System (NTFS)* file systems – or file type, we can derive from the *vnode* class a specific class to denote particular primitives (methods) for dealing with that file system. In the *vnode* class, each operation can be considered as pure virtual from the perspective of object-oriented methodology.

For uniformity, the *vnode* class will be used to represent pipes, devices, pseudo-devices, processes or network sockets, in the same manner. Each special device or communication line is viewed like a file and same primitives can be used to access particular information on that file [10].

In UNIX (particularly Linux), this model is implemented by means of a special data structure called *inode*-operations [10]. Each system resource is considered to be a file and different supported and mounted file systems will be managed by same primitives. For Linux *ext2* file system and for other UNIX file systems, the data structure used to deal with the file information in an indexed manner is called *i-node*.

For distributed file systems, we must consider the *naming* and the *transparency* of files, apart of other characteristics.

Naming is a mapping between logical and physical objects of the network. A significant problem for naming is to discover a proper naming scheme. A practical solution is to connect remote directories to local directories, thus going the appearance of a coherent directory tree structure (i.e. the *mount* protocol in UNIX via NFS). Another approach is to use Uniform Resource Identifiers (URIs) [1, 4] to locate files.

If a distributed file system is transparent, the file location is not essential for a user. This approach leads to the option of file *replication*, a useful redundancy for improving data availability. The replication process is used on Windows 2000/XP systems and on Linux systems – in this second case via *Network Information Service (NIS)*, previously known as Yellow Pages. The main reason of file replication is to offer information that has to be known throughout the network, to all hosts of that network.

In the present and the near future, the operating systems must integrate various Internet services, especially World-Wide Web facilities to remotely access Web files (resources) using file system mechanisms. The general requirements [5, 6] for such distributed and Internet-enabled file systems are:

- scalability,
- support for client/server architecture,
- location-transparent global organization of files,
- on-line administration,
- log-based recovery/restart,
- safe replication,
- security.

### III. RESOURCE DESCRIPTION FRAMEWORK (RDF)

*Resource Description Framework (RDF)* [4, 9] is a standardized basis for processing metadata. The used metadata format should permit to reason about data. The RDF is intended to be used to capture and state the conceptual structure of information offered in the Web. The RDF assertions can be viewed as a data model for describing machine processable semantics of data to build the infrastructure for Berners-Lee's *Semantic Web*.

RDF consists of a model for the representation of named properties and property values. RDF properties may thought of as attributes of resources and in this sense correspond to conventional attribute-value pairs. RDF properties also characterize relationships between resources and therefore a RDF model can resemble an entity-relationship diagram.

To facilitate the definition of metadata, RDF is based on *classes*. A collection of classes, typically designed for a specific purpose or domain, is called a *schema* [2]. Through the sharability of schemas, RDF supports the reusability of metadata definitions. The RDF schemas may themselves be written in RDF.

Figure 1. shows the level where RDF is situated in the context of Web data modeling layers.

The basic model of RDF consists of three object types [9]:

- **Resources**

All objects being described by RDF expressions are called *resources* and they are always named by Uniform Resource Identifiers (*URI*) [1] plus optional anchor identifiers. Using URI schemas (i.e. *http*, *ftp*, or *file* schemas), every type of resource can be identified in the same manner.

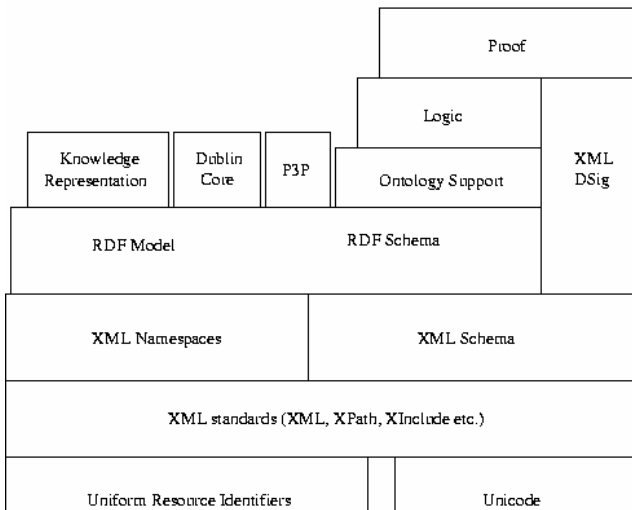


Figure 1. Web data modeling layers

- **Properties**

A *property* is an explicit aspect, characteristic, attribute, or relation to express a resource. Each property has a specific meaning, defines its permitted values, the type of resources it can state, and its relationship with other properties (via RDF Schema).

- **Statements**

A specific resource together with a named property, plus the value of that property for that resource is an RDF *statement*. These three individual parts of a statement are named, respectively, the *subject*, the *predicate*, and the *object*. The object of a statement (e.g., the property value) can be another resource or a literal.

Also, RDF specifies three types of container objects:

- *Bag* – an unordered list of resources or literals,
- *Sequence* – an ordered list of resources or literals,
- *Alternative* – a list of resources or literals that represent alternatives for the single value of a property.

The containers may be defined by an URI pattern. RDF can also be used to make statements about other RDF statements (higher-order statements).

The RDF data model provides an abstract, conceptual framework for defining and using metadata. Currently, there are several proposals of model-theoretic semantics for RDF and RDF Schema [7, 8].

The concrete RDF syntax is based on Extensible Markup Language (XML) [3] elements and attributes. The EBNF grammar of the RDF/XML constructs can be found in [4, 9].

#### IV. RDF DESCRIPTION OF DISTRIBUTED FILE SYSTEMS

To represent RDF statements about various file characteristics we have to define first an XML-based language used to store file properties, called *Extensible File Properties Markup Language (XFiles language)* [5, 6].

##### *An XML schema for XFiles language*

For validation and parsing purposes, we will present an XML schema for this language. An XML schema provides a formal specification of a grammar for an XML language by using XML syntax.

```
<?xml version="1.0" ?>
<Schema name="FileSchema">
  <!-- File type -->
  <ElementType name="Type">
    <AttributeType name="mime" />
    <attribute type="mime" />
  </ElementType>
  <!-- File location -->
  <ElementType name="Location">
    <AttributeType name="dns" />
    <attribute type="dns" />
  </ElementType>
  <!-- Authentication method -->
  <ElementType name="Auth" />
  <!-- Login name of file owner -->
  <ElementType name="Login">
    <AttributeType name="uid" type="int" />
    <attribute type="uid" />
  </ElementType>
  <!-- Group of file owner -->
  <ElementType name="Group">
    <AttributeType name="gid" />
    <attribute type="gid" type="int" />
  </ElementType>
  <!-- Password of file owner -->
  <ElementType name="Password">
    content="textOnly" />
  <!-- Real name of file owner -->
  <ElementType name="Name" />
  <!-- Owner information -->
  <ElementType name="Owner" order="many">
    <element type="Login" maxOccurs="1" />
    <element type="Group" maxOccurs="*" />
    <element type="Password" maxOccurs="1" />
    <element type="Name" maxOccurs="1" />
  </ElementType>
  <!-- File size -->
  <ElementType name="Size">
    content="textOnly">
    <AttributeType name="max" />
    <attribute type="max" type="int" />
  </ElementType>
  <!-- File permission -->
  <ElementType name="Permission">
    <AttributeType name="preserved"
      type="enumeration"
      values="on off" />
  </ElementType>
</Schema>
```

```

        default="on" />
    <attribute type="preserved" />
    <AttributeType name="inherited"
        type="enumeration"
        values="on off"
        default="on" />
    <attribute type="inherited" />
</ElementType>
<!-- File permissions set -->
<ElementType name="Permissions"
    order="many"
    content="eltOnly">
    <element type="Permission" maxOccurs="*" />
</ElementType>
<!-- File timestamp -->
<ElementType name="Timestamp">
    <AttributeType name="type"
        required="yes"
        type="enumeration"
        values="access
            modification
            change" />
    <attribute type="type" />
</ElementType>
<!-- File version
    (used in CVS environments) -->
<ElementType name="Version" content="mixed" />
<!-- File parser (associated application) -->
<ElementType name="Parser">
    <Attribute name="params" />
    <attribute type="params" />
</ElementType>
<!-- File properties
    (document root element) -->
<ElementType name="Properties" order="many">
    <element type="Type"
        minOccurs="0" maxOccurs="1" />
    <element type="Location"
        minOccurs="0" maxOccurs="1" />
    <element type="Auth"
        minOccurs="0" maxOccurs="*" />
    <element type="Owner"
        minOccurs="0" maxOccurs="*" />
    <element type="Size"
        minOccurs="0" maxOccurs="1" />
    <element type="Permissions"
        minOccurs="0" maxOccurs="1" />
    <element type="Timestamp"
        minOccurs="0" maxOccurs="*" />
    <element type="Version"
        minOccurs="0" maxOccurs="*" />
    <element type="Parser"
        minOccurs="0" maxOccurs="*" />
</ElementType>
</Schema>

```

The root element of an *XFiles* document is the *Properties* element. This element may contain, in any order, the following sub-elements:

- The *Type* element reflects the file type: ordinary, directory, pipe, symbolic or hard link, character or block device, or socket, on UNIX-like systems; the *mime* attribute specifies the MIME (Multipurpose Internet Mail Extensions) type for a file (i.e. *text/html*, *image/png*, or *application/executable*);
- The *Location* element denotes the IP address of the host on which file resides; the *dns* attribute is used to specify the Domain Name System (DNS) entry for the given IP address;

- The *Auth* element specifies the authentication method for accessing a given file;
- The *Owner* element denotes the information about the owner of a file: login name, password, group, real name; it is possible to have many owners for a single given file;
- The *Size* element specifies the actual file size; *max* attribute denotes the maximum permitted size for a file;
- The *Permissions* element reflects the set of file permissions;
- The *Timestamp* element gives the possibility to track the access, modification or status-change time of a specific file;
- The *Version* element can be used in a Concurrent Versions System (CVS) environment, for versioning purposes;
- The *Parse* element denotes the application(s) used to process a file (e.g. file editors, compilers, viewers etc.); the *params* attribute can be used to pass additional options to a program.

We omit other low-level details (such as file i-nodes or device numbers).

The proposed specification can be applied for any (distributed) file system.

From this moment, the proposed XML-based language can be used in a RDF statement to express different properties about the resources of a common file system.

### Examples

We provide a couple of examples of RDF constructs about the resources of a distributed file system:

To specify an ownership property and a password-based authorization method to access a set of files stored on the local machine and on a remote host, we can express the subsequent RDF assertions:

```

<rdf:RDF>
  <rdf:Bag ID="myfiles">
    <rdf:li resource="file:///tmp/article.tex" />
    <rdf:li
      resource="ftp://ftp.tuiasi.ro/pub/src/gaen" />
  </rdf:Bag>

  <rdf:Description about="#myfiles">
    <f:Properties>
      <f:Auth>Basic</f:Auth>
      <f:Owner>
        <rdf:Description
          about="http://www.infoiasi.ro/~busaco">
          <f:Login uid="714">busaco</f:Login>
          <f>Password>NU74b33cs</f>Password>
        </rdf:Description>
      </f:Owner>
      <f:Permissions>
        <f:Permission>User-Read</f:Permission>
        <f:Permission>User-Write</f:Permission>
        <f:Permission>Group-Read</f:Permission>
      </f:Permissions>
    </f:Properties>
  </rdf:Description>

```

```

    </f:Permissions>
  </f:Properties>
</rdf:Description>
</rdf:RDF>

```

We express the fact: "For the given collection of files, the owner of these files is the user *busaco*. The files will be accessed by providing a password and only the owner will be able to read and write them. The owner's group members will be able only to read them." The *f*namespace corresponds to all elements and attributes of our defined *XFiles* language.

The next RDF document details the alternatives for a remote execution of an application (an XML parser):

```

<rdf:RDF>
  <rdf:Description
    about="file://localhost/article.xml">
    <f:Properties>
      <f:Type mime="text/xml">ordinary</f:Type>
      <f:Owner uid="714">busaco</f:Owner>
      <f:Parser params="-q">
        <rdf:Alt>
          <rdf:li>
            <rdf:Description
              about="file:///sbin/expat">
              <f:Type
                mime="application/executable">
                ordinary
              </f:Type>
              <f:Location dns="localhost">
                127.0.0.1
              </f:Location>
            </rdf:Description>
          </rdf:li>
          <rdf:li>
            <rdf:Description
              about="nfs://host/xmlled.exe">
              <f:Type
                mime="application/octet-stream">
                ordinary
              </f:Type>
              <f:Location dns="it2.infoiasi.ro">
                193.231.30.228
              </f:Location>
            </rdf:Description>
          </rdf:li>
        </rdf:Alt>
      </f:Parser>
    </f:Properties>
  </rdf:Description>
</rdf:RDF>

```

In this second example, we can remark the use of RDF statements to specify two applications (defined as RDF *Alt* elements), one on the local machine, the second on a Windows file system accessed via NFS [13]. One of these applications will be executed to process the given file (in our example, an XML source file).

#### Expressing temporal relations between resources

We can extend *XFiles* language to allow expressing simple assertions between temporal relations between resources of a distributed file system or between resources of a Web site. These temporal relationships can be used to determine the dynamics of that Web sites' content and can assist,

among others, the WWW robots or agents in mirroring/searching activities.

An example of such extension follows:

```

<rdf:RDF>
  <rdf:Bag id="RecentlyChanged">
    <rdf:li resource="index.html" />
    <rdf:li resource="figure.gif" />
    <rdf:li resource="styles.css" />
  </rdf:Bag>
  <rdf:Description aboutEach="#RecentlyChanged">
    <!-- spatial information -->
    <f:Location
      f:dns="www.site.org">
        193.231.30.1
    </f:Location>
    <!-- metadata information -->
    <f:Owner>
      <rdf:Description
        about="http://www.infoiasi.ro/~busaco">
        <f>Login f:uid="714">busaco</f>Login>
      </rdf:Description>
    </f:Owner>
    <!-- temporal information -->
    <f:link f:type="temporal" f:action="Update">
      f:end="Sun Mar 17 19:35:14 EET 2002">
        <f:Finishes f:dur="2sec" />
    </f:link>
  </rdf:Description>
</rdf:RDF>

```

The collection of resources denoted by *RecentlyChanged* identifier is stored on *www.site.org* machine. For this collection an update was performed on March 17 2002. All given resources were synchronized to end the updating activity after 2 seconds.

## V. CONCLUSIONS

In this paper, we have proposed a RDF-based description that can be used to express different (spatial or temporal) relations between Web resources. The RDF constructs specify various relationships between resources and components of a single file system or related file systems.

For validation and parsing purposes, we have specified an XML schema for the *Extensible File Properties Markup Language (XFiles)* language. This language can be associated to RDF statements and can be used in design and implementation stages of file naming and replication.

The given RDF model can be validated and manipulated by *Simple RDF Parser and Compiler (SiRPAC)* [12] – a freely available Java servlet based on Megginson's SAX (Simple API for XML) processor.

An implementation of defined RDF model and proposed XML language can be also based on the *Document Object Model (DOM)* specifications [4, 14] or on the freely available processing library – *Libxml* – under Linux operating system [11].

## REFERENCES

- [1] T. Berners-Lee et al. (eds.), Uniform Resource Identifiers (URI): General Syntax, Internet Standard, RFC-2396, 1998
- [2] D. Brickley, R. Guha (eds.), Resource Description Framework (RDF) Schema Specification, W3C Candidate Recommendation, Boston, 2000: <http://www.w3.org/TR/rdf-schema>
- [3] T. Bray et al. (eds.), Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, Boston, 2000: <http://www.w3.org/TR/REC-xml>
- [4] S. C. Buraga, Web Technologies (in Romanian), Matrix Rom Publishing House, Bucharest, 2001.
- [5] S. C. Buraga, "A RDF Description of Distributed File Systems", in The Scientific Annals of the "Alexandru Ioan Cuza" University of Iasi, Computer Science Section, Tome IX, 2000
- [6] S. C. Buraga, "A Model for Accessing Resources of the Distributed File Systems", in The Proceedings of NATO ARW on Cluster Computing – Mangalia 2001, Lecture Notes in Computer Science LNCS 2326, Springer-Verlag, 2002.
- [7] W. Conen, R. Klapsing, "A Logical Interpretation of RDF". Linkoping Electronic Articles in Computer and Information Science, 5, 2000: <http://www.ida.liu.se/ext/epa/cis/2000/013/tcover.html>
- [8] P. Hayes (ed.), RDF Model Theory, W3C Working Draft, Boston, 2002: <http://www.w3.org/TR/rdf-ml/>
- [9] O. Lassila, R. Swick (eds.), Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation, Boston, 1999: <http://www.w3.org/TR/REC-rdf-syntax>
- [10] A. Tanenbaum, Modern Operating Systems, Third Edition, Prentice-Hall PTR, 2001.
- [11] \* \* \*, Libxml: <ftp://ftp.gnome.org/pub/GNOME/sources/libxml/>
- [12] \* \* \*, SiRPAC: <http://www.w3.org/RDF/Implementations/SiRPAC>
- [13] \* \* \*, The NFS Distributed File Service. Sun's NFS White Paper, 1995: <http://www.sun.com/software/white-papers/wp-nfs>
- [14] \* \* \*, World Wide Consortium's Technical Reports, Boston, 2002: <http://www.w3.org/TR/>