

A RDF Description of Distributed File Systems

Sabin Corneliu Buraga
Faculty of Computer Science, “A.I.Cuza” University of Iasi
G-ral. Berthelot Str., 16, Iasi, Romania
busaco@infoiasi.ro

Abstract

The modern operating systems must integrate various Internet services, especially World-Wide Web facilities to access Web resources using file systems mechanisms. In this paper we present a high-level model describing an abstract distributed file system. The proposed description is based on *Resource Description Framework (RDF)* recommendation of the World-Wide Web Consortium, a standardized foundation for processing metadata. To represent the RDF statements about various file characteristics, we propose an XML-based language called *Extensible File Properties Markup Language*.

1 Introduction

A *distributed system* is a collection of loosely coupled computers interconnected by a communication network. From the point of view of a specific computer in a distributed system, the rest of the machines (also known as *hosts*) and their respective resources are remote, whereas its own resources are local.

Important part of a distributed operating system, a *file system* provides file services to clients (other hosts of the network). A client interface for a file service is formed by a set of primitives, called *file operations*, such as open a file, remove a file, read from a file, write to a file, and so on.

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 vary and it is difficult to determine the best implementation. A distributed file system can be implemented as part of a distributed operating system or by a software layer whose primary function is to manage 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 [6, 18], broadly used on Unix-like systems, Microsoft’s *Active Directory* implemented on Windows

2000 [15], *Prospero* [16], an Internet-compatible virtual system model based on Uniform Resource Identifiers (URIs), and *Coda* [7], an experimental file system developed at Carnegie Mellon University.

In this paper we propose a high-level description of a virtual (distributed) file system using *Resource Description Framework (RDF)*, a model for processing metadata. RDF provides interoperability between applications that exchange machine-understandable information on the World-Wide Web.

One of the major goals of RDF is to make it possible to specify semantics for data based on *Extensible Markup Language (XML)* in a standardized, platform-independent, and object-oriented manner. RDF can be used in resource discovery, in cataloging activities, by intelligent software agents, in content rating, in describing collections of data, etc. We describe the important features and the syntax of RDF in section 3.

Our RDF model can be applied for a particular distributed file system. To illustrate some specific issues we choose the Unix file system structure. For expressing various file properties, in section 4.1 we will define an XML-based language called *Extensible File Properties Markup Language (XFiles language)*. The elements of XFiles language will be used to specify RDF statements about the components of a distributed file system or about the relationship between these components.

Also, the proposed RDF description can be used to formulate high-level assertions about main characteristics of a distributed file system in a standardized and platform- and implementation-independent manner.

2 File Systems

Most visible aspect of an operating system, the file system consists of two distinct parts: the collection of the actual *files*, each 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):

- *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.

$$seek(h, p), h \in \text{Handlers}, p \in \mathbb{Z}$$

- *read()* is used to read from a file specific data into a memory buffer.

$$read(h, m), h \in \text{Handlers}, m \in \text{Memory}$$

- *write()* is used to write to a file specific data stored into a memory buffer.

$$write(h, m), h \in \text{Handlers}, m \in \text{Memory}$$

Formally, the Handlers, FileNames, Memory sets are abstract data types. In practice, $\text{Handlers} \subset \mathbb{N}$, $\text{FileNames} \subset \text{Chars}^+$ and $\text{Memory} \subset \mathbb{N}$, where Chars is a set of valid filename characters (subset of ASCII code).

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

Each particular file system presents same 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* 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 specialized class to specify particular primitives (methods) for dealing with that file system. In the *vnode* class, each operation can be considered as pure virtual.

For uniformity, the *vnode* class will be used to represent pipes, devices, pseudo-devices, processes or 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 [9, 11].

In Unix (particularly Linux), this model is implemented by means of a special data structure called *inode-operations* [9]. 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 manage file information 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. An important problem for naming is to find a proper naming scheme. A practical solution is to attach 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) to locate files. The detailed description and complete syntax of the URIs is presented in [1].

If a distributed file system is transparent, the file location is not important for an user. This approach leads to the possibility of file *replication*, a useful redundancy for improving data availability. The replication process is used on Windows 2000 systems and on Linux systems – in this second case via *Network Information Service (NIS)*, formerly known as Yellow Pages. The main purpose of file replication is to provide 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 [7] 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.

A solution is given by the Prospero distributed file system. Another one is to propose a high-level RDF description of the file system resources using XML syntax (see section 4).

3 Resource Description Framework

3.1 General presentation

Resource Description Framework (RDF) is a standardized foundation for processing metadata [8]. RDF consists in a model for representing named properties and property values. RDF properties may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs. RDF properties also represent relationships between resources and a RDF model can therefore resemble an entity-relationship diagram. In object-oriented design terminology, resources correspond to objects and properties correspond to instance variables.

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*. Through the sharability of schemas, RDF supports the reusability of metadata definitions. The RDF schemas may themselves be written in RDF.

The basic model of RDF consists of three object types:

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

properties A *property* is a specific aspect, characteristic, attribute, or relation used to describe a resource, as stated in [8]. Each property has a specific meaning, defines its permitted values, the type of resources it can specify, 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 called, 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.

The RDF data model provides an abstract, conceptual framework for defining and using metadata. The concrete RDF syntax is based on *Extensible Markup Language (XML)* [4, 5, 12] – a platform independent, World-Wide Web Consortium’s standardized meta-language, subset of *Standard Generalized Markup Language (SGML)*.

3.2 RDF Syntax

The Extended Backus-Naur Form (EBNF) notation for RDF constructs takes the form (for more details, see [8]):

```
[1]  RDF      ::= ['<rdf:RDF>'] descript* ['</rdf:RDF>']
[2]  descript ::= '<rdf:Description' idAboutAttr? '>'
           propElt*
           '</rdf:Description>'
[3]  idAboutAttr ::= idAttr | aboutAttr
[4]  idAttr     ::= 'ID="' Idsymbol '"'
[5]  aboutAttr  ::= 'about="' URI-ref '"'
[6]  propElt    ::= '<' propName '>' value '</' propName '>' |
           '<' propName resAttr '/>'
```

- [7] propName ::= QName
- [8] value ::= descript | string
- [9] resAttr ::= 'resource=' URI-ref ''
- [10] QName ::= [NSprefix ':'] name
- [11] URI-ref ::= string
- [12] IDsymbol ::= (any XML legal symbol)
- [13] name ::= (any XML legal symbol)
- [14] NSprefix ::= (any XML namespace prefix)
- [15] string ::= (any XML data)

Using this syntax, we can represent in RDF/XML the following assertion about the owner of a particular file:

```
<rdf:RDF>
  <rdf:Description about="file:///home/busaco/">
    <f:Owner>Sabin-Corneliu Buraga</f:Owner>
  </rdf:Description>
</rdf:RDF>
```

In this example, the namespace prefix *f* refers to a specific namespace prefix chosen by the author of the RDF expression and defined in an XML namespace declaration such as:

```
xmlns:f="http://some.host/files-schema"
```

The *rdf* namespace is defined by World-Wide Web Consortium to be specified in every RDF statement.

The XML namespaces [3] are used to avoid parsing conflicts for identical elements or attributes names included in the same XML document.

We can specify different namespaces as follows:

```
<rdf:RDF>
  <rdf:Description about="file:///home/busaco/">
    <f:Owner>
      <rdf:Description
        about="http://www.infoiasi.ro/busaco">
          <o:Login>busaco</o:Login>
          <o:Group>profs</o:Group>
          <o:Name>Sabin-Corneliu Buraga</o:Name>
        </rdf:Description>
      </f:Owner>
    </rdf:Description>
  </rdf:RDF>
```

In this example, we express the following assertion: “The individual referred by *http://www.infoiasi.ro/busaco* is named Sabin-Corneliu Buraga and he has login name *busaco* and his user group is *profs*. The resource (file) */home/busaco* is owned by this individual”.

3.3 RDF Containers

The RDF model defines three types of container objects:

Bag (an unordered list of resources or literals);

Sequence (an ordered list of resources of literals);

Alternative (a list of resources or literals that represent alternatives for the single value of a property).

The EBNF syntax for RDF containers is:

```
[16] contain ::= seq | bag | alt
[17] seq     ::= '<rdf:Seq' idAttr? '>'
              member*
              '</rdf:Seq>'
[18] bag     ::= '<rdf:Bag' idAttr? '>'
              member*
              '</rdf:Bag>'
[19] alt     ::= '<rdf:Alt' idAttr? '>'
              member*
              '</rdf:Alt>'
[20] member  ::= referItem | inlineItem
[21] referItem ::= '<rdf:li resourceAttr' '/>'
[22] inlineItem ::= '<rdf:li>' value '</rdf:li>'
```

The collections can be used instead of *Description* element, and the new syntactic rules are:

```
[1a] RDF    ::= '<rdf:RDF>' obj* '</rdf:RDF>'
[8a] value  ::= obj | string
[23] obj    ::= descript | contain
```

The object being described (indicated by the *about* attribute of *Description* element) is called the *referent*. The RDF model permits to define distributive referents expressed by statements about the members of a container. For example, to specify the configuration files of the *thor.infoiasi.ro* machine, we can write:

```

<rdf:Bag id="ConfigFiles">
  <rdf:li resource="file:///etc/passwd" />
  <rdf:li resource="file:///etc/shadow" />
  <rdf:li resource="file:///etc/group" />
  <rdf:li resource="file:///etc/gshadow" />
</rdf:Bag>
<rdf:Description aboutEach="#ConfigFiles">
  <f:Location dns="thor.infoiasi.ro">
    193.231.30.225
  </f:Location>
</rdf:Description>

```

The containers may be defined by an URI pattern. RDF can be used for making statements about other RDF statements (higher-order statements), too.

3.4 Formal model for RDF

The RDF data model can have three equivalent representations: as 3-uples, as a graph, and in XML. Formally, the RDF data model is defined as follows:

1. There are three sets called *Resources*, *Literals*, and *Statements*. There is a subset *Properties* \subset *Resources*. Each element $s \in$ *Statements* is a triple $s = \{pred, sub, obj\}$, where $pred \in$ *Properties*, $sub \in$ *Resources*, and $obj \in$ *Literals* \cup *Resources*.
2. The *reification* of a $\{pred, sub, obj\} \in$ *Statements* is a new resource $r \in$ *Resources* as follows:

$$\begin{aligned}
 s_1 &: \{\text{type}, [r], [RDF : \textit{Statement}]\} \\
 s_2 &: \{\text{predicate}, [r], [pred]\} \\
 s_3 &: \{\text{subject}, [r], [sub]\} \\
 s_4 &: \{\text{object}, [r], [obj]\}
 \end{aligned}$$

where $s_1, s_2, s_3, s_4 \in$ *Statements* and type, predicate, subject, object \in *Properties*.

3. There is an element of *Properties* known as *RDF : type*. Members of *Statements* of the form $\{RDF : type, sub, obj\}$ must satisfy the following condition: *sub* and *obj* are members of *Resources*.
4. There are three elements of *Resources*, not contained in *Properties*, known as *RDF : Seq*, *RDF : Bag* and *RDF : Alt*. There is a subset *Ord* \subset *Properties* corresponding to the ordinals (1, 2, 3, ...). We refer to elements of *Ord* as *RDF_i*, where $i = 1, 2, 3, \dots$

4 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)*.

4.1 An XML schema for XFiles language

For validation and parsing purposes, we will present an XML schema for this language. We adopt the *XML Schema* specification (see [5]) instead of *Document Type Definition* approach, because schemas are more flexible and powerful and they can be easily processed by XML parsers. An XML schema gives 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"
    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:

- *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) [2] type for a file (i.e. `text/html`, `image/gif`, or `application/executable`);
- *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;
- *Auth* element specifies the authentication method for accessing a given file (e.g. basic user authentication);
- *Owner* element denotes the information about the owner of a file: login name, password, group, real name; it is possible to have multiple owners for a single given file;

- *Size* element specifies the actual file size; *max* attribute denotes the maximum permitted size for a file;
- *Permissions* element reflects the set of file permissions, e.g. `read`, `write`, `execute` (on Unix) or `Full Control`, `Change Permissions`, `Read`, or `Take Ownership` (on Windows);
- *Timestamp* element gives the possibility to track the access, modification or status-change time of a specific file;
- *Version* element can be used in a Concurrent Versions System (CVS) environment, for versioning purposes;
- *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). This 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 various properties about the resources of a general file system.

4.2 Examples

In this section, we will give three examples of RDF constructs about the resources of a distributed file system.

1. To model the remote access mechanism for all Postscript files of a given user (e.g. *busaco*), we can formulate the following RDF statement:

```
<rdf:RDF>
  <rdf:Description
    aboutEachPrefix="http://www.infoiasi.ro/">
    <f:Properties>
      <f:Type mime="application/postscript">
        ordinary
      </f:Type>
      <f:Owner>
        <f:Login uid="714">busaco</f:Login>
        <f:Group gid="201">profs</f:Group>
      </f:Owner>
    </f:Properties>
  </rdf:Description>
</rdf:RDF>
```

The *f* namespace corresponds to all elements and attributes of our defined *XFiles* language.

2. To specify an ownership property and a password-based authorization method to access a set of files stored on the local machine, we can define the following RDF assertions:

```
<rdf:RDF>
  <rdf:Bag ID="myfiles">
    <rdf:li resource="file:///tmp/book.tex" />
    <rdf:li resource="file:///home/busaco/" />
  </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>
</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 to read them, only.”

3. The next RDF document specifies 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:Version>
        <rdf:Description
          about="http://www.infoiasi.ro/busaco">
          <dc:Subject>
            <rdf:Bag>
              <rdf:li>
                computer science
              </rdf:li>
              <rdf:li>
                RDF
              </rdf:li>
              <rdf:li>
                file systems
              </rdf:li>
            </rdf:Bag>
          </dc:Subject>
        </rdf:Description>
      </f:Version>
      <f:Parser params="-q">
        <rdf:Alt>
          <rdf:li>
            <rdf:Description
              about="file:///usr/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/My%20Progs/xmlled.exe">
              <f:Type mime="application/octet-stream">
                ordinary
            </rdf:Description>
          </rdf:li>
        </rdf:Alt>
      </f:Parser>
    </f:Properties>
  </rdf:Description>

```

```

        </f:Type>
        <f:Location dns="it3.infoiasi.ro">
            193.231.30.227
        </f:Location>
    </rdf:Description>
</rdf:li>
</rdf:Alt>
</f:Parser>
</f:Properties>
</rdf:Description>
</rdf:RDF>

```

The *dc* namespace is defined by *Dublin Core Metadata Initiative* [14] which provides 15 types of elements used to describe the content of various World-Wide Web resources. In our case, only the *Subject* element is used. In this example, we can observe the presence of *Version* element and the use of RDF statements to specify two applications (defined as RDF *Alt* elements), one on the local machine, the second on an Windows file system accessed via NFS. One of these applications will be executed to process the given file (in our example, an XML source file).

5 Conclusions

In this paper, we have proposed a RDF description for distributed file systems. Our XML-based approach is platform and implementation independent and can be used for any particular file system.

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)* presented in subsection 4.1. This language can be associated to RDF statements and can be used in design and implementation stages of file naming and replication.

The flexibility and extensibility of XML schemas may be key factors in concrete implementations. The given RDF model can be validated and manipulated by *Simple RDF Parser and Compiler (SiRPAC)* [17], a freely available Java servlet based on Megginson's SAX (Simple API for XML) processor.

An implementation of our RDF model and proposed XML language can be also based on the *Document Object Model (DOM)* [13] specification.

The RDF description of distributed file systems needs to be enriched by a formal RDF schema and particular (low-level) specifications for Linux and Windows platforms.

References

- [1] T. Berners-Lee *et al.* (eds.), *Uniform Resource Identifiers (URI): General Syntax*, Internet Standard, RFC 2396, 1998
- [2] N. Borenstein, N. Freed, *MIME (Multipurpose Internet Mail Extensions)*, RFC 1341, Network Working Group, 1992
- [3] T. Bray, D. Hollander, A. Layman (eds.), *Namespaces in XML*, W3C Recommendation, Boston, 1999:
<http://www.w3.org/TR/REC-xml-names>
- [4] T. Bray *et al.* (eds.), *Extensible Markup Language (XML) – version 1.0 (updated)*, W3C Recommendation, Boston, 2000:
<http://www.w3.org/TR/REC-xml>
- [5] A. Ceponkus, F. Hoodbhoy, *Applied XML*, Wiley Computer Publishing, New York, 1999
- [6] D. Comer, D. Stevens, *Internetworking with TCP/IP: vol III: Client-Server Programming and Applications*, Prentice Hall, New Jersey, 1993
- [7] M. Kramer, *Distributed File Systems*, IBM White Paper, Boston, 1996
- [8] 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>
- [9] D. Rusling, *The Linux Kernel Reference*, 2000:
<http://metalab.unc.edu/pub/Linux/docs/LDP/linux-kernel>
- [10] A. Silberschatz, J. Peterson, P. Galvin, *Operating Systems Concepts*, Addison-Wesley, Reading MA, 1992
- [11] W. R. Stevens, *Advanced Programming in the Unix Environments*, Addison-Wesley, Reading MA, 1992
- [12] N. Welsh, *A Technical Introduction to XML*, ArborText Inc., 1998
- [13] L. Wood (ed.), *Document Object Model (DOM) Level 1 Specification*, W3C Recommendation, Boston, 1998:
<http://www.w3.org/TR/REC-DOM-Level-1>
- [14] *Dublin Core Initiative Web site*:
http://purl.org/metadata/dublin_core
- [15] *Microsoft Windows 2000 Web Site*:
<http://www.microsoft.com/windows2000>

- [16] *Prospero Web Site*:
<http://www.cuhk.hk/guides/earn/prospero.html>
- [17] *SiRPAC*:
<http://www.w3.org/RDF/Implementations/SiRPAC>
- [18] *The NFS Distributed File Service*. Sun's NFS White Paper, 1995:
<http://www.sun.com/software/white-papers/wp-nfs>