

Towards a Semantic-Aware File Store

Zhichen Xu, Magnus Karlsson, Chunqiang Tang* and Christos Karamanolis
HP Laboratories, 1501 Page Mill Rd., MLS 1177, Palo Alto, CA 94304

{zhichen,karlsson,chunqian,christos}@hpl.hp.com

Abstract—Traditional hierarchical namespaces are not sufficient for representing and managing the rich semantics of today’s storage systems. In this paper, we discuss the principles of semantic-aware file stores. We identify the requirements of applications and end-users and propose to use a generic data model to capture and represent file semantics. A distinct challenge that we face is to handle dynamic evolution of the data schemas. Further, we outline a framework of basic relations and tools for generating and using semantic metadata. The proposed data model and framework are aimed to be more generic and flexible than what is offered by existing semantic file systems. We envision a range of applications and tools that will exploit semantic information, ranging from personal storage systems with features for advanced searching and roaming access, to enterprise systems supporting distributed data location or archiving.

1 Motivation

Over the last several years, we have witnessed an unprecedented growth of the volume of stored digital data. In 1999, a study estimated the amount of original digital data generated annually to be in excess of 1,700 petabyte [14]. It is estimated that this number has been nearly doubling annually since then [21]. This explosive growth is reflected on the ever increasing complexity and cost for storage management. One instance of this problem occurs in file stores. The traditional hierarchical file system is no longer adequate for systems that need to store billions of files and capture different types of semantic information that is required to efficiently access, share, and manage those files.

Consider, for example, the case of a digital movie production studio. Digital movies consist of hundreds of scenes. Each scene is composed of thousands of different data objects, including character models, backgrounds, and lighting models. These objects are typically implemented as files that are shared by tens of artists. There is a range of semantic information that needs to be captured and used in this environment. When a new version of the hair of a character is created, it has to be annotated with the changes done. Further, it is compatible with only certain

versions of the head. Such information about *versions* and *dependencies* among files is important when rendering a scene; it is required to combine objects that are compatible with each other and make sense in some context. When composing a scene, an artist uses material that other people have edited and stored in the system. *Content-based searching* (e.g., search for “green lush grass”) as opposed to searching by file name can greatly simplify collaboration and improve productivity. The *view* of what data are stored in the system may potentially be different depending on application and user. For example, an artist wants to see only objects that are compatible with the version of the character she is working on; a backup system only sees files that are marked as “persistent” by the artists. Further, tracking context information, such as the files accessed before, accessed by users, and other statistical information may enable intelligent resource provisioning, data caching and prefetching, and improve search efficiency and accuracy.

Examples of common types of semantic information that needs to be captured include: (i) file versioning, (ii) application-based dependencies, (iii) attribute-based semantics, (iv) content-based semantics, and (v) context-based information.

Considered individually, some of these types of semantic information are captured and used by existing applications and tools, such as versioning control systems or software configuration tools. However, different types of semantic information often depend on each other and are related to other functions of a storage system. For example, application-based dependencies are defined on versions of files. Also, dependencies need to be considered during archiving, to save a consistent snapshot of the application state. We argue that it is easier and more efficient to manage all the above types of semantic information in a single, general-purpose system, that many applications can use.

Along these lines, we propose a *semantic-aware* file store, named *pStore*, that extends file systems—a storage abstraction assumed by many applications—to support semantic metadata. The paper makes the following contributions.

- Proposes using a generic data model to represent semantic information in file systems. The data model has two main features. First, it is extensible to cover seman-

*Chunqiang Tang is with Department of Computer Science, University of Rochester, Rochester, NY.

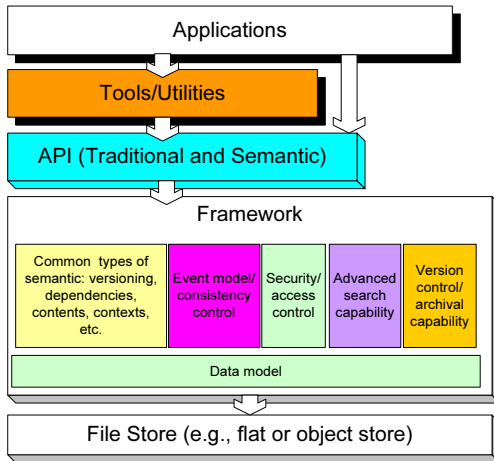


Figure 1: Architecture of pStore.

tic information other than the types described above. Second, handles schema evolution, which is essential for many data management applications where semantic information is discovered incrementally.

- Introduces a framework with built-in support for representing and providing access to a set of basic types of semantic information in file systems.
- Outlines a range of applications and tools that can exploit rich semantic information.
- Concludes with a list of research challenges that need to be addressed to realize the vision.

2 Architecture of pStore

The architecture of pStore is illustrated in Figure 1. pStore makes no particular assumption of the underlying file repository, except that it provides a flat space of unique object IDs. The core of pStore is a generic data model that is used to represent semantic information. On top of the data model, a set of basic functionality modules are provided to programmers that wish to develop tools of applications that use or change the semantic data. We describe the basic components of pStore in the following sections.

2.1 Semantic data model

pStore proposes using a generic data model to capture different types of semantic information in file stores. The data model should meet the following requirements.

- Allow to specify well-defined schemata (schema definition language).
- Support dynamic schema evolution to capture new or evolving types of semantic information.
- Be simple to use, lightweight, make no assumptions about the semantics of the metadata.

- Be platform independent and provide interoperability between applications that manage and exchange metadata.
- Facilitate integration with resources outside the file store and support exporting metadata to the web.
- Leverage existing standards and corresponding tools, such as query languages.

Database systems do not fulfill the above requirements, because of two main reasons. First, DBs typically require a predefined schema and impose strict integrity constraints. They cannot effectively deal with incremental and dynamic schema evolution, which is common in managing unstructured data. Second, not all applications require the heavyweight ACID properties and all the features of a fully-fledged DB. For example, Unix file systems do not guarantee the ACID properties in the face of system failures.

Based on these requirements, we propose using a data model that is based in the *Resource Description Framework* (RDF) [22]. RDF has been proposed to encode, exchange and reuse metadata on the Web (a fundamental tool for realizing the Semantic Web vision [20]). RDF has two main advantages. First, it provides the means to capture schemata for metadata that are both human-readable and machine-processable (RDF notations are typically defined in XML). Second, it is designed to allow reuse and extensions of existing schemata for an ever evolving set of semantic metadata.

RDF is a model that describes resources. Relations, in RDF, are expressed as tuples of the form:

subject property object

In our case, the subject is a file in the file store. The properties (one or more) that are associated with the subject capture some type of semantic property of the corresponding file. The object of the relation corresponds to the value of the property for the subject, which may be another file or some metadata structure (a literal or composite). Thus, files and metadata structures are both considered resources. In fact, relations themselves can be used as resources for constructing more complex metadata relations.

RDF provides no vocabulary that assumes or refers to application-specific semantic information, e.g., certain properties for media files or relations of files that are accessed by the same user. Instead, such classes of resources and properties are defined in the form of an RDF schema. The same RDF notation is used to specify RDF schemata [23]. This is achieved by providing a set of predefined resources, namely *Classes* and *Properties*. For example, in our case, a Class may refer to files with a certain type of content or files that are used by a certain application. For the model, the specific files are resources

that are instances of a certain Class. A Property is defined in the schema to have a *domain* and a *range*. Each of them can be defined to refer to resources of one or more classes. Classes and Properties can be defined in a hierarchical fashion resulting in schemata that capture complex semantic information.

The principles of RDF resemble those of graph-based data models that have been proposed to handle structural irregularity and incompleteness of schemata and rapid schema evolution [1]. In such systems, the schema is non-mandatory, i.e., it provides some information about the current type of the data, but it does not constrain the format of the data. We have chosen RDF, as it is simple and standardized.

A remaining issue is how to implement a repository of RDF relations in a system. We intend to use some lightweight, RISC-style database systems, like the one proposed by Chaudhui and Weikum [4].

2.2 Basic relations

In the following, we describe a number of relations that cover the set of common types of semantic information listed in Section 1. An RDF schema is defined for each of these relations, but it is not provided here, due to space restrictions. Neither do we use RDF notation to describe relations. Instead, we use an informal triplet notation, as above, using curly brackets to represent composite properties (constructed by means of blank properties or containers in RDF).

File versioning. Each file in pStore corresponds to one *file object* and multiple *file version objects*¹. Each update to the file automatically creates a new file version. The notion of a “file” will be represented by a data object that captures some of the basic attributes of the file (owner, file name, etc). For example, it could be the root node in a hierarchical content-addressable storage system [16]. As soon as the file has some content, each version of the file is represented by another object.

There are two types of relations between a file and its versions. Relation *o1 has_version{o2, v1}* states that object with id *o2* is version *v1* of *o1*. Similarly, *o1 latest_version{o2}* states that object *o2* is the latest version of *o1*. Property *has_version* may have additional attributes, such as *creation_time*, and *comment*.

Hierarchical name space. The traditional hierarchical name space is defined using the *is_parent_of* and *in_directory* properties. E.g., “*movie1 is_parent_of sequence2*” represents the file path “*movie1/Sequence2*”. File system access control is represented by the *access_control* property. The range of this property is a Class that defines, e.g., an ACL structure.

Dependencies. In addition to the hierarchical relations, a user can define other types of dependencies among ob-

¹These are data objects, not necessarily related with the object of an RDF relation.

jects. In fact, *is_parent_of* is just one instance of Property schema *Depend_on*. Instances of this Property may be application specific. For example, the relation *Shrek char_dep Ogre*, where *char_dep* is an instance of *Depend_on*, means that file *Shrek* has a dependency on file *Ogre*. Another example of dependency is the relationship between the master copy of the data and its replicas.

Associative semantics. Another common relationship is that of a metadata object describing an ordinary file. For instance, *Fiona comments text* indicates that object *text* describes the *Fiona* character. Such metadata will, in many cases, be automatically extracted and used for searching, as explained in the next section.

Context information. The data model can also be used to track context information from the file system and user behavior. Examples of related properties include *no_reads*, *no_writes*, *accessed_before*, *accessed_by*, and *accessed_from*. For example, we can use *hair accessed_before {time=5s, nose}* to record the fact that file *hair* is accessed 5 seconds before accessing file *nose*. This information can be used, to gather statistics that pStore (or applications) can use to improve the performance of the system. Examples include prefetching and caching in distributed environments, data placement, as well as advanced searching.

An important challenge that needs to be addressed is automatically extracting various types of semantic information from data. E.g., people use vector space models to extract features from text documents and images [2, 5]. Similarly, they derive frequency, amplitude, and tempo feature vectors from music data [6]. More recently, Soules and Ganger [18] proposed methods for capturing file attributes and inter-file relations, by analyzing user access patterns.

2.3 Dynamic evolution of schema

We expect pStore to provide a set of default schemata, like the ones above (and possibly more). However, we expect users to modify these schemata. For example, in many data management applications, relationships among data objects are identified after the objects are created and may change during the lifetime of the objects, as their usage changes. The usage of data and metadata is often unpredictable and may depend on the actual user or workload. Incremental elaboration of data object classes and their properties is often inevitable. We also expect users to define their own schemata and share them in ad-hoc manners to cover application or site-specific requirements among communities of users.

RDF supports dynamic evolution of schema in multiple ways. First, it supports refinement of schema through class inheritance and property polymorphism. Second, the *namespace* feature of RDF allows for schemata to evolve differently in different contexts, such as application versions or user communities. Last, but not least, the fact that RDF provides a machine-readable notation, facilitates the design of programmable interfaces and tools that allow for

automatic extraction, manipulation and exchange of relations and schemata.

2.4 Framework

The pStore framework offers built-in support for representing and accessing semantic metadata in file stores.

Event model/consistency control. Inter-file dependencies is an important type of semantic information captured by pStore. Often, such dependencies imply some consistency requirement users assume between the related files. Such requirements vary for different instances of a relation, or even across time.

We capture such consistency requirements by augmenting dependency relations with an associated relation of type `Event`. An event consists of an ordered list of $\langle \text{precondition: action} \rangle$ tuples (implemented as a `rdf:seq` container in RDF). When a data object is accessed (e.g., open, write), the system checks each of these preconditions and executes the corresponding actions if the precondition holds. Suppose that object *Shrek* depends on object *Ogre*. One of the events associated with that relation may look like $\langle \text{modified: rebuild}(\text{Shrek}) \rangle$, specifying that *Shrek* needs to be regenerated if *Ogre* is modified.

Customized name space views. In addition to the conventional hierarchical name space, the data model provides the basis on which customized per-user or per-application name spaces can be constructed. We sketch several ways that this can be done.

One way to construct customized name spaces is by constraining the corresponding relations. A special case is when the customized name space is a sub-graph of the original file system hierarchy. For instance, *Shrek* `is_parent_of` $\{ \text{user}=\text{Mary}, \text{script} \}$ states that object *Shrek* is a parent directory of object *script* only for user Mary. Another possibility is to exploit Property inheritance in the schema. For example, `Property land_mammal\{feet\}` can be regarded as a *super class* of `Property elephant\{feet, trunk\}`.

In principle, a virtual directory can be created to include links to an arbitrary set of files, e.g., results for content-based searches [8].

Security and access control. In an enterprise environment such as a digital movie studio, data is its biggest asset. Thus, data dependability is of paramount importance. They use mechanisms such as encryption and access control to protect the data and mechanisms such as erasure coding and replication for high reliability and availability. We envision that such data dependability mechanisms can be represented using our data model. They include, for example, relations such as `allow_user` and `deny_user` to be used for access control, or relations that capture the number and location of file replicas. RDF Property inheritance can be used to fine tune the relations for certain types of data.

Advanced searching capabilities. One of the open research questions in storage systems today is how to per-

form advanced and efficient searching of content in large corpuses of data. Our model and framework provide a uniform platform for integrating *content*, *attribute*, and *context-based* searching. For example, it can be used in combination with information retrieval algorithms [2] that depend on semantic information from the data. Similarly, our model can capture context information (such as access patterns) and inter-file relationships that can be used for advanced context-based searching [18]. We would also like to provide searching with variable recall and precision to be able to trade-off this against speed. Especially for queries where the recall and precision are not 100%, the ranking of the search results becomes important. This is an area where context information has been successfully used, for example in Google.

Archival support. An on-line archival storage system is one of the main applications we envision for pStore. Compression and versioning are essential given the volume and complexity of the data [16]. The semantic information that our model can capture about the data can be used to reduce storage consumption [11] and facilitate efficient data organization for fast data storage and retrieval.

3 Application Scenarios

In the following paragraphs, we describe some examples of applications of pStore other than a digital movie studio to demonstrate the generality of our proposal.

Online data sharing. In general, it is desirable that each object can have an arbitrary metadata structure suitable for describing its contents as well as its relationships with other objects. Objects can relate to each other in many different ways: an object may overlap with or include other objects; multiple objects may share descriptive data. In practice, meaningful objects are often identified and associated with their descriptive data incrementally and dynamically, after the data is stored in the system.

To provide adequate control, users can be given different access privileges. To facilitate collaboration, in addition to a shared global view of all the data, there may also be customized per-user and per-application views. Advanced searching capabilities are needed to allow people to effectively navigate among the various digital components.

A semantic, deep archival system. It is now practically affordable to archive each individual version of a file. Such archival storage systems are becoming essential for many critical applications. We list some desirable features.

First, a user would like the file store to have a “travel-in-time” capability—every change to an object or to the name space is recorded, and a user can travel arbitrary back in time to retrieve any version of a file that ever existed [11]. An important challenge is to maintain the various dependencies among different versions of objects and handle time as yet another type of semantic information.

Second, to reduce storage space consumption, objects should be stored efficiently. Various data clustering and

compression techniques are being explored. One way to do this is to exploit the available semantic information. E.g., when generating a new version of a file, the semantic information is used to identify an existing (base) file with similar contents. Only the differences between the new and the base file are stored.

Last, in restoring a backed-up version, the biggest headache is to find the right document and the right version. With pStore’s rich metadata model, the semantic information of files can be associated with files. In the restoring operation, the user describes a desired feature that is known to exist in the recovered version. For example, the system may use content extracts to locate the right version, without requiring the user remembering the exact name or creation date of the restored file.

Digital content distribution. In addition to search capabilities, a large-scale distributed file system can utilize the relationships among files to guide data placement, and perform caching and prefetching. CDN more efficient. Another related application is to support data hoarding for mobile users. Before disconnected from the network, all frequently used data for the user are identified through examining the metadata, and are automatically moved to a portable device. Systems such as SEER [10] use simple semantic hints such as user activity and directory membership for hoarding related files. Their effectiveness is limited by operations such as running the UNIX `find` utility across an entire file system.

Personal storage for desktop users. Many of the features described above can benefit ordinary desktop users as well. As desktop users, we would like to keep every version of important files that we ever created or downloaded, add arbitrary annotations to the files, relate them to their sources, and create cross links among them. Automated file hoarding can relieve much of the pain to manually identify and move files among computers and mobile devices. Many of us have painful experiences of not finding files. The advanced searching capability would make search much easier.

4 Related Work

Contemporary file systems use file type information to associate files with the appropriate applications to access them. Further, several systems have experimented with the idea of attribute-based file naming [7, 8, 12, 15, 17]. The file system supports searching on the basis of attributes; the results are reflected on virtual directories that contain pointers to the actual locations of files.

SFS [7] uses a hierarchical directory structure to organize refinements to previous query results. HAC [8] attempts to combine the benefits of hierarchical and content-based access to files at the same time. A virtual directory (resulting from a query) is an actual directory that allows ordinary file system operations. To maintain the consistency between links in a virtual directory and the files they point to,

HAC re-executes queries periodically to update the links in virtual directories.

Several systems allow for more flexible ways to combine the hierarchical name space with attribute-based file naming. A file system by Transarc [3] allows each file to have an associated wrapper, called a synopsis, that contains tag/value attributes and defines methods to manipulate those attributes. Synopses are organized in inheritance hierarchies. Similarly, in a system described in [17], each query is given a label. Users can impose “ancestor-descendant” relationship on labels, and consequently can name files by specifying either the path name that contains labels, or a list of queries the files satisfy, or both. In the Prospero system [12], users can program “filters” that create personalized views of file systems.

In Presto [15], documents can be organized according to properties (attributes) that are associated with the documents, without the limitations of hierarchies. Properties can be specific to an individual document consumer. Unlike HAC, Presto does not intend to handle backward compatibility to the traditional file system abstraction.

All these systems focus mainly on simple attributes; queries are limited to ad-hoc attribute match. pStore provides a generic data model and implementation that capture a more extensive set of semantics. We anticipate that these attributed-based file systems can be easily implemented using pStore and pStore’s generality can be explored to provide new functionalities that do not exist in these systems.

Several projects study metadata management in a file system setting. Roma [19] provides an available, centralized repository of metadata to “synchronize” a single user’s files across a diversity of digital storage devices. Roma metadata include fully-extensible attributes that could be used for organizing and locating files. However, the current prototype of Roma does not utilize attributes for searching.

The Inversion file system [13] runs on top of the POSTGRES database. It allows fine-grained time travel—a user may ask to see the state of the file system at any time in the past. Accesses to the file system are transactional. It is possible to issue ad-hoc queries on the file system metadata, or even to file data. IBM’s DataLink [9] project uses a relational database to capture a wide set of semantic information in file systems. The database contains references to objects in the file system. However, not all applications require the heavyweight ACID properties and features of a fully-fleshed database system. Moreover, database systems cannot effectively handle the incremental evolution of schema, common when managing unstructured data.

Our work complements the semantic Web [20] by concentrating on the system aspects and metadata management in a storage setting. Further, pStore provides additional functionality, e.g., tunable consistency based on an event-

framework. It is a framework that provides predefined but customizable components. One example is the predefined types of metadata (e.g., content- and context-based semantics) each possibly with predetermined consistency models.

5 Conclusion and Open Issues

The paper motivates the need to incorporate semantic metadata in file stores. We identify the basic types of semantic information required by applications and end-users and propose a generic data model to capture and represent file semantics. The model provides the basis for a framework of tools and APIs for generating and using semantic metadata. There is a large number of research problems that need to be addressed to realize a semantic-aware file store. We enumerate some of them below.

- The basic semantic relations sketched in section 2.2 are yet to be evaluated and finalized through the use of real applications.
- Investigate the design of semantic-aware deep-archival systems. In particular, what kind of semantic information can be used for improved data clustering and compression techniques. Also, how to maintain rich semantics for multiple versions of files; inheritance of semantic relations and their representation and use.
- Use semantic metadata for intelligent data placement in distributed storage systems. The goal is to satisfy the QoS requirements of end-users or applications with low infrastructure cost.
- Design and implement a basic set of tools and APIs for using the semantic information captured in such systems. These tools should be extensible and customizable. What these tools will be and how they will interact with each other is an open issue.
- Devise a simple declarative query language that can be used to specify constraints on both structured and unstructured data components.
- Investigate how the proposed data model and framework can be implemented in a distributed file system efficiently. One hard question is how to store RDF relations using a lightweight DB.

We are currently implementing a prototype of pStore to demonstrate its benefits in an online archival storage system.

References

- [1] S. Abiteboul. Querying semi-structured data. In *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, pages 1–18, 1997.
- [2] M. Berry, Z. Drmac, and E. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999.
- [3] M. Bowman. Managing Diversity in Wide-Area File Systems. In *Second IEEE Metadata Conference*, September 1997.
- [4] S. Chaudhuri and G. Weikum. Rethinking database system architecture: Towards a self-tuning RISC-style database system. In *The VLDB Journal*, pages 1–10, 2000.
- [5] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3(3/4):231–262, 1994.
- [6] J. Foote. An overview of audio information retrieval. *Multimedia Systems*, 7(1):2–10, 1999.
- [7] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O. Jr. Semantic file systems. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, 1991.
- [8] B. Gopal and U. Manber. Intergrating content-based access mechanisms with hierarchical file systems. In *the 3rd Symposium on Operating Systems Design and Implementation (OSDI)*, New Orleans, Louisiana, USA, 1999.
- [9] H.-I. Hsiao and I. Narang. DLFM: A Transactional Resource Manager. In *SIGMOD Conference 2000*, 2000.
- [10] G. H. Kuenning and G. J. Popok. Automated hoarding for mobile computers. In *Symposium on Operating Systems Principles*, pages 264–275, 1997.
- [11] M. Mahalingam, C. Tang, and Z. Xu. Towards a semantic, deep archival file system. In *The 9th International Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, May 2003.
- [12] B. C. Neuman. The prospero file system: A global file system based on the virtual system model. *Computing Systems*, 5(4):407–432, 1992.
- [13] M. A. Olson. The design and implementation of the Inversion file system. In *Proceedings of the USENIX Winter 1993 Technical Conference*, pages 205–217, San Diego, CA, USA, 25–29 1993.
- [14] P. Lyman, H.R. Varian, J. Dunn, A. Strygin, and K. Searingen. How much information, October 2000. <http://www.sims.berkeley.edu/research/projects/how-much-info>.
- [15] A. L. Paul Dourish, W. Keith Edwards and M. Salisbury. Using properties for uniform interaction in the presto document system. In *The 12th Annual ACM Symposium on User Interface Software and Technology*, Asheville, NC, USA, November 7–10 1999.
- [16] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *First USENIX conference on File and Storage Technologies*, Monterey, CA, USA, 2002.
- [17] S. Sechrest and M. McClennen. Blending hierarchical and attribute-based file naming. In *12th International Conference on Distributed Computer System*, Yokohama, Japan, June 1992.
- [18] G. A. N. Soules and G. R. Ganger. Why can't i find my files? new methods for automating attribute assignment. In *9th Workshop on Hot Topics in Operating Systems (HotOS-IX)*, Lihue, Hawaii, May 18-21 2003.
- [19] E. Swierk, E. Kiciman, V. Laviano, and M. Baker. The roma personal metadata service. In *Proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications*, Monterey, CA, USA, December 2000.
- [20] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.
- [21] The Enterprise Storage Group. Reference information: The next wave “the summary of: A snapshot research study by the enterprise storage group”, 2002. <http://www.enterprisestoragegroup.com>.
- [22] W3C. Resource description framework (rdf) model and syntax specification, February 22 1999. <http://www.w3.org/TR/REC-rdf-syntax/>.
- [23] W3C. Resource description framework (rdf) schema specification, March 3 1999. <http://www.w3.org/TR/1999/PR-rdf-schema-19990303/>.