The SCAM Framework: Helping Semantic Web Applications to Store and Access Metadata

Matthias Palmér, Ambjörn Naeve, and Fredrik Paulsson

KMR group at CID/KTH (Royal Institute of Technology)
Lindstedtsvägen 5
100 44 Stockholm, Sweden
{matthias,amb,frepa}@nada.kth.se

Abstract. In this paper we discuss the design of the SCAM framework, which aims to simplify the storage and access of metadata for a variety of different applications that can be built on top of it. A basic design principle of SCAM is the aggregation of metadata into two kinds of sets of different granularity (SCAM records and SCAM contexts). These sets correspond to the typical access needs of an application with regard to metadata, and they constitute the foundation upon which access control is provided.

1 Introduction

The web of today is by far the largest source of information ever created by man. However, the information about this information, its metadata, is not well-structured, which leads to the well-known difficulties in finding what one is looking for. The semantic web is an international effort to improve this situation by providing machine-interpretable expressions for relations between and statements about a variety of resources.

However, the strength of the semantic web approach will not be fully apparent until all resource producers – large as well as small – will express information about these resources as a natural part of the creation process. In order for this to happen, they must be stimulated to supply the neccessary information, which requires not only good technology but also convenient and user-friendly metadata applications that cater to existing needs.

A necessary requirement for such applications is a good metadata storage system, where metadata is accessed, i.e. created, stored, retrieved and updated. The kind of access that is provided has an important impact on the simplicity and the efficiency of the application. Moreover, the kind of access that would be most helpful for an application depends on the choice and usage of metadata, which in turn is a consequence of the purpose of the application. Hence, when defining a framework that will support many different applications the granularity of access cannot be tied to a specific metadata standard.

There are many storage systems for semantic web metadata, for instance Sesame [11], RDFSuite [6] and Redland [9]. These systems [8] focus on supporting

RDF Schema [10], on effective query capabilities, inference, as well as on how to store large quantities of metadata. However, as described in section 3.1 there are several aspects of metadata access that these systems do not address.

In this paper we describe the design of the *Standardized Contextualized Access to Metadata(SCAM)*[1] *framework* that provides a basis upon which different applications can be built [23]. The SCAM framework is developed as an open source project and it supports applications in storing and accessing metadata. The design of SCAM is derived mainly from the demands of applications such as archives and personal portfolios. Most prominently, SCAM provides access to sets of semantic web metadata, i.e. not to individual triples but to *sets* of triples. Like other storage systems, SCAM also provides search capabilities. To help applications protect the integrity of the metadata, basic access control is provided.

In Section 2, we briefly discuss the SCAM framework from a more technical perspective in order to provide a setting for the specific design issues discussed in this paper. In Section 3 we look at the basic system design of SCAM, starting by listing the requirements behind it. In Section 4, the issue of metametadata is treated. In Section 5, we present two specific applications that have been built upon SCAM and how they make use of the functionality. Finally, in Section 6 we provide a conclusion and future work.

2 The SCAM Framework

In this section we will see how the SCAM framework is implemented as a J2EE[2] application using the JBoss[3] platform. There is documentation on its homepage[4] where more technical details can be found. SCAM can be divided into the repository, the metadata storage and the middleware, where the application logic is placed.

2.1 Repository

The repository relies on Jena2 [3] as a triple store, which in turn connects to various Relational Database back-ends. The following five Enterprise Java Beans (EJB) provide all the functionality of the repository.

- **AdministerBean – supports the administration of SCAM.** Through this bean users, groups (see Section 3.5) and containers for metadata are administered.
- **StoreBean – supports the access to metadata sets.** The access to metadata records and containers of records are discussed in detail in Section 3.3 and 3.4.

[1] The previous full name of SCAM, *Standardized Content Archiving Management* wich occurs in previous publications, was changed since it was slightly misleading.
[2] Java 2 Platform, Enterprise Edition (J2EE).
[3] JBoss is an application server `http://www.jboss.org/index.html`
[4] `http://scam.sourceforge.net`

– **SearchBean – supports the querying of metadata sets** The query capabilities are discussed in Section 3.6.
– **ManifestBean – supports the organization of resources** Many applications need to provide hierarchical views of resources. We have chosen to implement support for a specific kind of organization, the metadata part of IMS Content Packaging [2], mainly because of its wide acceptance in the learning community. To be able to store the IMS content packaging metadata within SCAM, we have implemented a RDF-binding for it.
– **ContentBean – supports the storage of content** The ContentBean provides some useful functionality for handling content together with its associated metadata. With *content* we mean only those digital representations of resources that are administered within SCAM. Using the ContentBean, a simplified WebDAV [15] has been implemented, where the content resources are accessed in accordance with the IMS Content Packaging hierarchy described above.

2.2 Middleware

The main purpose of the middleware is to provide HTTP access to the repository. In fact, it is suitable to consider SCAM as a Model View Controller (MVC) application framework [24], where the repository is the model and the middleware provides the view and controller. The configurable controller that SCAM provides is somewhat similar to the Struts framework[5]. We are considering moving towards a more widely spread solution such as Struts or more recent initiatives such as JavaServer Faces[6]. The view has to be defined by individual applications that build on SCAM by using JavaServer Pages and Taglibs[7]. Some *utility taglibs* are provided in order to simplify the management of metadata. One of the utility taglibs is a *metadata form generator*. The forms can be configured to be either presentations or editors for metadata – according to a specific application profile. This *configurable form system* is called SHAME[8], and is briefly discussed in [13]. A more detailed report on the SHAME system is in progress.

Stand-alone applications work directly against the repository, and hence ignore the middleware altogether. Here we will not consider the middleware further. We refer to the SCAM homepage for its documentation.

3 Basic System Design

In this section we will discuss what constitutes the basic design of the framework, i.e. the management of metadata. This corresponds to AdministerBean,

[5] A framework for building java web applications can be found at `http://jakarta.apache.org/struts`
[6] `http://java.sun.com/j2ee/javaserverfaces`
[7] Taglibs are a part of the JavaServer Pages, which can be found at `http://java.sun.com/products/jsp`
[8] Standardized Hyper-Adaptable Metadata Editor

StoreBean and SearchBean in the repository part of SCAM. We will begin by listing the requirements. Each requirement will then be analyzed, together with the corresponding design decisions.

3.1 Functional Requirements

Since the SCAM system is a framework upon which several different types of applications can be built, the design should support, but not be limited to, individual applications. However, the requirements that motivate the design have been inspired by practical needs, most prominently by the needs of archiving and personal portfolio applications. In such applications many users express metadata around individual resources in a web-like environment. The metadata is then provided through various interfaces, typically via exploration or search. Furthermore, the metadata is allowed to change often, not only updated but entirely new metadata constructs are allowed to be added on the fly.

The six requirements below will be treated in the following subsections.

1. SCAM should be independent of application profiles, i.e. not be limited to a set of metadata elements adapted to the needs of a specific community.
2. SCAM should support the ability to work with metadata records, i.e. to work with metadata centered around individual resources.
3. SCAM should support the administration and separation of metadata between different contexts. Specifically, contexts should:
   (a) provide simple mechanisms for exchanging metadata with other systems, e.g. another SCAM system, another metadata management system, an external storage system.
   (b) allow the expression of different opinions that if kept together would be inconsistent.
4. SCAM should support access control of both metadata records and contexts.
5. SCAM should provide search capabilities of metadata records.
6. SCAM should provide support for vocabularies.

Before continuing let us consider some existing RDF stores with respect to these requirements. RDFSuite [6] fails to fulfill 1 (since it can only store statements if there is a schema), as well as 2, 3 and 4 (since there is only one RDF model to upload statements into), but it does manage 5. Redland [9], Sesame [11] and Jena[3] fulfill 1, 3, and 5 more or less directly. Since SCAM builds on Jena and fulfills 2, 4 and to some extent 6 via conventions and algorithms, SCAM could in principle be built on top of either Sesame or Redland. This would probably result in a similar amount of work.

3.2 Independence of Application Profiles

The concept of an application profile has been developed and clarified over time. We have chosen to follow the definition in [12]:

> "An application profile is an assemblage of metadata elements selected from one or more metadata schemas and combined in a compound schema. Application profiles provide the means to express principles of modularity and extensibility. The purpose of an application profile is to adapt or combine existing schemas into a package that is tailored to the functional requirements of a particular application, while retaining interoperability with the original base schemas. Part of such an adaptation may include the elaboration of local metadata elements that have importance in a given community or organization, but which are not expected to be important in a wider context."

We think that the use of the expression "metadata schema" is somewhat too general for our discussion around application profiles. Therefore, from now on, we will use the more narrow expression *metadata standard*, which will refer only to metadata schemas with well defined sets of metadata elements – as introduced by specific standardization bodies. Two examples of metadata standards are Dublin Core [5] and IEEE/LOM [17]. An example of an application profile is CanCore [1], which uses metadata elements from the metadata standard IEEE/LOM.

Since the intention with SCAM is to be independent of application profiles, which may collect metadata elements from several metadata standards, we argue that SCAM should be independent of metadata standards as well. This is not only a technical consequence, but reflects our firm belief in the benefits of a multitude of independently developed metadata standards and application profiles covering different needs. It is not an option to wait for this diversity to be incorporated as extensions to one fundamental metadata standard. Hence, choosing any specific metadata standard today (or at any specific point in time) would limit the expressibility of the applications built upon SCAM.

In our design of SCAM, we have discarded the approach to support several metadata standards separately, and more specific application profiles on top of those, since this will either create numerous problems with incomplete metadata records or result in a "combinatorial explosion" of the many translations between them.

Instead we have chosen a solution offered by semantic web technology, more specifically by RDF(S) [16] [10], which provides a common meta-model, within which metadata elements – and hence metadata standards – can be expressed with well defined syntax and semantics. The choice of RDF is perhaps best motivated when it comes to defining application profiles in terms of combinations and reuse of bits and pieces from previously existing application profiles and metadata standards. This is supported by the fact that the RDF-bindings of major metadata standards – such as e.g. IEEE/LOM and Dublin Core – are specifically designed in order to allow such combinations and reuse of their constituent parts.

Today there already exists RDF bindings of Dublin Core and vCard and the RDF binding[20] of IEEE/LOM is nearly finished. These three metadata standards have provided the basic building blocks from which we have constructed the various application profiles on top of SCAM. When these "standard building

blocks" have been unable to provide sufficient expressive power, we have invented new metadata constructs or – if possible – extended the existing ones via the RDF vocabulary description language (RDFS).

3.3 SCAM records

We have defined SCAM records so they are suitable for managing the metadata of a single resource, independently of which application profile that is used. With "manage" we here mean create, update, store, retrieve or present an entire metadata record. A prerequisite requirement has been that the granularity of a SCAM record should be comparable to that of the document-centric approach for exchanging metadata records of a fixed application profile.

A SCAM record is defined to be the *anonymous closure* of a RDF graph computed from a given non-anonymous RDF resource. The anonymous closure is computed by following chains in the natural direction, from subject to object, of statements until a non-anonymous RDF resource or RDF Literal is found. Furthermore, each anonymous resource is not allowed to occur in more than one SCAM record. Any anonymous resource that fails to follow this requirement is duplicated when imported into SCAM. Within a RDF graph, a SCAM record is uniquely identified by the URI-reference of the non-anonymous RDF resource from which the anonymous closure is computed.

SCAM records coincide – except with respect to reifications – with Concise Bounded Resource Descriptions as defined in URIQA[9].

URIQA also defines a retrieval mechanism together for Concise Bounded Resource Descriptions, forcing them to always be retrieved from an authoritative server, i.e. the server specified by the host part of the resource's URI-reference. This approach does only allow one set of metadata for a specific resource, clearly conflicting with the idea of a metadata ecosystem as presented in [21]. Hence, even though we could implement the URIQA protocol on top of SCAM, this is not a preferred way to go, since it would only allow access to a subset of all SCAM records on a given server.

In fact we have chosen not to commit SCAM to a specific protocol for accessing SCAM records since there is no mature standards for this yet. We are keeping a close watch on initiatives such as URIQA, the RDF WebAPI in Joseki [4] and the metadata retrieval support in WebDAV [15].

These SCAM records have several useful properties:

1. In terms of RDF statements, SCAM records are always well-defined and disjoint, which avoids problems when updating, removing etc.
2. SCAM records connect the administration of metadata constructs with things that have a public identifier. This means that things that have no identifier cannot be administered on their own.
3. SCAM records with different identifiers can co-exist in the same RDF graph.
4. RDF query engines need not respect the borders of SCAM records, which allows more advanced queries.

[9] The URI Query Agent Model, see `http://sw.nokia.com/uriqa/URIQA.html`

5. SCAM records have a granularity for management that matches most application profiles, such as LOM, Dublin Core etc.

A drawback of SCAM records is that there are RDF graphs that aren't accessible or expressible. For example, a statement with an anonymous resource as subject that does not occur anywhere else and with a non-anonymous resource as object cannot be reached through SCAM records. There are also statements – potentially small graphs – that consist solely of anonymous RDF resources and RDF literals that simply have no natural connections to any SCAM records.

From a more practical perspective, statements that might be interesting to include in a SCAM record include anonymous reifications, as a way to express metametadata about individual statements. See Section 4 for a more thorough discussion of reifications and metametadata.

3.4 SCAM Contexts

When administrating a SCAM system, it is often not enough to work on the level of SCAM records, simply because they are of too small granularity. For example, many common administrative operations would involve hundreds or thousands of SCAM records. Hence, we introduce a *SCAM context* to denote a set of SCAM records that somehow belong together. Exactly which SCAM records that belong together depends on the application at hand.

Another requirement on SCAM is to allow several SCAM records for the same resource. We will call such SCAM records *siblings*. For example, an application serving many users may contain several different opinions about the same resource; opinions that will have to be kept separate if metametadata about their origin are to be preserved, e.g. access rights, author etc. To be able to refer to individual siblings, we need – apart from the resource – some kind of marker to distinguish them.

Now, with knowledge about SCAM records, siblings and SCAM contexts, let us take a closer look at how we actually store them as RDF graphs in SCAM. We have considered three basic approaches:

1. Keep all SCAM records together in one RDF graph and use some markers telling what SCAM contexts they belong to.
2. Keep all SCAM records belonging to the same SCAM context together in one RDF graph.
3. Keep every SCAM record in a separate RDF graph and use some marker to tell what SCAM contexts that the SCAM record belongs to.

The first approach is immediately disqualified since it does not allow siblings. The second approach allows siblings but forces them to be in separate SCAM context. Consequently, all SCAM records are uniquely identified within a SCAM context, no separate marker is needed. The last approach opens up for having several siblings within a single SCAM context. A separate marker is needed to distinguish siblings.

We have chosen the second approach, where SCAM contexts contain only one sibling for a single resource. This has the advantage of being simple and no extra marker is needed. See Figure 1 for a illustration of how SCAM contexts and SCAM records relate. On the other hand it is clear that this choice is a limitation on the use of siblings in SCAM contexts. However, without this limitation a SCAM context cannot always be collected into a single RDF graph. The reason is that if you want to preserve the integrity of siblings in a single RDF graph, triples have to be reified and be 'owned' by one or several SCAM contexts. To be 'owned' would have to be expressed on the reifications by some property that is yet to be invented. Hence, we would end up with a situation where import and export of metadata would either be *non standardized* – via ownership on reifications – or more *complicated* – several RDF graphs would represent a single SCAM context.

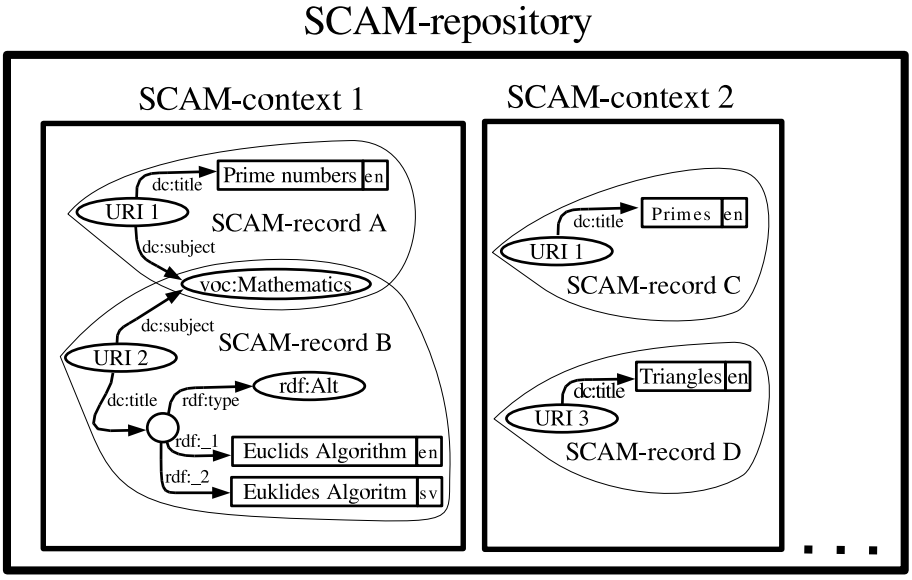

Fig. 1. A SCAM repository has SCAM contexts as separate RDF graphs, which are further divided via the anonymous closure algorithm into SCAM records. *SCAM records A* and *C* are siblings since they both express metadata for a resource identified via the URI reference *URI1*.

3.5 Access Control Lists

SCAM provides access control on all SCAM records. The access control is focused on the access to the metadata in the SCAM record and, if present, to the resources that are described therein. We use a form of Access Control Lists

(ACL) for describing the rights. Our solution is inspired by – but not derived from – ACLs in file systems such as NTFS[10], AFS[11] etc. In the future we will take a closer look at the access control protocol specified by WebDAV [15] and initiatives for expressing ACLs in RDF such as W3C ACLs[12].

The basic functionality of an Access Control List is simple. It provides a list of *principals*, i.e. users or roles, paired with intended rights. For SCAM the access control list states the access rights and user privileges only in terms of right to read and/or write. Other rights might be considered, e.g. the right to lock, change metametadata etc. Due to performance and complexity reasons, in the current representation there is no negation of permissions.

Currently SCAM only supports groups for roles. The groups in SCAM can be defined by how they relate to the users:

1. Every user belongs to a set of groups.
2. Groups are defined by which users they contain, no empty groups are allowed, which means that you have to define some users before you introduce the groups they should belong to.
3. Groups cannot be defined in terms of other groups.

Even though SCAM currently is limited to groups, the representation of the ACLs could as well be used for expressing things like Role-Based Access Control (RBAC) as defined in [14]. See [7] for a comparison between RBAC and the more basic approach of ACLs that we have chosen for SCAM. RBAC was not chosen for SCAM, since it was deemed unnecessarily complicated for our current needs. More specifically:

1. It was required that access should be possible to grant to individual users without creating a role for every user.
2. It was not required to support the change between sessions of roles assigned to a user. For example, the ability of the same user to log in as an administrator or a regular user depending on the situation was not a requirement.

Another – somewhat independent – restriction is that a SCAM context must always be "owned" by a principal, i.e. a user or a group. A user will have all permissions on all SCAM records in a SCAM context, if either the user is the owner of that context, or the owner is a group and the user belongs to that group. Furthermore, there are two principals in SCAM who have special meaning, the "Administrator" and the "Guest". The group Administrator always overrides the access control, which means that all users in that group has the permission to perform "everything". A Guest is a fictional principal that is used by unauthenticated users. If the Guest is included in an ACL it means that everyone has the given permission.

[10] NTFS is the Windows NT file system developed by Microsoft.

[11] Andrew File System is a distributed file system developed at Carnegie-Mellon University.

[12] `http://www.w3.org/2001/04/20-ACLs`

Since the SCAM records are represented in RDF, we have chosen to express the ACLs in RDF as well. Figure 2 shows the RDF structure of an ACL in SCAM.

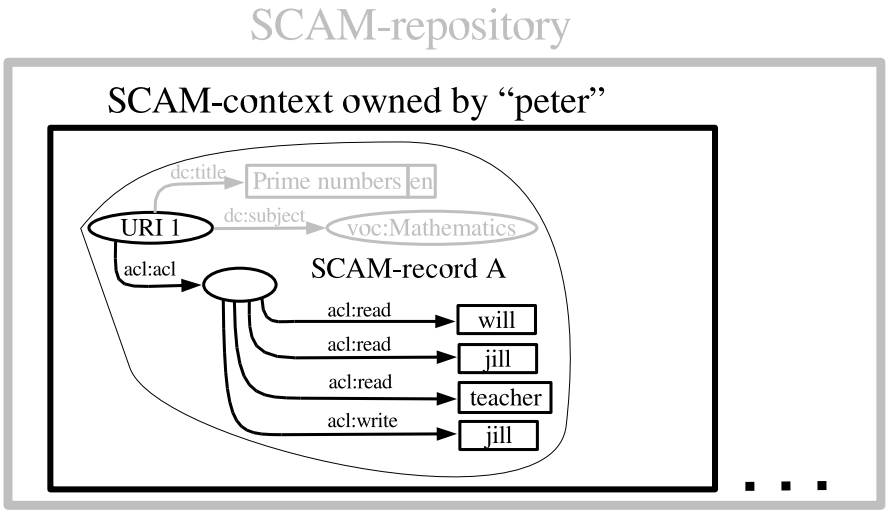

Fig. 2. A user named *peter* is the owner of the SCAM context and consequently has full access rights to the SCAM records within it. All the other principals e.g. *will*, *jill* and *teacher* have read-access, in addition *jill* also has write-access.

The ACL RDF graph is expressed via an *anonymous resource*, which allows it to be automatically included into the corresponding SCAM record. The semantics of pointing to the ACL directly from the resource is somewhat dubious (see our discussion on metametadata in Section 4).

3.6 Queries

In order to support queries against metadata, we had to make some "strategic decisions" concerning the following two questions:

1. Against what portions of metadata should the queries be allowed to execute?
2. What query languages and result types should be supported?

In response to the first question, we have chosen to allow queries to be executed against *one* SCAM context at a time – or *all* SCAM contexts *at once*. To search against several – but not all – SCAM contexts at once is not supported (due to technical limitations that we expect to be able to overcome soon). Another technical limitation has forced us to skip the check for access restrictions expressed in the ACLs when executing the search.

In response to the second question, we have chosen to support simple free-text search, RDQL [18] and QEL [22].

The two first query languages are motivated by the fact that typical applications provide two levels of search, first simple free-text search and second, a more advanced search on specific metadata elements. With free-text search we mean sub-string matching of literals – disregarding which metadata elements they belong to. Strictly speaking, free-text search does not require a special treatment, since it can be expressed in the other query languages, rather it is provided for simplicity and efficiency reasons. Until recently, advanced search has been achieved through the use of RDQL, a rather restricted SQL-like language that nevertheless is expressive enough to perform most simple tasks. The reason for supporting RDQL is because it is the query language supported by Jena2, which is the RDF API we have used internally. QEL is a very powerful datalog-like language that is part of the Edutella project [19] which aims to provide a peer to peer network where semantic web metadata can be searched for. The QEL language is quite powerful, it provides conjunction, disjunction, rules, outer join, built in predicates − linear and general recursion. The implementation that SCAM uses does not yet support linear and general recursion.

With support for QEL, metadata in individual SCAM repositories can be made searchable over Edutella and consequently discovered by people without previous knowledge of the systems location. We foresee that QEL will to some extent replace RDQL for doing local searches as well.

3.7 Support for Vocabularies

Most applications make use of vocabularies. In many cases queries and graphical user interfaces require the presence of such vocabularies in order to work as expected. The obvious design, to keep vocabularies together with other metadata, is seldom preferable since it complicates administration and update. SCAM contexts dedicated for managing one or several vocabularies is probably a better alternative. A necessary complication is that the query engines would somehow have to be aware of this. For expressing vocabularies, in most cases RDF Vocabulary Description Language (RDF Schema) is sufficient.

In order to understand class and property hierarchies, support for inference should be added. Currently we have satisfied such needs by forward chaining (i.e. calculating and storing implicit triples) in copies of SCAM contexts. The vocabularies have been merged into these copies, and all queries are then executed against them. This is a temporary design that we are investigating alternatives for.

4 Support for Metametadata

Metadata is data about data, consequently, metametadata is data about metadata. E.g. the date when a SCAM record was created or modified, access control lists controlling the access to a SCAM record or an entire SCAM context.

In RDF there is a basic mechanism for creating metametadata called *reification*. A reification is a resource that is treated as a placeholder for a statement. Statements around a reification are implicitly statements around the statement that has been reified. Furthermore, reified statements can be collected into collections, which provide handles for expressing statements on sets of other statements without repeating them. In principle, reification is very precise – since it allows you to express statements about any set of statements. Unfortunately, from a more practical perspective it is a rather clumsy mechanism, especially if the most common case is to express statements about all the statements in a RDF graph. (This adds at least 4 new statements for each statement to be reified).

In our case, we would like to express metametadata around all statements within every SCAM record. Currently, we express the metametadata directly on the resource identifying the SCAM record. We realize that the semantics of such statements may be wrong, and that they may conflict with the chosen application profile. Lets consider two problematic situations:

1. Metametadata such as the date when a SCAM record was created uses the Dublin-Core-created property directly attached on the resource. The semantics – as expressed by Dublin Core – says that the resource itself was created at the specific date, not – as we intended – the SCAM record.
2. Applications – building on SCAM – that manage both metadata and data, incorrectly let the ACLs apply to the data as well. Unfortunately, the current design of ACLs carries with it the natural interpretation that they should apply to the data (content) – and not to the metadata (SCAM record) as defined. Hence, the ACLs presently used for SCAM records are in the way for the data-access-controlling ACLs.

Both these situations could be remedied by defining new properties with the correct semantics. However, this would not allow reuse of existing standardized metadata elements on the metametadata level.

Instead, we are redesigning SCAM so that metametadata properties apply to an intermediate anonymous resource pointed to via a 'metametadata' property from the resource identifying the SCAM record. Consequently, ACLs can be provided on both the metadata and the resource (the latter only when the resource represents some content that is within the control of the SCAM content extension system). See Figure 3 for an illustration.

If there is a need to express metametadata at a higher granularity level, there are several possible solutions. For example, a specific SCAM context can be used in order to express metametadata on SCAM contexts.

At the other end of the scale, there might be a need for more fine-grained metametadata. It is quite easy to imagine scenarios where there is a need for metametadata on specific parts of the SCAM record. For example, the Dublin Core description property could in some situations need a date of modification or even a complete version history with information about who the changes was made by.

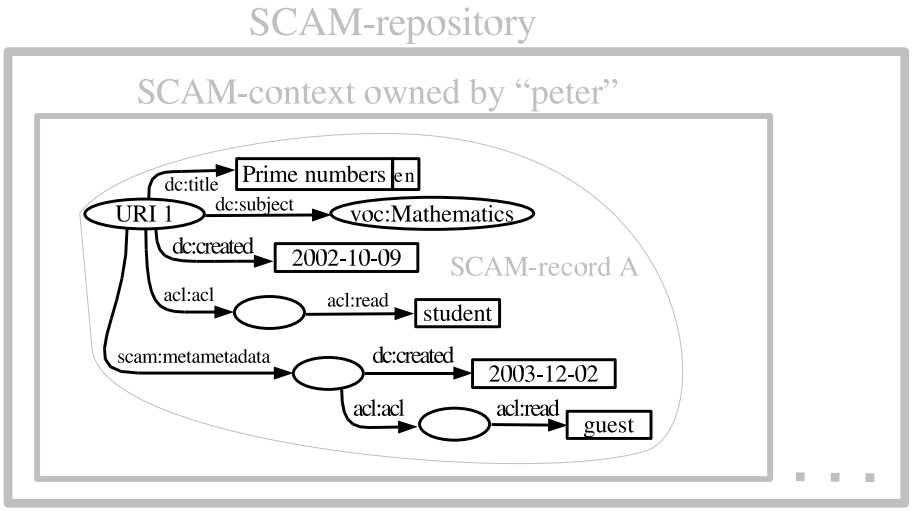

Fig. 3. The resource, which can be thought of as a file, has metadata that says that it is readable by all students, was created “2002-10-09”, is about “mathematics” and has a title that reads “Prime numbers”. The metametadata on the other hand says that the metadata was created “2003-12-02 and is readable by everyone.

In this case the right way to go is probably to use reification directly since the alternative, to invent specific metadata constructs where metametadata is integrated with the metadata, has as a consequence that established metadata standards such as Dublin Core and IEEE / LOM cannot be used.

For this to be reasonably efficient, especially in combination with the special case of version history, there would have to exist specific support in SCAM, something that has yet to become a priority.

5 Applications

There are currently about ten different applications that have been built upon SCAM[13]. For example, the *SCAM portfolio* provides storage of metadata on resources such as web resources, uploaded files, traditional books or abstract ideas. The resources can be organized into folders that can be shared with others. Just as SCAM itself, the SCAM portfolio is not bound to a specific application profile. Instead there is a configurable web user interface (SHAME) mentioned above, where editors, exploration view and query interface can be configured. The aim is to make this configuration so simple that the end-user could create new application profiles from a toolbox whenever new kinds of resources are encountered.

[13] For a more complete list see SCAM’s homepage `http://scam.sourceforge.net`

A SCAM application that has a fixed application profile is *the Digital Media Library*[14] *of The Swedish Educational Broadcasting Company (UR)*. The main goal is to give public access – via a web interface – to rich metadata for all resources that UR produces. These resources include TV and Radio programs, series of those, related resources such as teacher tutorials, web sites etc. The resources are expressed as application profiles, where standard metadata elements and vocabularies are used when possible. We have used, IEEE / LOM, Dublin Core, DC-terms, VCard plus some national vocabularies regarding educational material. The interface should provide rich search capabilities as well as possibilities to find related resources via containments in series, explicitly given relations etc. Searches are performed via vocabularies. The resources, i.e. the programs themselves, are administered outside of SCAM.

6 Conclusions and Future Work

In this paper we have descibed the design of the SCAM framework which helps applications to store and access metadata on the semantic web. We have defined a SCAM record as the anonymous closure from a resource and claim that in most cases this coincides with how metadata is accessed according to an application profile. Furthermore we have introduced SCAM contexts as a way to handle large amounts of SCAM records. SCAM contexts can be imported and exported as RDF graphs providing a mechanism for backup, communication with other systems etc. Moreover, we have defined access control on the level of SCAM records and query capabilities that can either span all SCAM contexts or be limited to a single SCAM context. We have also seen how different applications can make use of the SCAM framework for different purposes.

Future work will include support for inference and special treatment of vocabularies / ontologies. We will include specific support for metametadata, which e.g. will change the way that ACLs are applied. The ACL model will be revised and extended with e.g. time aspects and authentication by formal deduction rules. Furthermore, we plan to do a quantitative investigation of SCAM's performance and storage capabilities.

7 Acknowledgements

We are indepted to Jan Danils and Jöran Stark for their programming efforts and to Mikael Nilsson for sharing his expertise on metadata standards. The SCAM project has several stakeholders, the support of which we gratefully acknowledge. Prominent among them are the Swedish National Agency for School Improvement, the Swedish Educational Broadcasting Company, the Swedish National Center for Flexible Learning,Uppsala Learning Lab and the Wallenberg Global Learning Network. The financial support from Vinnova is also gratefully acknowledged.

[14] `http://www.ur.se/mb`

References

1. Canadian Core Learning Resource Metadata Application Profile. http://www.cancore.org.
2. IMS Content Packaging. http://www.imsproject.org/content/packaging.
3. Jena - A Semantic Web Framework for Java. http://jena.sourceforge.net.
4. RDF WebAPI. http://www.joseki.org/protocol.html.
5. The Dublin Core Metadata Initiative. http://dublincore.org.
6. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In *The Second International Workshop on the Semantic Web - SemWeb'2001.*
7. B. Barkley. Comparing simple role based access control models and access control lists. In *Proceedings of the second ACM workshop on Role-Based Access Control*, pages 127–132, Fairfax, Virginia, United States, 1997.
8. B. Beckett and J. Grant. Mapping Semantic Web Data with RDBMSes. http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report.
9. D. Beckett. The Design and Implementation of the Redland RDF Application Framework. In *Proceedings of the tenth World Wide Web Conference*, 2001.
10. D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/2002/WD-rdf-schema-20020430.
11. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF.
12. E. Duval, W. Hodgins, S. Sutton, and S.L. Weibel. Metadata Principles and Practicalities. D-Lib Magazine Vol. 8 No. 4, April 2002.
13. H. Eriksson. Query Management For The Semantic Web. http://kmr.nada.kth.se/papers/SemanticWeb/CID-216.pdf.
14. D.F. Ferraiolo and D.R Kuhn. Role Based Access Control. In *15th National Computer Security Conference*, 1992.
15. Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP Extensions for Distributed Authoring – WEBDAV, 1999.
16. K. Graham and J. J. Carroll. Resource Description Framework (RDF):Concepts and Abstract Syntax. http://www.w3.org/TR/rdf-concepts.
17. Learning Technology Standards Comittee of the IEEE:. DraftStandard for Learning Objects Metadata IEEEP1484.12.1/D6.4, June 2002.
18. L. Miller, A. Seaborne, and A. Reggiori. Three Implementations of SquishQL, a Simple RDF Query Language. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 423–435, 2002.
19. W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. EDUTELLA: A P2P Networking Infrastructure Based on RDF. In *Proceedings of the 11th World Wide Web Conference*, 2002.
20. M. Nilsson, M. Palmér, and J. Brase. The LOM RDF binding - principles and implementation. ARIADNE Conference 2003.
21. M. Nilsson, M. Palmér, and A. Naeve. Semantic Web Meta-data for e-Learning - Some Architectural Guidelines. In *Proceedings of the 11th World Wide Web Conference*, 2002.
22. M. Nilsson and W. Siberski. RDF Query Exchange Language (QEL) - concepts, semantics and RDF syntax. http://edutella.jxta.org/spec/qel.html, 2003.
23. F. Paulsson and A. Naeve. Standardized Content Archive Management – SCAM. IEEE Learning Technology newsletter Vol 5, Issue 1, January 2003.
24. Sun Microsystem Inc. *Web-Tier Application Framework Design.* 2002.

All URLs that are referenced in this paper have been accessed on 2004-01-12.