A Mashup-friendly Resource and Metadata Management Framework

Hannes Ebner, Matthias Palmér

School of Computer Science and Communication
Royal Institute of Technology (KTH), Sweden
{hebner, matthias}@csc.kth.se

Abstract: Mashups and mashed up Personal Learning Environments require easy to use frameworks to support the ease of creation of effective services. The focus of this paper[1] lies on establishing a generic and mashup-friendly resource and metadata management. The assumption is that if we can find an appropriate level of generic functionality, the development of targeted tools (e.g. e-portfolios, PLEs, etc) will become a matter of user interface design and specialization. We hope that such a framework does not result in a single implementation but rather a wide variety of interoperable systems that leverage plenty of functionality. In this paper we look at already existing standards and initiatives and show why they are not sufficiently generic. We propose a framework and take recent developments into consideration. We also show an implementation and introduce a tangible use case.

Introduction

A very basic element of the Web 2.0 and Social Software is a mashup. A mashup is a (web) application that combines several data sources into one user interface or result. To make mashup creation easy, most applications provide a public API, building upon standard protocols, such as the Hyper Text Transfer Protocol (HTTP), and standard data formats, like JavaScript Object Notation (JSON) and the Extensible Markup Language (XML). A Personal Learning Environment (PLE) can be seen as a kind of a mashup. It makes the composition of a personal environment possible; built out of several (not necessarily connected) systems, tools or just data sources. Such a collection of personally aligned fragments represents the freedom of choice for learners within PLEs. A PLE does not necessarily have to be a web application, it can also exist on the desktop. It may consist of production tools (e.g. wikis and blogs), feed readers, communication and collaboration tools, social networking services, storage services, identity management, and so forth. An e-portfolio is a common component of a PLE.

On a different level, to make all this work together, some kind of resource and metadata management is needed. This means that we have to differentiate between the resource itself, its descriptive information (metadata), and administrative information such as access control, modification date, and cache control. In addition,

1 This work has been carried out with financial support from the EU eContent*plus* project Organic.Edunet (ECP-2006-EDU-410012), which the authors gratefully acknowledge.

we also need a differentiation between digital and non-digital resources. This approach ensures a very flexible way of managing, integrating, and reusing resources or just information about them. Splicing everything together in a simple way requires simple and powerful techniques. RESTful Web Services [1] in combination with asynchronous JavaScript and XML (AJAX) are widely used state-of-the-art technologies which allow for quick and efficient querying and modification of resources, as well as communication between services.

In order to support such a mashed up PLE infrastructure, the new version 4 of the Standardized Contextualized Access to Metadata (SCAM) framework [2] is targeted towards such environments. Instead of using an own specific data and metadata layer, applications can rely on SCAM and take advantage of its flexibility. SCAM provides a unified mechanism of accessing the managed resources and its descriptive information, which might be (re)used by any number of tools. SCAM can be seen as the least common denominator between "mashed up" applications regarding resource and metadata management.

Successively we take a look at related work, where we point to related standards and initiatives, which we discuss in the context of mashups and PLEs. Thereafter we depict a generic design of a resource and metadata management system, which also forms the basis of SCAM 4. In the following section "Implementation" we show how it is implemented, and present a use case of an application using the framework. The last section "Conclusions" reconsiders the findings during the development process and gives a perspective on applications of the framework and future developments.

Related Work

There are several standards and initiatives aiming for resource and metadata management and exchange. We briefly summarize the most important ones.

A Content Package (CP), and in particular IMS CP [3], is used to organize and package resources and describe them with metadata. The IMS CP format has been reused particularly within IMS and SCORM, for example IMS ePortfolio [5], IMS Learning Design [6], and SCORM Content Objects [4]. The standard is targeted mainly towards transfer between systems rather than providing simple access to the packaged resources. Hence, IMS CP is not optimal from a mashup perspective.

WebDAV [7] extends HTTP with functionality which allows for collaborative file management. It basically makes the WWW writable, and has support for collections, resources and links. Additional extensions enable, among other things, searching and versioning, which are important for the management of resources. Unlike HTTP, it has support for resource properties, which can be seen as limited metadata. However, reusing the same resource, describing it in different contexts, or just providing extensive metadata is not possible.

The Atom Syndication Format (Atom) [8] is based on XML and mostly used by web feeds. The complementary publishing protocol AtomPub [9] is used for creating and updating resources on the web. The basic concepts behind AtomPub are collections, workspaces, and services. A service is a grouping of workspaces,

whereas a workspace is a grouping of collections. A collection is a feed containing entries, with describing metadata for each entry. The inherent service discovery and HTTP enable a RESTful way of managing resources. There is no explicit access control except for the HTTP authentication methods, no search functionality, and no support for references, which makes it impossible to provide remote metadata. In addition and perhaps most important, creation or modification of available services, workspaces or collections is outside the scope of the protocol.

SCAM may in the end support several of these standards as a complement, however none of them do really match up for a sound architecture which supports resource and metadata management as well as interoperability and easy integration (i.e. mashups) through standards-driven design.

Discussion

The primary objective of this paper is to introduce a mechanism to manage resources and their corresponding metadata. However, the concept of resources is rather vague and we need to clarify what we mean. Resources as regular files and links to web content are commonplace. A wider perspective includes books in libraries, physical persons, calendar events, comments, concepts, and so forth. Since we aim for supporting mashups and have decided to follow the principles of REST [1], it makes sense to adhere to the definition used by the W3C Technical Architecture Group (TAG) [10] as stated in the Architecture of the World Wide Web, Volume one [11] which says:

> *By design a URI identifies one resource. We do not limit the scope of what might be a resource. The term "resource" is used in a general sense for whatever might be identified by a URI. It is conventional on the hypertext Web to describe Web pages, images, product catalogs, etc. as "resources". The distinguishing characteristic of these resources is that all of their essential characteristics can be conveyed in a message. We identify this set as "information resources."*

This definition allows us to manage any resources that are identifiable via URIs, both "information resources" (digital resources) as well as other resources that have no digital representation. Whether a resource can be retrieved or not can be detected by trying to retrieve the resource over HTTP and inspection of the returned message. There is a recommendation by the W3C [14] on how to answer such requests. However, to follow this approach all the time is both inefficient and error prone. Servers can be down or not following the recommendation. Instead we propose that SCAM manages those pieces of information. Even if it is known that a resource is an information resource, it is unknown which format this resource is available in. This should be managed via one or several MIME types [16]. Unfortunately, the definition of resources from the W3C TAG [10] is not sufficient for our needs. For example does it not help in deciding how to distinguish a link from an uploaded file, as both can be "information resources". If a resource is managed outside the current system it should be considered to be a link. It is even possible to make a distinction

whether the metadata for the resource is managed in SCAM. Hence, we introduce the term *reference* to denote links where the metadata is managed outside of SCAM.

We introduce the concept of an entry which provides necessary information regarding the resource and the metadata for successful management in SCAM. With this definition it is more appropriate to think of a SCAM installation consisting of entries rather than of pairs of resources and metadata. Where to draw the line between what should go into the entry and what should go into the metadata is a question of pragmatism and semantics. As both the metadata and the entry will use RDF, we can build upon established standards and common practices as well the basic semantics of RDF. Providing access control on resources can be conveniently solved by expressing permissions inside the entry expression. When expressing these permissions, relevant users, groups or roles need to be available. To avoid the need for introducing additional complexity, we suggest to expose this information as specific built in resources. Other system specific entities such as ontologies, types, various configurations, etc. may also be exposed as built in resources. As these will appear as full entries with specific access control restrictions, it provides a powerful bootstrap mechanism that we envision will be used extensively.

Design

We introduce three different kinds of types that are more or less independent of each other. The *representation type* defines whether a resource has a digital representation or not. The *builtin type* indicates whether a resource gets a special treatment within SCAM. The *location type* indicates if neither, one, or both of the entry's resource and metadata is maintained within SCAM.

Representation type	Location type	Builtin type
Information resource – the resource has a representation, in the repository or elsewhere. *Resolvable information resource* – the resource is an information resource but requires a resolvable step, e.g. through a look-up procedure that might be protocol specific such as urn:path or doi. *Unknown* – the representation type of the resource is unknown. *Named resource* – the resource is not an information	*Local* – both metadata and resource are maintained in the entry's context. *Link* – the metadata (but not the resource) is maintained in the entry's context. *Link reference* - the resource as well as the metadata is maintained outside of the entry's context; in addition there is complementary metadata maintained in the entry's context. *Reference* – the resource and the metadata is	*Context* – a container resource which keeps track of a set of entries that should be managed together; at a minimum it provides default ownership of the contained entries. *System context* – a context that is specifically treated in SCAM. *Principal* – a user used in access control lists. *Group* – a group used in access control lists. *List* – an ordered list of entries.

resource but it can be referred to in communication but not transferred in a message.	maintained outside the entry's context.	*Result list* – a list that is dynamically generated. *None* – all other resources without specific treatment in the repository.

The builtin types deserve some further consideration. Firstly, *context* is the most important of the builtin types as it divides entries into disjoint sets. In fact, every entry is required to belong to a single context. A special rule says that access to an individual entry is decided by the context's or the entry's access control list, depending on which is most permissive. Secondly, principals correspond to the users and the groups in the system, they are managed in a special context which is referred to as the *principal manager*. Thirdly, the principal manager is an example of a *system context*. Another important system context is the *context manager* which contains all *contexts* as entries. Finally, *lists* are used to organize a set of *entries* (within one context) into a ordered lists. With the help of this terminology it is much easier to introduce the RDF design, which defines how to represent entries and common metadata in RDF, and the REST design, which defines how to interact with SCAM via HTTP.

RDF Design

The context construct was introduced as SCAM context in [28] and was at that time effectively an RDF graph. With the anonymous closure algorithm, that graph was used to detect a set of statements that where managed together, referred to as a SCAM record. In this paper we introduce entries as a replacement for SCAM records. The recent introduction of Named Graphs [19] has in part invalidated the approach of SCAM contexts and records. We took the approach to use up to three Named Graphs for each entry: the administrative entry information, the entry's metadata, and sometimes even the resource itself. To achieve an aggregation of entries into a context, the context resource is a Named Graph which contains an index of all Named Graphs for all its entries. It is encouraged, but not required, to support RDF representations of entries (see the figure for a schematic picture of the RDF expression) and common metadata in the REST API. The RDF design serves the purpose as a specification language as the semantics are well defined.

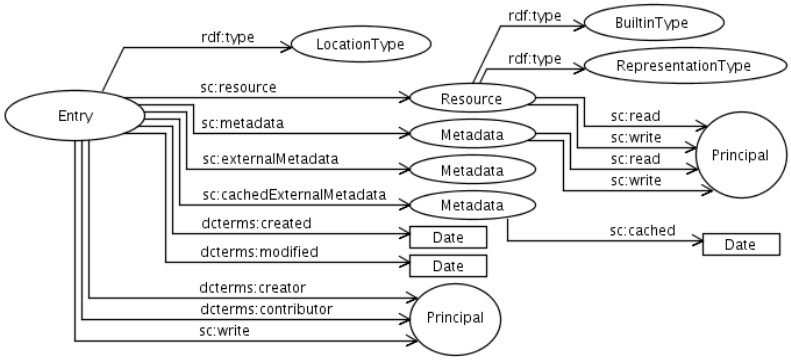

Figure: RDF design of an entry.

In contrary to the entry information, SCAM does not have an understanding of the metadata itself. This is up to the application on top of SCAM and the reason why the metadata graph cannot be generically depicted in this paper. The use case which is presented later on takes a very generic approach which allows for a flexible definition of *annotation profiles* [15] for the presentation and editing of metadata using of the SHAME library [29].

REST Design

The starting point when designing a system in a RESTful manner is to identify all things that should be accessible via separate URIs. These things are quite naturally called resources as the web is the most prime example of a REST architecture. However, as we also use the term resource in the SCAM design, this is somewhat confusing. In the following the term REST resources is used when we talk about those things that should have unique URIs in SCAM.

There are three basic kinds of REST resources in a context: resource, metadata, and entries. The following table shows the URIs and allowed HTTP operations for the three kinds of REST resources:

Operation	Method and URI
Fetch	`GET {base-uri}/{context-id}/{kind}/{entry-id}`
Modify	`PUT {base-uri}/{context-id}/{kind}/{entry-id}`
Delete	`DELETE {base-uri}/{context-id}/{kind}/{entry-id}`

base-uri is the base URI (namespace) that is specific for each system; *context-id* is an integer that uniquely identifies a context; *kind* is one of the three kinds of REST resources; *entry-id* is an integer that uniquely defines an entry within a context.

Resources which are links or references will most likely not have URIs that follow the pattern above and will probably only respond to GET requests. Furthermore, if the resource is not an information resource, although it is maintained locally, SCAM responds with a HTTP response containing a pointer to the URI of the entry, see the discussion in [14]. However, if it is a link or a reference, the response cannot be guaranteed as it depends on the configuration of the involved web server. For non-local metadata, i.e. when the entry indicates a reference, the URI will probably neither look like above nor will it work to fetch the metadata directly. In addition, the metadata might need to be converted from another format such as the XML binding of IEEE/LOM or extracted according to RDFa [17] or GRDDL [18].

In addition to the operations listed above, creation of new entries and listing of all entries in a context has to be possible. This is solved by introducing an additional REST resource "context". For searching we introduce a special service called "search". There are operations missing, for instance there is no way to create, delete or set access control on contexts. This is because contexts are treated as resources which are managed via entries in the context manager introduced above.

Implementation

The core of SCAM 4 is completely built on Semantic Web technologies, in particular the concept of Named Graphs [19]. With Sesame [20] we chose an open quad store with support for a variety of storage systems, high scalability, a flexible API, remote access via HTTP, several query languages, and a powerful extension API. The RESTful web services on top of the SCAM core are implemented using the Restlet framework [21], which also provides input to the upcoming Java API for RESTful Web Services (JSR-311) [22]. To serialize entries and metadata the formats RDF/XML, TriG [23], and JDIL [24] (based on JSON) are supported. In addition to RESTful web services, SCAM provides harvesting mechanisms using the protocols OAI-PMH [25] and FIRE/LRE [26]. The querying protocol SQI [27] is supported.

A Use Case: Confolio

The web-based e-portfolio Confolio is a use case where the internal types of SCAM are mapped to specific features. Contexts are used as portfolios, lists are used as folders, and entries are items in folders. In addition to the internal SCAM types, Confolio also makes use of MIME types [16]. This is necessary for the browser to know with which application a file should be opened. There are several different scenarios where Confolio can get to a meaningful deployment. As mentioned, it is appropriate as an e-portfolio, but can also be used as plain document and resource management application.

Confolio makes heavy use of the Dojo Toolkit [13]. The metadata is presented using the AJAX version of the SHAME library [29]. Confolio can be run as a stand-alone application or certain elements can be embedded into other applications.

Conclusions and Future Work

In this paper we have introduced a generic architecture and framework for resource and metadata management that is adapted to the needs of web applications, especially mashups. We believe that the most important innovation is the introduction of an entry as a solution for how to manage a resource, its metadata and the corresponding administrative information. The broad definition of an entry allows a wide range of different situations, manifested through three type schemes. These schemes are expressed in every entry and specify whether the resource is digital or not, if the resource or the metadata is managed within the repository, and if the resource is one of a few built in types that SCAM has special knowledge of. Furthermore, the architecture introduces the context resource and requires that every entry must belong to a single context. This requirement makes it possible to have an ideal administration of entries in the system by giving specific users or groups access to a single context and correspondingly to all contained entries. We provide a simple HTTP-based interface to access entries, resources and metadata according to the principles of RESTful web services. We aim to support a wide range of data formats, although we started with RDF and JSON.

The choice of HTTP in combination with the possibility of having referencing entries enables SCAM installations to be loosely connected [12], e.g., in the Confolio use case this means that folders and other resources can be unintrusively mounted across systems. Future work includes stabilizing a search API which guarantees that the access control list is respected. The plan is to have both free-text search on metadata and qualified searches using some subset of SPARQL.

References

1. Fielding, R. T.: Architectural Styles and the Design of Network-based Software Architectures, Chapter 5: Representational State Transfer (REST). Dissertation, University of California, Irvine (2000)
2. Standardized Contextualized Access to Metadata (SCAM) framework, http://project.iml.umu.se/projects/scam
3. IMS Content Packaging Specification, http://www.imsglobal.org/content/packaging/
4. Official ADL SCORM Overview, http://www.adlnet.gov/scorm/
5. IMS ePortfolio Specification, http://www.imsglobal.org/ep/index.html
6. IMS Learning Design Specification, http://www.imsglobal.org/learningdesign/
7. Dusseault, L. M.: HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV). RFC 4918. IETF (2007)
8. Nottingham, M., Sayre, R.: The Atom Syndication Format. RFC 4287. IETF (2005)
9. Gregorio, J., de hOra, B.: The Atom Publishing Protocol. RFC 5023. IETF (2007)
10. W3C Technical Architecture Group, http://www.w3.org/2001/tag/
11. Jacobs, I., Walsh, N.: Architecture of the World Wide Web, Volume One. W3C Recommendation, http://www.w3.org/TR/webarch/ (2004)
12. Ebner, H., Palmér, M., Naeve, A.: Collaborative Construction of Artifacts, Proceedings of 4th Conference on Professional Knowledge Management, Potsdam, Germany (2007)
13. The Dojo Toolkit, http://dojotoolkit.org

14. Sauermann, L., Cyganiak, R.: Cool URIs for the Semantic Web. W3C Interest Group Note, http://www.w3.org/TR/cooluris/ (2008)
15. Palmér, M., Enoksson, F., Nilsson, M., Naeve, A.: Annotation Profile Specification. Deliverable 3.2, LUISA IST-FP6-027149, http://www.luisa-project.eu (2007)
16. Freed, N., Borenstein, N.: Multipurpose Internet Mail Extensions (MIME), Part Two: Media Types. RFC 2046. IETF (1996)
17. Adida, B., Birbeck, M.: RDFa Primer - Bridging the Human and Data Webs. W3C Working Draft, http://www.w3.org/TR/xhtml-rdfa-primer/ (2008)
18. Halpin, H., Davis, I.: GRDDL Primer. W3C Working Group Note, http://www.w3.org/TR/grddl-primer/ (2007)
19. Carroll, J. J., Stickler, P.: TriX: RDF Triples in XML. Tech Report. HP Labs (2004)
20. Sesame RDF Framework, http://openrdf.org
21. Restlet - Lightweight REST framework for Java, http://www.restlet.org
22. JSR 311: JAX-RS: The Java API for RESTful Web Services, http://jcp.org/en/jsr/detail?id=311
23. Bizer, C., Cyganiak, R.: The TriG Syntax. http://www4.wiwiss.fu-berlin.de/bizer/TriG/Spec/
24. JDIL - Data Integration in JSON, http://jdil.org
25. The Open Archives Initiative Protocol for Metadata Harvesting, http://www.openarchives.org/OAI/openarchivesprotocol.html
26. FIRE/LRE: The EUN Learning Resource Exchange, http://fire.eun.org
27. Simon, B., Massart, D., van Assche, F., Ternier, S., Duval, E.: Simple Query Interface Specification. http://www.prolearn-project.org/lori
28. Palmér, M., Naeve, A., Paulsson, F.: The SCAM Framework: Helping Semantic Web Applications to Store and Access Metadata. Proceedings of the European Semantic Web Symposium 2004. Springer (2004)
29. SHAME metadata library, http://shame.sourceforge.net