GeS 2007 German e-Science

Available online at http://www.ges2007.de

Versioning of Digital Objects in a Fedora-based Repository

Matthias Razum, Frank Schwichtenberg, Rozita Fridman

FIZ Karlsruhe, Hermann-von-Helmholtz-Platz 1,
76344 Eggenstein-Leopoldshafen, Germany
Email: {firstname.surname}@fiz-karlsruhe.de

Abstract

This presentation gives an overview on the complex versioning requirements for digital objects in the scope of the project eSciDoc and discusses our solution based on the flexible repository architecture Fedora. It describes the conceptual background of how the eSciDoc infrastructure handles versioning of digital objects. Versioning and object identification are strongly interwoven concepts. Therefore, the unique identification of objects and their versions are discussed as well.

1 Introduction

eSciDoc aims to realize a platform for communication and publication in scientific research organizations [1]. One of the major goals for eSciDoc is to support scientific collaboration in future eScience scenarios. Collaboration requires a shift from traditional digital library systems to a more interactive environment in which the concept of 'ownership' of an object is loosened and partly replaced by an ongoing authoring process that spans persons, places and time. Collaborative authoring raises additional requirements on versioning of digital objects. In distributed teams, authors may work at the same time on the same object, or may disagree with changes made by a co-author. If all intermediate or working versions of objects become part of the repository – and not just the final versions – these conflicts are detectable and can be resolved. Not all new versions created during the authoring process will necessarily be released to become publicly available.

2 Previous Work

Versioning is a common problem to many disciplines in computer science. In software development, versioning control systems like CVS or Subversion [2] track the differences between versions of (text) files. WebDAV (Web-based Distributed Authoring and Versioning) implements a set of extensions to the HTTP protocol that allows users to collaboratively edit and manage files on remote web servers [3]. It has a built-in support for version management, and in this respect resembles document management systems (DMS). However, digital libraries and institutional repositories impose additional requirements on the versioning capabilities. Stable references and

citations request for persistently identified versions. Long term archiving requires the use of a standards-based and open architecture that allows for easy migration of objects including their version history.

The NISO/ALPSP Working Group on Versions of Journal Articles [4] has focused its research on journal articles, but in general, the identified issues apply to other object types as well. However, their work concentrates on publicly available versions of objects. eSciDoc intends to cover the whole lifecycle of digital objects. This requires a broader scoping of versioning. The two JISC-funded projects RIVER (Scoping Study on Repository Version Identification) [5] and VERSIONS [6] provide a good overview on version identification, highlighting the problem domain, use cases, and give recommendations. Crane [7] raises the issue of the different citing traditions across scientific disciplines, pointing out the special requirements of the humanities. Here, citations often precisely identify pages or even individual lines and words. Even minor changes to a text, eventually triggered by just reformatting the text for presentation, might invalidate such citations. It is therefore crucial to trace all modifications of an object and create new versions appropriately. In such scenarios, citations and references need to include a pointer to a fixed version of a cited object. Out of this scenario, four major requirements pertaining versioning have to be fulfilled by the underlying repository architecture.

3 Versioning Requirements

3.1 Versioning on Object Level

eSciDoc intends to cover the whole lifecycle of digital objects. Storage of digital objects in eSciDoc is based on the Fedora Repository Architecture [8]. An object consists of relations to other objects, binary content (or 'content payload'), metadata, technical properties, and policies. Fedora splits up digital objects into these constituent parts. Each part is stored as a datastream. Accordingly, a Fedora object can be seen as an aggregation of datastreams. Fedora's versioning concept is based on the versioning of datastreams. Every modification of a datastream leads to a new version of the datastream, but not of the object itself. On the other hand, authors and editors perceive objects rather as one coherent entity, and not so much as a set of datastreams. They request a "whole-object" versioning, which complies with their mental model.

3.2 Internal and public versions

The eSciDoc system differentiates between versions and revisions. Versions represent intermediate work statuses and are only visible to authors of digital objects, whereas revisions are published versions of objects with persistent identifiers. Creating a revision is an intellectual step, which most often includes some form of quality assurance, whereas versioning is an automated process.

3.3 Fixed and Floating Object References

Scholarly work strongly relies on referencing existing material. Repositories may amplify this by providing not only secondary information (e.g. articles), but primary data and supplementary material as well. Authors may inter-relate all kind of objects that are accessible online, thus creating added value by revealing until then hidden or unknown connections. Other object types relying on references include annotations, translations, transcriptions, and so forth. In the digital domain, these associations may be expressed as object relations.

Versioning on objects then raises the question how to handle relations pointing to a versioned object. eSciDoc implements two approaches: fixed references pointing exactly to a given version of an object, and floating references always pointing to the latest version of an object. The author of the referencing work has to decide which form of references is appropriate. Citations of type “this collection contains relevant information” should point to the latest revision (floating reference), whereas a citation e.g., in the humanities pointing to a phrase or words would rather rely on a fixed reference pointing always to the same exact version of the cited object.

3.4 Container Objects

eSciDoc allows the grouping of objects by means of container objects like collections or bundles. Bundles may represent e.g. books or manuscripts. Contained objects might be chapters or pages. Containers are again objects within the repository. The grouping is done by defining object relations (expressed as RDF statements). Containers and their members form a graph-oriented composition of objects. Containers are citable objects with their own persistent identifier. Revisioning of contained objects forces a new revision of the container object too.

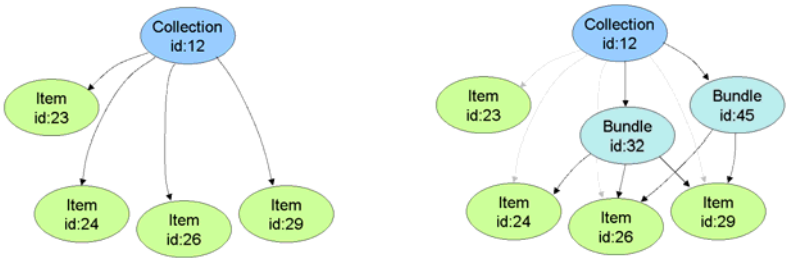

Figure 1: A collection with items (on the left) and the same collection with two bundles added (on the right). The container and item objects are connected by relations, thus forming a graph-oriented composition of objects.

4 Technical Approach

4.1 Fedora's Versioning Mechanism

Fedora's internal object model is composed of four major parts: a system identifier, a set of key descriptive properties, datastreams, and object integrity components. Datastreams either encapsulate the actual bytestream content internally or reference it externally. Object integrity components are special datastreams managed by Fedora pertaining relations, policies, and audit trails.

Versioning is automatically done by Fedora at datastream level. Every time a user modifies an object, Fedora creates new versions of all affected datastreams within the digital object, while maintaining all prior datastream versions. Each new datastream version is identified by a timestamp. In order to retrieve a version of a datastream, the Fedora API provides methods to access datastreams based on a timestamp.

Fedora versioning lacks a mechanism to tag versions with a name or number at the object level. A versioned object is just an aggregated view of all datastream versions valid at a given point in time. The user needs to retrieve every single datastream of an object with the appropriate timestamp to get the representation of a specific object version. With Fedora's built-in versioning mechanism, it is difficult to see how many versions for an object have been created over time. A versioning of the entire object, as envisioned for eSciDoc, has to associate the timestamp of modifications with a version number or name. The eSciDoc system can retrieve the datastreams that belong to a concrete object version using such a version number and its associated timestamp.

4.2 Object identification

One of the keys to the development of distributed digital libraries or archives is the ability to rely on references between digital resources to remain valid over time. Such references may be citations, bookmarks, or links embedded in bibliographies, indexes, and catalogues. Much of the value of digital resources for scholarly communication lies in enabling resources to be referenced reliably with resolvable and actionable links over long periods. Therefore, libraries, archives, academic institutions, and publishers have an interest in the persistence of resource identification [9]. System Identifiers are assigned by a system in order to manage resources within the scope of the system. They are not guaranteed to be persistent over time. Persistent Identifiers have a stable name and an adaptable reference to the currently valid system identifier. This association is updated whenever the system identifier changes. Persistent identification requires a resolution system, which handles the association between the persistent identifier and the system identifier and ensures the uniqueness of the persistent identifier.

Persistence is provided through governance rather than through purely technical constraints. The reliability of a persistent identifier therefore depends on the reliability of the organization, which is responsible for the resolution system, and the reliability of the responsible Naming Authority [10]. For the sake of simplicity, we will exclude persistent identification from our further examinations.

System Identifiers in the eSciDoc system are generated as serial numbers. They identify an object in its latest version. To address a previous version of an object, the version number is appended to the system identifier, separated by a colon (see Figure 2).

```
<system id> ::= <namespace> ‘:’ <serial number>[ ‘:’ <version number>]
```

Figure 2: eSciDoc System Identifier Syntax

A federation of distributed eSciDoc systems is envisioned in the medium term. In such scenarios, the problem of moving objects from one repository to another needs to be addressed [11]. As system identifiers are only unique within one system, transferal of objects may result in duplicate identifiers. These ambiguous identifiers then lead to corrupted systems. Therefore, eSciDoc system identifiers are prefixed by a namespace that distinguishes the sequence of serial numbers local to one repository from a sequence of serial numbers local to another repository. Organizations running more than one eSciDoc repository are able to guarantee the uniqueness of identifiers across all their repositories by configuring them with differing namespaces.

At the same time, eSciDoc repositories are able to handle multiple namespaces, so that namespaces (including all resources created under this namespace) may be transferred to another repository. This allows transferring objects from one repository to another without changing the system identifier. Fedora is already able to handle system identifiers in different namespaces.

4.3 Version Information in the Resources Properties

An eSciDoc object is treated as a *resource* that consists of several sub-resources. Sub-resources of an eSciDoc object are representations of the datastreams of the corresponding Fedora object. Each resource comes with a set of properties used to describe the resource itself. If a client requests a representation of a versionable resource, the respective resource handler will provide additional properties pertaining three object versions: the latest version, the latest revision and the current version. Based on the user role and the context, the user may only be allowed to see revisions. In this case, the latest version will be identical with the latest revision and the current version will be a revision too.

The latest revision is the newest version of the object tagged as revision. This part of the properties section will be missing if the object has never

been released yet. The current version is the version of the retrieved resource representation.

The current version section contains the versions number, timestamp, status, and validity, while the latest version section contains only the number and timestamp and the latest revision section the number, timestamp and the persistent identifier of that revision.

```
<item:current-version xlink:type="simple" xlink:title="3rd version">
  <item:number>3</item:number>
  <item:date>2007-03-15T15:03:39.431Z</item:date>
  <item:version-status>released</item:version-status>
  <item:valid-status>valid</item:valid-status>
</item:current-version>
<item:latest-version xlink:type="simple" xlink:title="4th version">
  <item:number>4</item:number>
  <item:date>2007-03-16T22:10:03.834Z</item:date>
</item:latest-version>
<item:latest-revision xlink:type="simple" xlink:title="3rd version">
  <item:number>3</item:number>
  <item:date>2007-03-15T15:03:39.431Z</item:date>
  <item:pid>hdl:12345/ns:100:2</item:pid>
</item:latest-revision>
```

Figure 3**:** Extract of the property part of an object's XML representation

To retrieve a version of an object, users append the version number to the object identifier, separated by a colon. Depending on the user role and context, not all versions of an object may be accessible by users. Using the object identifier without any suffix will always retrieve the latest accessible version of an object. Based on the user role and the context, this can be either the latest version or the latest revision.

4.4 Introducing a Whole-Object Versioning Datastream

eSciDoc creates an additional datastream for "whole object versioning" in each Fedora object. With every modification of the eSciDoc object, and therefore with every modification of at least one datastream of the corresponding Fedora object, this whole-object versioning datastream gets a new version entry. Basically, a version entry consists of the version number and a timestamp. The timestamp is used to retrieve the appropriate datastream versions related to that "object version" from Fedora (see Figure 4).

The whole-object versioning datastream is a XML tree with *version* nodes in a *version-history* root node. A version entry is a *version* node containing the version number, the versions timestamp, status and validity and an optional comment. A version with a version status "released" is a revision. The eSciDoc resource handler updates this XML tree whenever a 'write' operation is completed.

When retrieving a specific version of a resource, the system extracts the timestamp of the requested version from the versioning datastream and retrieves the appropriate datastreams the representation of the eSciDoc object consists of. If a datastream is newer than the provided timestamp, it will not be included in the resulting object. This ensures that the system can always

reconstruct the exact composition of an object, as it was when the version was created. Users may request the version history of a resource by retrieving its “versions” sub-resource. Based on the user role and the context, the system will retrieve either a complete list of all versions or just the publicly accessible revisions from the versioning datastream.

```
<versions:version-history xml:base="http://www.escidoc.de/">
  <versions:version xlink:title=“ver 2“ xlink:href="/ir/item/id:21:2">
    <versions:version-no>2</versions:version-no>
    <versions:timestamp>2006-07-07T12:13:34Z</versions:timestamp>
    <versions:version-status>released</versions:version-status>
    <versions:valid-status>valid</versions:valid-status>
    <versions:comment>2nd version, 1st revision</versions:comment>
  </versions:version>
  <versions:version xlink:title=“ver 1“ xlink:href="/ir/item/id:21:1">
    <versions:version-no>1</versions:version-no>
    <versions:timestamp>2006-05-10T00:21:57Z</versions:timestamp>
    <versions:version-status>submitted</versions:version-status>
    <versions:valid-status>valid</versions:valid-status>
    <versions:comment>1st whole object version</versions:comment>
  </versions:version>
</versions:version-history>
```

Figure 4: XML tree representation of the whole object versioning metadata

The versioning datastream itself is a non-versionable datastream. The system sequentially prepends new object versions to the version-history node. Prepending newer versions allows for a more efficient parsing and faster access to the most recent versions, especially when using a streaming parser to access the XML tree.

4.5 Versioning of Graph-oriented Compositions of Objects

One important aspect of version is the handling of complex compositions of objects. Container objects have strong relationships with their member objects. Relations are expressed as RDF statements and form the edges of a graph. They are stored in a special datastream of the source object, the so-called RELS-EXT datastream.

The basic goal of the versioning datastream is to keep track of modifications upon all digital objects that, together, form a graph-oriented composition. If the relationships of a container object are changed by adding or removing objects, a new version of the RELS-EXT datastream is created by Fedora. eSciDoc adds a new version entry to the whole-object versioning datastream. If a specific version of the container object is retrieved, the relations are extracted from the appropriate RELS-EXT datastream version, based on the timestamp found in the whole-object versioning datastream. This approach ensures that only those member objects are considered as parts of the graph-oriented composition that belong to the requested version. Members that were added in a later version are not part of the retrieved version of the RELS-EXT datastream.

5 Conclusions

Versioning is an essential feature for repositories that cover the whole object lifecycle. Fedora already comes with a powerful versioning mechanism, but cannot fulfill all requirements of eSciDoc. The presented whole-object versioning covers those aspects and can even handle versioning on object graphs (e.g., a container with its members). The proposed approach provides a solution for advanced versioning requirement and at the same time is a demonstration of Fedora's flexibility and adaptability. At this time, eSciDoc versioning is in a prototype state. The eSciDoc team works in close cooperation with the Fedora Core Development team to generalized the approach and eventually merge it into the current Fedora code base.

Acknowledgements

The concepts presented in this paper are partly based on eSciDoc's Logical Data Model by Natasa Bulatovic (MPDL, Max Planck Society) and the internal technical paper 'Versioning of Publication Items' by Inga Overkamp (MPDL, Max Planck Society). A joint workshop of the eSciDoc Team with Sandy Payette and Carl Lagoze greatly influenced the presented approach. The work has been funded by the German Ministry of Education and research (BMBF) as part of the eSciDoc project.

References

1. See project website <http://www.escidoc-project.de/homepage.html>
2. Concurrent Versions System website <http://www.nongnu.org/cvs/>
Subversion website <http://subversion.tigris.org/>
3. Whitehead, J. 1998. Collaborative Authoring on the Web: Introducing WebDAV. Bulletin of the American Society for Information Science, Vol. 25, Issue 1, 25-29. <http://www.asis.org/Bulletin/Oct-98/webdav.html>
WebDAV website <http://www.webdav.org/>
4. <http://www.niso.org/committees/Journal_versioning/JournalVer_comm.html>
5. Rumsey, S., Shipsey, F., Fraser, M., Noble, H., Bide, M., Look, H., Kahn, D. 2006. RIVER (Scoping Study on Repository Version Identification). Working Group on Scholarly Communications, JISC (Draft Final Report)
<http://www.jisc.ac.uk/uploaded_documents/RIVER%20Final%20Report.pdf>
6. See VERSIONS project website <http://www.lse.ac.uk/library/versions/>
7. Crane, G. 2002. Cultural Heritage Digital Libraries: Needs and Components. In Agosti, M., Thanos, C. (Eds.). 2002. ECDL 2002, Lecture Notes in Computer Science, Vol. 2458, 626-637. Springer Berlin / Heidelberg
8. Lagoze, C., Payette, S., Shin, E., Wilper, C. 2006. Fedora: an architecture for complex objects and their relationships. International Journal on Digital Libraries. Volume 6, Issue 2. Springer Berlin / Heidelberg
<http://dx.doi.org/10.1007/s00799-005-0130-3>
9. Powell, A., Lyon, L. 2001. The DNER Technical Architecture: scoping the information environment.
<http://www.ukoln.ac.uk/distributed-systems/jisc-ie/arch/>
10. Kahn, R., Wilensky, R. 1995. A Framework for Distributed Digital Object Services. Corporation for National Research Initiatives, cnri.dlib/tn95-01
<http://www.cnri.reston.va.us/k-w.html>

11. Tansley, R. 2006. Building a Distributed, Standards-based Repository Federation, The China Digital Museum Project. D-Lib Magazine, Volume 12, Issue 7/8. <http://www.dlib.org/dlib/july06/tansley/07tansley.html>