eSciDoc Infrastructure: A Fedora-Based e-Research Framework

Matthias Razum, Frank Schwichtenberg, Steffen Wagner, and Michael Hoppe

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

Abstract. eSciDoc is the open-source e-Research framework jointly developed by the German Max Planck Society and FIZ Karlsruhe. It consists of a generic set of basic services ("eSciDoc Infrastructure") and various applications built on top of this infrastructure ("eSciDoc Solutions"). This paper focuses on the eSciDoc Infrastructure, highlights the differences to the underlying Fedora repository, and demonstrates its powerful und application-centric programming model. Further on, we discuss challenges for e-Research Infrastructures and how we addressed them with the eSciDoc Infrastructure.

1 Introduction

Digital Repositories undergo yet again a substantial change of paradigm. While they started several years ago with a library perspective, mainly focusing on publications, they are now becoming more and more a commodity tool for the workaday life of researchers. Quite often the repository itself is just a background service, providing storage, persistent identification, preservation, and discovery of the content. It is hidden from the end-user by means of specialized applications or services. Fedora's approach of providing a repository architecture rather than an end-user tool matches this evolution. eSciDoc (Dreyer, Bulatovic, Tschida, & Razum, 2007), from the start of the project nearly five years ago, has always been in line with this development by separating backend services (eSciDoc Infrastructure) and front-end applications (eSciDoc Solutions)[1].

E-Science and e-Research trigger several new challenges (Hey & Trefethen, 2005). Whereas many e-Science applications concentrate on massive amounts of data and how to store, manipulate, and analyze them, eSciDoc focuses more on a more holistic approach to knowledge management in the research process. If not only the final results of the research process, but all intermediate steps from the first idea over experimentation, analysis, and aggregation should be represented within a repository, the digital library becomes a 'e-Research Infrastructure'. Figure 1 depicts the process and eSciDoc's approach with a generic infrastructure and specialized solutions supporting the various steps of the research process.

[1] http://www.escidoc.org/

M. Agosti et al. (Eds.): ECDL 2009, LNCS 5714, pp. 227–238, 2009.

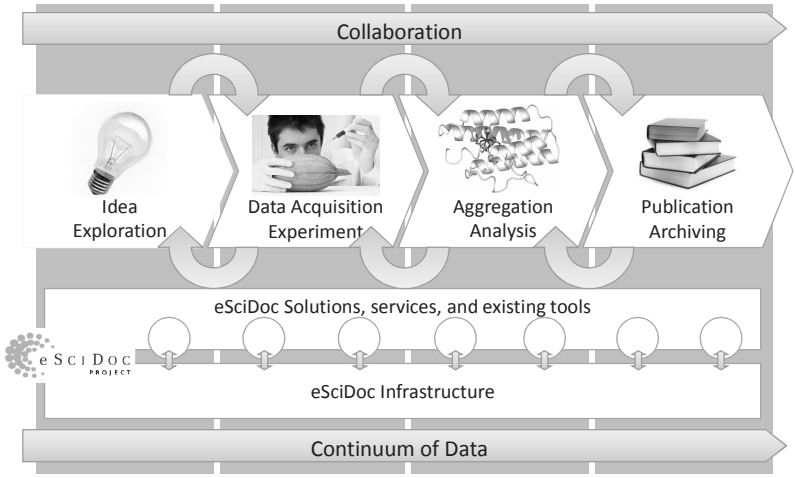

Fig. 1. eSciDoc supports the whole research process with a generic infrastructure and specialized applications, services, and integration of existing tools

2 Challenges for e-Research Infrastructures

A whole new set of requirements have to be taken into consideration for e-Research infrastructures. In the following, we will name a few of them and show how eSciDoc currently supports researchers, data curators, librarians, and developers with powerful, yet simple-to-use features. Intentionally left out are challenges like the data deluge (Hey & Trefethen, 2003) and massive parallel computing/grid computing, which we don't address with eSciDoc.

Maintain both data and publications. Primary data differs a lot from traditional publications. It comes in various and sometimes exotic file formats, metadata profiles are specific to disciplines or even projects, and it quite often includes datasets consisting of several files. eSciDoc's flexible content models and compound objects allow to store these complex data objects. The support for arbitrary metadata profiles for each object, both on the logical and the physical (file) level allows for proper description and discovery.

Reliable citations. Citations have always been the backbone of scholarly communication and a challenge for the web, digital libraries, and electronic archives. Much of the value of digital resources for scholarly communication lies in enabling resources to be cited 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 (Powell & Lyon, 2001). New publication types like datasets, simulations, etc. in combination with increasing numbers of born-digital materials requires the adoption of a digital equivalent of the traditional paper-based citations. eSciDoc supports a wide range of persistent identifiers on the object and file level. Versioning ensures that once cited, objects will always appear in the same way as originally perceived by the citing author. To avoid 'broken' references, eSciDoc will

never delete a once published object. Instead, objects may be withdrawn, which means that access to the files is inhibited, but the metadata and a note explaining the reason for the withdrawal are still accessible.

Focus on researchers. E-Research infrastructures are built for researchers, so their needs and working attitudes should be the main focus. Research is a dynamic process, and tools should support this process, not block it. eSciDoc allows researchers to build their own solutions easily because of much basic functionality (storage, search, authentication, access rights) being provided by the infrastructure. This separation of concerns allows them to focus on their scholarly 'business logic'. Open programming interfaces support a wide range of programming and scripting languages when building your own solution. More important, researchers only reluctantly change their working habits. eSciDoc's APIs comply with the web architecture, thus facilitating mash-ups, integration with existing tools and scientific software packages, and data exchange with other scientific repositories.

Veracity and fidelity of research and re-use of data. Supporting the whole research process with tools and maintaining all (digital) artifacts that are created or modified in the course of this process may help to enrich traditional publications with supplementary materials. Such data may help others in reproducing and validating results. Provenance metadata, tagging, semantic links between objects, and multiple metadata records of arbitrary profiles provide context not only for long-term preservation, but as well for discovery and re-use of objects.

Collaboration. Collaboration has always been an important aspect of scholarly work. E-Research infrastructures can support and even amplify the collaborative aspect by technical means. Cross-institutional teams can benefit from distributed authentication (Shibboleth), which supports virtual working groups. Flexible access policies allow for fine-grained rights management, thus giving researchers the option to exactly control the dissemination of their artifacts created in the course of the research process at any given time. Team work, especially when geographically distributed, requires versioning and audit trails, so that team members can track changes and eventually roll back unwanted or unintentional modifications of any artifact.

Mixture of open access and private material. Research data is not always freely accessible. Researchers often dislike the idea of releasing intermediate results before the final publication. Some datasets might not be publishable at all (e.g., due to privacy issues). Even within a project team, fine-grained access rights might be necessary. eSciDoc implements a single point of policy enforcement as part of the infrastructure, which leads to a lightweight solution-side implementation. Even badly designed solutions cannot compromise the overall security.

Preservation. The manifold file formats, the complex compound objects in e-Research, and the lack of standards in describing datasets appropriately impede the long-term preservation of the data stored or referenced in e-Research infrastructures. eSciDoc doesn't provide a comprehensive solution for this problem, especially as preservation has many organizational (i.e., non-technical) aspects to be looked at. However, eSciDoc supports several metadata records of arbitrary profiles per record, which not only caters for descriptive, technical, and administrative metadata records,

but allows describing the same object out of different perspectives. This is especially important for discovery and re-use of objects across disciplines. An Audit Trail and a PREMIS-based event history[2] keep track of changes to the object. Fedora's standardized METS[3]-based XML object container ensures a software-independent storage format, can be validated, and supports checksums for the digital content. The file-based storage model eases backup and restore operations, accompanied with Fedora's unique rebuild capabilities.

3 Differences between Fedora and eSciDoc

Fedora (Lagoze, Payette, Shin, & Wilper, 2005), the Flexible Extensible Digital Object Repository, is a renowned solution for all kinds of applications, including e-Research use cases. Fedora is the repository underneath the eSciDoc Infrastructure, and it already provides many features to address the above mentioned challenges. So why did we chose to encapsulate Fedora in an extensive middleware approach (which the eSciDoc Infrastructure actually is) and what are the main differences on a functional level between an out-of-the-box Fedora installation and eSciDoc?

Fedora provides a very generic set of functionalities, addressing the needs of various communities and use cases. On the other hand, this means that it only provides low-level functionality, requiring developers to spend time implementing high-level services. eSciDoc tries to fill that gap by adding these high-level services on top of Fedora while hiding some of the more complex aspects of Fedora, thus increasing the productivity of developers. However, this advantage contrasts with a reduction of flexibility.

1. Datastreams and Object Patterns
A Fedora object consists of several datastreams, which contain either XML or binary content. The repository developer has to define the layout and allowed contents for each datastream – the 'content model'. Fedora 3.0 implements the Content Model Architecture (CMA)[4], which codifies the previous implicit content model approaches. CMA assures that objects always comply with the model, but it does not help to find the best possible model. Content modeling is one of the challenges to start with Fedora, even for simpler use cases. Therefore, eSciDoc introduced 'object patterns' for basic object types: Items and Containers (see figure 2). For both object patterns, the layout, naming, and allowed content is predefined. eSciDoc objects are therefore less flexible, but simplify data modeling. Still, they provide flexibility where needed (e.g. storing metadata records). Our experience shows that this model accommodates for most use cases.

eSciDoc provides several more basic objects types, which are more fixed in nature and therefore are not seen as 'object patterns'. The most relevant ones are the Organizational Unit and the Context. Organizational Units are used to represent hierarchical structures of organizations with its institutes, departments, working groups, and so on. These hierarchies can be used to identify owners of objects, but at the same time can

[2] http://www.loc.gov/standards/premis/
[3] http://www.loc.gov/standards/mets/
[4] http://www.fedora-commons.org/documentation/3.0b1/userdocs/digitalobjects/cmda.html

be referenced from within metadata records to state the affiliation of authors. Contexts are administrative entities, which on the one hand side state the legal and organizational responsibility, on the other hand side they are used to store configuration information.

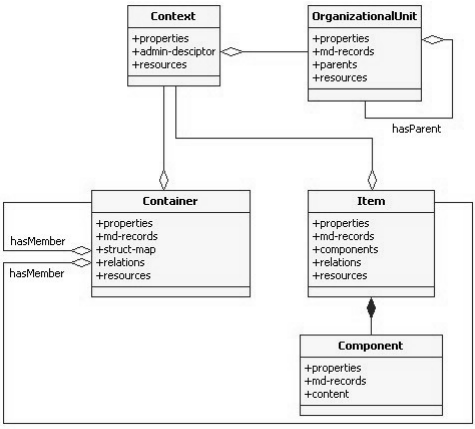

Fig. 2. eSciDoc Data Model, showing the two object patterns 'Item' and 'Container'. Items may have one or more components, which hold the different manifestations of the content.

The **Item pattern** represents a single entity with possibly multiple manifestations. Each manifestation is included in the Item as separate part, the **Component**. A Component includes the manifestation-related metadata, some properties (e.g., mime-type, file name), and the content itself. The **Container pattern** aggregates Items or other Containers. Object patterns support multiple metadata records, object relations (within and outside of eSciDoc), and some logistic and lifecycle properties. All the information is stored in one or more Fedora objects, following the 'atomistic' content model paradigm. However, the user will only work with a single eSciDoc object. All the complex work is done behind the scenes by the eSciDoc Infrastructure.

2. Object Lifecycle

Items and Containers both implement a basic lifecycle in the form of a simple workflow. Each eSciDoc object derived from one of the object patterns is created in status 'pending'. Submitting the object forwards it to the quality assurance stage, from where it can be either sent back to the creator for revision or released (i.e., made publicly accessible). In rare cases, released Items need to be withdrawn (e.g., because of copyright infringements), which is the last status in the lifecycle. For each status, different access rights (based on policies, roles, and scopes) may be defined. Moving objects from one status to the next is as easy as invoking a single method. Figure 3 depicts the available states of the object lifecycle and their succession.

Objects that are in status 'released' are not fixed. Authors may decide to further work on them. In this case, a new version of the object will be created, and a version status will be set to 'pending', whereas the object status will remain 'released'. This

means that the author has a working version accessible only to her and maybe invited collaborators, whereas the rest of the world still sees the released version of the object. As soon as the new working version has been submitted and released, it will be presented to non-privileged users as the latest version, complementing the previously released version.

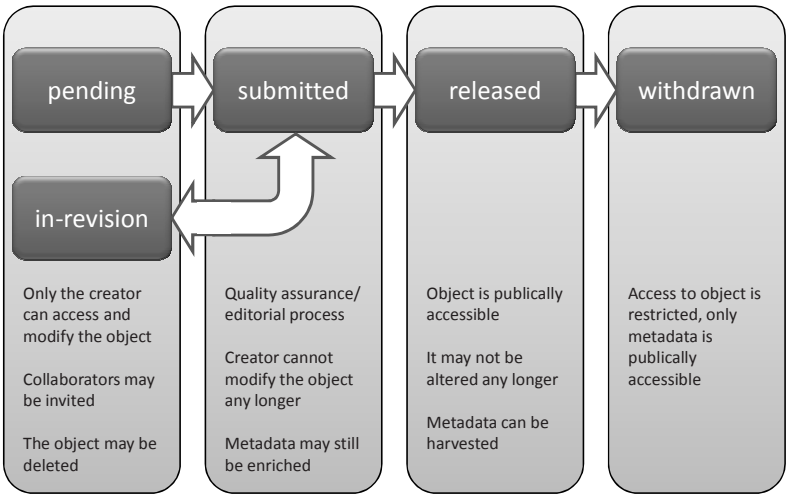

Fig. 3. eSciDoc's object lifecycle with different states and default access policies

3. Versioning

eSciDoc extends Fedora's versioning approach. Because of the atomistic content model, a single eSciDoc object may in fact be a graph-oriented composition of Fedora objects. However, the user conceives eSciDoc objects as one entity and expects what we call 'whole-object versioning' (WOV). Therefore, the eSciDoc Infrastructure maintains a special datastream to facilitate that view by keeping track of modifications upon all digital objects that, together, form a graph-oriented composition, such as multi-object content models with whole/part (Item resource with Component parts) or parent/child (Container resource with member objects) relationships.

eSciDoc differentiates between 'working versions' and 'releases', which is tightly coupled with the object lifecycle. Working versions are only accessible to authors and collaborators, whereas releases are publicly visible. For releases, eSciDoc implements a graph-aware versioning. If a part changes, there must be a new version of the whole, i.e., if a Component is updated, the Item itself is updated as well. If a child is either added or removed, there must be a new version of the parent. If an existing child is just modified, the parent object is not versioned. This is a simplification to avoid too many versions of parents with many children. However, if a Container is released, the Infrastructure ensures that all relationships to its members are updated first, which means that the simplification only affects working versions and not releases (which are the only ones to be cited or referenced).

Keeping track of changes within a compositional network can be done without actually creating separate copies of all its objects, which could raise scalability issues when there are many changes. Instead, eSciDoc versioning re-uses the existing Fedora content versioning scheme, augmented with the additional WOV metadata stream carried in "parent" and "whole" digital objects. This "whole-object versioning" metadata is a simple XML tree with a node for every version of the resource or the parent/child graph. The Object Manager updates the WOV metadata whenever a write operation is completed upon a part or child object. So, the eSciDoc Infrastructure will generate a single intellectual version of the whole graph-oriented composition of Fedora objects even for actions that involve modifications of several datastreams or several Fedora objects. However, not all method invocations create new versions, e.g. forwarding an object to the next state in its lifecycle. For operations that create no new versions eSciDoc maintains an additional event log that tracks all actions using PREMIS.

4. Application-oriented Representation

The fact that an eSciDoc object actually is a graph of Fedora objects with multiple datastreams is completely transparent to the user. The eSciDoc Infrastructure exposes its contents as XML representations, which contain all relevant information for typical application scenarios. That includes version information, metadata, and references to other objects or parts within the same object. References are expressed as XLink simple links (DeRose, Maler, & Orchard, 2001). Making object relations explicit by means of XLink simple links allow for easy navigation through the object graph. Rarely requested parts of an eSciDoc object, like the event log, are not included in the standard representation, but can easily be retrieved by means of 'virtual resources'. Virtual resources again are represented as XLink simple links with an 'href' attribute, so access to this additional information is just a 'mouse click away'.

'Parts' of an object may be the object properties, one of the metadata records, or a component. Each part is retrievable via its XLink 'href' attribute and thus may be seen as sub-resource of the entire resource. If, for example, a user is interested in just one metadata record, there is no need to retrieve the representation of the entire object.

5. Authentication and Authorization

eSciDoc relies on distributed authentication systems like directory servers with an LDAP interface and Shibboleth (Scavo & Cantor, 2005). In distributed authentication systems like Shibboleth, users are maintained in the identity management system (or 'identity provider', IdP) of their home institution, where their credentials like usernames and passwords are kept. Consequently, eSciDoc includes no user management. However, in order to be able to associate users with roles, the infrastructure creates and maintains proxy objects for users that have accessed at least once an eSciDoc Solution. Each user proxy object consists of a unique name and a set of attributes. The attributes are initially populated during the first login and are updated with each subsequent login, based on the unique name. The attribute set depends on either the Shibboleth federation or the configuration of the directory server. The attributes are mainly used for personalization features.

eSciDoc replaces Fedora's built-in authorization mechanism, mainly because eSciDoc secures access to resources stored both within and outside of Fedora, and the

need to handle object graphs as single entities. All authentication and authorization functionality is encapsulated in the eSciDoc Infrastructure, therefore the application programmer has not to consider the complexity of its implementation – this is especially important when several applications run on the same infrastructure, eventually sharing data. Every request to the infrastructure has to pass through the included authentication and authorization layer to access a method of the business logic. The eSciDoc authorization relies on five concepts: users (represented by the above mentioned user proxy objects), policies, roles, scopes, and groups:

Policies define access rights of users for resources, based on a set of rules. eSciDoc uses XACML to express these policies and rules. During the evaluation of policies, both standard XACML and eSciDoc-specific attributes are matched against conditions expressed in these rules. Additionally, policies may define target actions. In this case, the policy only is evaluated if the requested action matches one of the actions defined in the target of the policy. Typically, actions are directly mapped to method invocations of an eSciDoc API. If the action matches, rules are applied. Rules again may define target actions. In such a case, the rule will only get evaluated if the requested action matches one of the rule's target actions.

Roles define a set of rights, expressed in one or more XACML policies (Moses, 2004). Typically, administrators grant roles instead of explicit policies to users. Roles quite often describe a real-world responsibility like 'author', 'metadata editor', or 'collaborator'. Any non-authenticated, anonymous user gets granted with the 'default user' role, i.e. the role encompasses all actions that are allowed for everyone (e.g., retrieving Items in status 'released').

Scopes further constrain roles. They correspond to eSciDoc resources (e.g., Containers, Contexts, or Organizational Units) and are expressed by eSciDoc-specific resource attributes. An example might help to understand the concept: Ann works for the 'Foo' project. The system administrator created a new Context for her project and assigned her the role 'author'. But Ann shall only be allowed to create and modify objects within the Context of her own project. So if a second Context for Bob's project 'Bar' exists, the system administrator will grant Ann the generic role 'author' with the scope 'Context Foo'. Bob will have the same role, but with a different scope 'Context Bar'.

Groups are an additional concept to simplify the management of access rights. Groups may contain users or other groups. The member definition may be static (i.e. explicitly defined by the system administrator) or dynamic (i.e. based on user attributes). Groups can be associated with roles. If a user belongs to a group, he or she automatically inherits the roles of the group. If a user belongs to several groups at the same time, all associated group roles are inherited, complementing the roles that are directly associated with the user. Groups allow for the automatic assignment of roles (and thus policies) to users that have never logged in before. A common scenario is to give deposit access to the Context of an institute based on the attribute that contains the name of the institute (Organizational Unit).

With this powerful and fine-grained authentication and authorization mechanism, eSciDoc is well equipped to fulfill all kinds of access right requirements in common e-Research scenarios.

6. Persistent Identification

A persistent identifier identifies a resource independently from the storage system or location. There are many competing persistent identification systems, like PURL[5], Handle System[6], and DOI[7]. The general approach to make an identifier persistent is by separating the identifier from the locator and provide a mapping mechanism between both. If a resource is moved to a new location, only the mapping needs to be updated with new location, whereas all references stay stable. This, persistent identifiers provide ongoing access and reference to a resource and can be used whenever a permanent link is required. In science and humanities, stable references (e.g., for citations) are of great importance. With the advent of e-Research, not only publications, but all relevant objects should become citable (Klump, Bertelmann, Brase, Diepenbroek, Grobe, & Höck, 2006).

A persistent identifier typically consists of two parts: a prefix or namespace, which uniquely identifies the organization that is responsible for the persistently identified content, and a suffix, which uniquely identifies a resource within the scope of the organization. The process to create the suffix of PIDs is generally called minting. Different opinions exist on how much semantics a suffix should include (Kunze, 2003). Keeping identifiers semantic-free is a widely accepted approach (Sollins & Masinter, 1994). A minter should be configurable to address the varying requirements of different organizations.

Fedora comes with a default, database-based Identifier module, which only generates local identifiers. It has no built-in support for minting and assignment of PIDs. OhioLINK has extended Fedora with their HandlePIDGeneratorModule[8] for the Handle System. It can automatically assign a handle instead of a local identifier for each Fedora object, but constraints the length of the handle – similar to the default identifier – to a length of 64 characters.

eSciDoc persistent identifiers are not a replacement for the local identifier, but additional attributes of an object. A resource is retrievable either via its PID or its local identifier. Length and structure of a PID is not limited. Different PID generators (minters) are configurable, including simple serial numbers, checksums, and advanced rules based on the NOID minter[9].

eSciDoc differentiates between three kinds of PIDs: Object PID, Version PID, and Content PID. Each resource has an Object PID, which identifies the whole resource, including all of its versions. Using the Object PID will always retrieve the latest version of a resource. Each version of a resource can be identified with a Version PID, which will always retrieve the exact version of a resource. Additionally, all binary content of an object (e.g., PDF, image, etc.) is persistently identifiable by its Content PID.

The default configuration of eSciDoc's object lifecycle ensures that each resource is assigned with an Object and Version PID before a version is 'released' (i.e., made publicly available). The assignment of PIDs is not triggered automatically by a status change in the object lifecycle. Instead, it is the responsibility of each eSciDoc solution

[5] http://purl.org/
[6] http://www.handle.net/
[7] http://www.doi.org/
[8] http://drc-dev.ohiolink.edu/wiki/HandleGenerator
[9] http://www.cdlib.org/inside/diglib/ark/noid.pdf

to trigger the minting and assignment of PIDs. So it is up to the application designer when and how to assign PIDs, with respect to differing content models and file types. eSciDoc Solutions may even provide suffix values for the PID handler.

Minting and registering of PIDs in eSciDoc is managed by the PID Manager, one of the services of the eSciDoc Infrastructure. In its default configuration, the PID Manager currently supports only the Handle System. The PID Manager Service can be extended to support other systems as well (e.g. PURL). A concurrent use of several PID systems is possible, as well as including externally managed PIDs (i.e. PIDs that are assigned to an object before it is ingested into eSciDoc).

4 Application-Oriented Programming Model

The eSciDoc e-Research environment is built as a service-oriented architecture (SOA). The infrastructure consists of several independent services. Each service implements both a REST (Fielding, 2000) and a SOAP API. The APIs support simple CRUD (Create, Retrieve, Update, and Delete) and task-oriented methods. These APIs together with object patterns, their XML representations, versioning, and the powerful authentication and authorization form eSciDoc's application-oriented programming model, which focuses on the mindset of application developers and hides the technical details of the implementation as much as possible.

As already mentioned earlier, we see the ability to integrate existing tools and software packages with the eSciDoc Infrastructure as an important design goal. Another one is to account for various programming languages for solution development, including scripting languages for fast prototyping, mash-ups, and thin-client development. As a proof of concept, we have integrated the *Schema Driven Metadata Editor for eResearch* developed by ARCHER[10] and MAENAD[11] in order to allow comfortable editing of metadata records of eSciDoc objects. Another example is the integration of DigiLib, the versatile image viewing environment for the internet[12]. The most simplistic solution can be built based on XSLT transformations of object representations delivered by the eSciDoc Infrastructure. All necessary transformations are delegated to the browser of the user.

The API is representation-oriented, i.e. changing an object typically means retrieving the representation, changing it, and sending it back to the eSciDoc Infrastructure, thus following a *load-edit-save* paradigm. It is possible to just modify parts or single values in the XML representation of a resource and send it back for update. Some properties or parts of the representation are purely informational and therefore not updateable. These are ignored by the eSciDoc Infrastructure when a representation is sent back in order to store it. So it is actually possible to use a retrieved object representation as template for creating a new one. A developer has just to care for the essential properties.

In contrast to the representation-oriented approach, some actions like forwarding a resource in workflow are executed via task-oriented methods, which encapsulate these operations and strictly separate them from others.

[10] http://www.archer.edu.au/
[11] http://metadata.net/sfprojects/mde.html
[12] http://digilib.berlios.de/

The XML representations of eSciDoc objects and their exposure as web resources, together with the built-in methods of the HTT Protocol (GET, PUT, DELETE, etc.), form the CRUD-based web programming interface of the eSciDoc infrastructure. That can easily be used from existing applications or even websites. It fits well with AJAX web development techniques and supports easy integration of repository features into applications. Because of the built-in simple workflow, versioning, and authentication and authorization, already the simplest possible client solution supports basic repository features. The communication between client and infrastructure relies on well-known standards for data representation and distributed communication. Thus, the eSciDoc Infrastructure is a developer-friendly application framework.

5 Conclusion and Outlook

The eSciDoc Infrastructure encapsulates Fedora as its core component, but adds a wide range of higher-level services and its application-oriented programming model. It allows building various types of solutions, from light-weight Javascript hacks to fully-blown Java applications. It fulfills the vision of creating an efficient, flexible, programmer-friendly e-Research framework supporting web-based publication, collaboration and communication for research environments assembled with the repository capabilities of Fedora Digital Repository.

Acknowledgements

eSciDoc has been funded by the German Ministry for Education and Research (BMBF) from 2004-2009. Both Max Planck Society and FIZ Karlsruhe have substantially added additional resources to the project to meet the very ambitious project goals. We are very grateful for this financial support.

The design and implementation of the eSciDoc Infrastructure as well as the underlying concepts of eSciDoc have been conceived by a much larger team than represented by the authors of this paper, both at FIZ Karlsruhe and at the Max Planck Digital Library. We received a lot of very valuable input from researchers and librarians from various institutes of the Max Planck Society and other organizations. Finally, we would like to thank Fedora Commons for their great software and support.

References

DeRose, S., Maler, E., Orchard, D.: XML Linking Language (XLink) Version 1.0. W3C Recommendation (2001), http://www.w3.org/TR/2000/REC-xlink-20010627/

Dreyer, M., Bulatovic, N., Tschida, U., Razum, M.: eSciDoc - a Scholarly Information and Communication Platform for the Max Planck Society. In: German e-Science Conference, Baden-Baden (2007)

Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Retrieved from University of California (2000), http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

Hey, T., Trefethen, A.: Cyberinfrastructure for e-Science. Science 308(5723), 817–821 (2005)
Hey, T., Trefethen, A.: The Data Deluge: An e-Science Perspective. Grid Computing, 809–824 (May 30, 2003)
Klump, J., Bertelmann, R., Brase, J., Diepenbroek, M., Grobe, H., Höck, H.: Data Publication in the Open Access Initiative. Data Science Journal 5, 79–83 (2006)
Kunze, J.A.: Towards Electronic Persistence Using ARK Identifiers (July 2003), http://www.cdlib.org/inside/diglib/ark/arkcdl.pdf
Lagoze, C., Payette, S., Shin, E., Wilper, C.: Fedora: an architecture for complex objects and their relationships. International Journal on Digital Libraries 6(2), 124–138 (2005)
Moses, T.: eXtensible Access Control Markup Language Version 2.0. OASIS Committee draft 04: access_control-xacml-2.0-core-spec-cd-04 (2004)
Powell, A., Lyon, L.: The DNER Technical Architecture: scoping the information environment. JISC Information Environment Architecture (May 18, 2001)
Scavo, T., Cantor, S.: Shibboleth Architecture – Technical Overview. Working Draft: draft-mace-shibboleth-tech-overview-02 (2005)
Sollins, K., Masinter, C.: Functional Requirements for Uniform Resource Names. Request for Comment 1737, IETF Network Working Group (1994)