Abstract: This paper reports on the EASY On Fedora (EOF) project that aims at building the next version of the EASY repository software, an online repository tool for Dutch scholars in the humanities, arts and social sciences. This paper presents an overview of the software’s system architecture, the lessons learned from one year of collaboration with the German funded eSciDoc project, the use of a multi-tier architecture in a repository setting and our intentions of using portlets for a collaborative repository platform. This paper concludes with a vision that has evolved out of our involvement with repository software and a critical look at the current state of repository services.

1. Introduction – DANS, EASY and EASY On Fedora

DANS, Data Archiving and Networked Services, is an institute under the auspices of the Royal Dutch academy of Arts and Sciences (the Dutch KNAW) and is also supported by the Netherlands Organisation for Scientific Research (the Dutch NWO). Since its establishment in 2005, DANS has been storing and making research data in the arts and humanities and social sciences permanently accessible. DANS develops permanent archiving services, stimulates others to follow suit and works closely with data managers to ensure as much data as possible is made freely available for use in scientific research. An important activity carried out by DANS is the setting up, managing and continued improvement of the user-friendly archiving system EASY (Electronic Archival System). EASY is an online self-archival tool Dutch scholars can use to share their datasets and access others.

EASY has been running for over 3 years and is in need of replacement. A project was created with two primary requirements:

1. Creating a new version of EASY that is able to scale better and has an extensible architecture.

EASY has problems handling large amounts of datasets. Large collections were often stored on other information systems. EASY’s software architecture does not have clear defined extension points, thus extensions often became part of the system itself leading to tight coupling and low cohesion.

We held a user survey among Dutch scholars and archivists who use the EASY software regularly. A list of improvements that need to be made to the current state of the software and a list of possible future issues and wishes was compiled. The survey included a question on which areas we should improve the most. These were the most important areas identified (in order of importance):

- Performance
- Security
- Availability
- Searchability
- Interoperability

2. To build the new EASY using open-source, open standards repository components

EASY does not use any repository components. Common tasks like searching metadata and exposing common repository standards like OAI-PMH were all developed internally. It was recognized, however, that staying up to date with repository infrastructures and fulfilling the needs of EASY’s customers can be simplified by using repository components. That these components would need to be open-source and use open standards is part of DANS’ philosophy, as “DANS does not archive and develop technology on its own behalf, but in cooperation with numerous partners in the arts and humanities and the social sciences and in the interest of scientific practice in those disciplines as a whole.”
2. Our choice for the Fedora repository software

A first step in the project was to find a repository solution that could alleviate us of some of the common work needed for repositories. Among dozens of repository system we quickly narrowed down our search to reviewing the three most popular (open-source) repository systems to this day: ePrints, DSpace and Fedora. The conclusion of our review led us to believe that our requirements match the Fedora software best, mainly because Fedora possesses a generic repository architecture while still being compliant to the OAIS references model\(^6\) as defined by a recommendation of the Consultative Committee for Space Data Systems\(^5\). Fedora’s generic architecture enables one to build any kind of repository, be it a medical repository, a digital library or a cultural repository, whereas DSpace and ePrints are more or less out-of-the-box publications repositories. Fedora rightly stands for Flexible Extensible Digital Object Repository Architecture and has been described Fedora as an “architecture (that) is extremely flexible, and provides significant advantages as a platform on which to build other applications”\(^4\).

Fedora, conceptualized in 1997\(^4\), was one of the first of its kind. Repositories not simply as a name for a system where one may deposit something or other, but as an general-purpose repository server able to handle common tasks associated with these systems, comparable really to database servers, directory services, application- and web servers, with at its core not tables nor services or web pages, but digital objects. *Fedora handles common low-level repository tasks* (e.g. dissemination, versioning, semantics, replication, authentication and authorization, storage), but does not have any constraints on the type of data. Moreover, what makes Fedora an ideal repository system from a preservation point of view and sets it apart from the classical database systems is its ability to actively ensure the preservation of its data and metadata. *It is rare for a database to be preserved in an appropriate manner* (From Digital Volatility to Digital Permanence: Preserving Databases, 2003)\(^6\), yet a Fedora repository preserves its own data in an appropriate manner real-time. By always storing datasets and its metadata in XML on a low-level storage device Fedora’s highest priority is to preserve. If a software problem of any kind would occur, with Fedora or any of its components, Fedora is able to rebuild itself completely from what remained on your low-level storage device\(^8\).

Aside from the match with our requirements it was announced that in August 2007 that the Fedora team received $4.9M worth of new funding from the Gordon and Betty Moore Foundation to setup the non-profit Fedora-commons organisation\(^9\). Naturally we predicted a rapid growth of the software installed base and community. Indeed, the number of registered repositories using Fedora rose from 98 to 153 in 2008, several new Fedora events were organised world-wide by community members, one of which by DANS\(^10\), and a partnership was funded between DSpace and Fedora\(^10\).

3. High level repository services on top of Fedora – our choice for the eSciDoc infrastructure

Thus the EASY On Fedora project was initiated. After choosing the Fedora repository software we took a look at what others had created using Fedora to see if we could possibly reuse some of their work. We found that quite a number of high-level tools, services and front-ends have been built using the Fedora repository software. Only a small subset of these tools, services and front-ends are generic, reusable and still maintained by their creators and/or respective communities. Furthermore, *what exists now of reusable software based on or using Fedora is scattered over the Internet and virtually all are incompatible with each other*. One might find a citation service in one project and a duplication detection service in another, but in almost all cases one has to choose between one or the other. The Fedora-commons community ties these active sub-communities together, but as of yet offers no technical solution for harnessing for their collaboration. Efforts to create services for common tasks are still prevalent among projects using Fedora, but without a common platform neither common interfaces nor compatibility among these services is to be expected. Though they are not yet widely used, such platforms that enable collaboration do already exist. At the moment there are two such platforms that we know of: NCore\(^11\) and the eSciDoc infrastructure\(^12\). We actually never reviewed NCore as we became aware of its existence much later, but we quickly found a partner in the eSciDoc project.

The eSciDoc project, which gave birth to the open-source eSciDoc infrastructure is a German funded project. The Max Planck Society (MPS) and Fiz Karlsruhe received a 6 million Euro grant for 5 years for “building an integrated information, communication and publishing platform for web-based scientific work, exemplarily demonstrated for multi-disciplinary applications”\(^13\). Fiz and MPS pretty much took equal share in the project, but different, yet complementary, roles. Fiz Karlsruhe builds the software infrastructure (eSciDoc infrastructure). MPS works on several so-called solutions that make use of this infrastructure (eSciDoc solutions). By far the most comprehensive of these solutions is a web-application called PubMan\(^14\), a publication management tool comparable to such tools as DSpace and ePrints.
The eSciDoc infrastructure offers the most comprehensive set of coherent, (inter-)compatible and fully functional services using Fedora up to this date. Version 1 of eSciDoc, released at the end of 2008, contains a relatively large number of services (see Figure 1) that handle common low-level tasks, similar to the ones Fedora handles, as well as high level tasks.

---

![Diagram of eSciDoc Services](image)

**Figure 1** – a depiction of the services comprising eSciDoc version 1 made by the author

eSciDoc is repository middleware, a platform for repository services that use Fedora at its core. eSciDoc is a Service Oriented Architecture (SOA) where services are labelled to be of three types moving from generic (low) to specific (high): basic, intermediate and application. The basic services provide object access and security and are fundamentally data centric. Intermediate services are more specific to repositories and add function. The application services layer is a space left open for application developers to put there their own services in, the application services layer is like an empty canvas with infinite potential within its constraints. PubMan, the most comprehensive software solution for eSciDoc to date, has several application services that are specific to publication management, like a citation service, quality assurance and controlled vocabularies. Application services are not included with the eSciDoc infrastructure though and need to be obtained from the solution providers.

**Reasons we chose to use the eSciDoc infrastructure:**

- eSciDoc offers a lot of relevant high-level services that we can reuse.
- eSciDoc enforces a type of data model, called an object pattern, that fits perfectly with the EASY data model (see section 4).
- eSciDoc is a platform to build and share services on. At DANS we have a lot of different needs and are in the ‘business’ of creating services. eSciDoc would enable DANS to share these services with other eSciDoc users.
- Fiz Karlsruhe offered DANS early adaptors support and a memorandum of understanding.
- Both Fiz Karlsruhe and MPS have committed to work on eSciDoc until 2011 at least.
- Fiz Karlsruhe, though being a non-profit organization, allows for paid for services like support, custom development, training, and consulting.

---

### 4. eSciDoc object patterns

One of the reasons why we chose eSciDoc is the availability of object patterns (see Figure 2). Object patterns are essentially generic relation types that are predefined by eSciDoc. In short a digital object can be
a collection, an item or a component. Each of these objects is described by a content model object and has a context. The context is part of an organisational unit that describes the hierarchy of an organisation.

![UML diagram of the eSciDoc object patterns](image)

**Figure 2 – UML diagram of the eSciDoc object patterns**

In Fedora one needs to create these kind of abstract relationships ‘manually’ for digital objects have no logical relation with other digital objects. Digital objects are free floating concepts in the Fedora repository software. Through Fedora’s Content Model Architecture¹⁵, however, it is possible to define what kind of object (e.g. book, paper, research project, image) a digital object is. One does this by semantically relating the digital object to a content model object that describes that kind of digital object much like a class describes an object. The current latest version of Fedora, version 3.1, has content model objects that describe the dissemination behaviour and content of digital objects. Upcoming versions might also describe semantic relationships. Asger Blekinge–Rasmussen and Kjre Fiedler Christiansen, already successfully demonstrated this¹⁶. eSciDoc, however, has already been one step ahead for some time and describes abstract relationships, object patterns, between digital objects. These object patterns help find digital objects that logically relate to other digital objects without having to go through rounds of querying. Moreover, one can also rely on the eSciDoc infrastructure to maintain these relationships and optimize their usage. Thus it can be said that ‘with the separation of object patterns and content models and the creation of a content-model-aware middleware, we (eSciDoc) simplify the design and implementation of e-research applications dramatically.’ (from the eSciDoc project website)¹⁷.
4. The initial architecture of the new EASY repository infrastructure

When the requirements for the EOF projects were clear and we had chosen Fedora and eSciDoc to be the basis of the repository infrastructure our initial system architecture overview looked like Figure 3.

![Figure 3 – EOF initial system architecture](image)

**Presentation Tier**

We chose a portal to be the foundation for our presentation tier. A portal hosts the presentation for personalization and aggregation of portlets. Portlets are pluggable user interface components that are managed and displayed in a web portal (see Figure 4). The portal, much like the graphical user interface of an operating system, allows users to position, minimize and discard portlets.

Portlets seemed an ideal choice, because they provide the flexibility for users to customize the views they want to have on the repository’s content instead of those views being predefined. An archaeologist, for example, might want to see how data is geotagged on a map, thus he/she would open a GIS portlet in a data viewing context, while a sociologist might be much more interested in seeing previews of statistical data and would thus add a statistics portlet to that context. A portal enables one to pre-compose views out of groups of portlets for different groups and roles in different contexts, while still allowing users to manage their own environment individually. Furthermore, a repository’s user-interface as a customizable platform ensures the system does not overextend itself while handling many tasks for many disciplines. This prevents the user-interface from becoming bloated and overly complex with features most users do not need or understand.

Portlets are also able to run remotely while still acting as if they were running on the same machine. A portal is thus able to access various systems and present it as a single website. This means one or more portlets of the repository system might be embedded on a researcher’s personal website, as in the Dataverse

![Figure 4 – portlets in a portal environment](image)
We chose to use Portlets as defined by the Java Portlet Standard (JSR-168). The JSR-168 API enables interoperability of portlets between different Portal systems. We tested our Easy portlets on two portals to ensure interoperability: Jetspeed and Liferay.

Business tier
This is the heart of the EASY repository. It is made up out of two pieces: the Easy repository services and the Easy data model. The Easy Repository Services are repository specific, the Easy data model are content specific.

The Easy Repository Services contain general-purpose repository functions, like authentication and workflow, while the data models are object-oriented representations of the data model that are persisted by the lower tiers of the architecture (eSciDoc/Fedora). The main reason for this separation of concerns is extensibility and reusability. The user survey among EASY users (see the introduction) showed that EASY might need to serve several data models in the future or at least more advanced versions of the current EASY data model. If such a need would arise only a new data model would have to be build. As a next step this new data model could be represented by portlets that make use of its features.

We defined our data model objects according to the EASY domain model. Then we translated this to a Fedora/eSciDoc data model. This translation was made in such a way that the data model's objects can be entirely reconstructed from data that is stored in Fedora. Much like the OSI model where lower layers of the model are unaware containers for higher levels of the model, so Fedora is an unaware container of eSciDoc data and eSciDoc an unaware container for EASY data. Besides the data the schema of this data model is also entirely persisted in Fedora through Fedora content models.

Creating a usable data model needs insight into the workings of Fedora and eSciDoc for we often found that there are usually several strategies to implement a data model, none of them being necessarily the best. One strategy might be elegant, but technically problematic. There are performance, preservation quality and maintenance issues to consider. There is no set of rules that lead to the best possible Fedora data model.

Extensions
The initial EOF architecture has several extension points. Portlets allow different graphical user interface features. They do not necessarily have to connect to the EASY business tier and might even come from a different host. The business tier can be extended with new data models that need to be fit into the EASY repository system. And if necessary the eSciDoc system may be extended for increased control over the data or simply more generic approaches to repository access and dissemination.

5. Revision of the initial architecture

Portlets replaced by web framework
Just a few months into the project we dropped the idea of using a Portal for the simple reason that it was unnecessarily too complex and the ROI (Return Of Investment) too low. In fact all gains of this plan would return in the time after the project would have finished.

We tried to build JSR-168 portlets and tested them on Liferay and Jetspeed, but found the JSR-168 API lacking too such an extent that we needed to either work around it or use the API in conjunction with other API's like WSRP, thus adding to the complexity, and all this for doing tasks that would take virtually no time to do in any other web development environment!

JSR-286, the sequel to JSR-168, might have solved most of our problems, but at that time it was entirely unclear when JSR-286 was going to be finished. We had picked up rumours that it was not going to take much longer, yet no official roadmap could be found to back this rumour up. In fact JSR-286 had been in draft status for a long time, at least 2 years. We gave up on using portlets before seeing the final release of JSR-286 on the 12th of June 2008, five years after the release of JSR-168! Janus Boye wrote in a blog just a few days after the release:

"Standards generally go missing in this marketplace, but judging from the very limited attention this new version of the portlet spec has received, it makes me wonder whether the marketplace has already left the need for it in the dust. As a buyer the new industry standard might seem the preferred option over the many proprietary implementations that build on the shortcomings of JSR 168, but make sure to study the emerging implementations of the new standard carefully to avoid an early mover disadvantage."
The concept of portlets or, speaking in more abstract terms, embedded web applications will be a part of the future of the Internet, but if the Java Specification Requests are going to partake in this movement remains questionable. In the mean time we have decided to work with the Wicket framework\textsuperscript{23} that enables such modularity that if at some point in time we wish to switch to working with portlets the amount of effort would not be all that great\textsuperscript{24}.

**eSciDoc replaced by Fedora and our EOF high level services**

After almost a year of investing in eSciDoc we decided to stop using eSciDoc and switch to using Fedora directly. We had considerable difficulties getting the EOF project of the ground using eSciDoc. EOF’s budget was running out, its time frame needed extending, and all this while there was still little visible progress. In the mean time, though, we had gained significant knowledge of Fedora and felt that we could shorten the time needed for creating the EOF software by using Fedora directly.

Our three major reasons for replacing eSciDoc were:

1. **eSciDoc version 1 released late**
   eSciDoc’s release date had been extended from the beginning of 2008 until the end of 2008, beyond the first predicted end date of the EOF project. The third release of Fedora, which was planned to be part of the first release of eSciDoc, was also delayed for half a year. In the mean time we did not have a stable platform on which to build. This amounted to a lot of reworking and waiting. In the end we simply did not have the resources anymore to continue on this path, while there was still a definite need from within DANS to move forward as quickly as possible. We could not wait anymore.

2. **eSciDoc API complexity**
   At the beginning of the project, almost directly after we started using eSciDoc, we encountered a problem while trying to establish communication between our software and the eSciDoc services.
   eSciDoc services all have a REST and a SOAP API. And for all compound objects that eSciDoc sends over both these API’s it uses eSciDoc XML. Fedora exposes similar API’s, yet the difference between the two is that Fedora requires only one XML schema, whereas eSciDoc requires dozens, in most cases a few XML schemas per service. We had chosen an object oriented approach and therefore needed to marshal all this eSciDoc XML into objects. This turned out to be no straightforward job.
   At first we tried automatically generating objects, but this resulted in such unnecessarily complex objects because of XSD nesting that we had to retreat to building our own mappings between simple value objects and the eSciDoc XML. Next we employed JIBX\textsuperscript{25} to create mappings between generic eSciDoc value objects and the eSciDoc XML. Though this worked fine this required a lot of (human) insight into the eSciDoc XML to define appropriate value objects and it still was a lot of work.

3. **eSciDoc hides Fedora**
   eSciDoc wraps Fedora entirely, effectively hiding all Fedora API’s and services. A ‘user’ of eSciDoc therefore theoretically needs no knowledge whatsoever of Fedora to be able to use eSciDoc. Though this might seem an advantage we feel that the disadvantage of not being able to fully participate with the Fedora community is bigger than the advantages of abstraction.
   An advantage of this design choice would be that generic complexity is hidden from the user. eSciDoc effectively adds a layer of function on top of Fedora that allows a user to ‘speak’ in more natural terms to the repository services. Theoretically one does not need to know the more generic approaches employed by eSciDoc to be able to execute these functions. But we found in practice that this is not always true, at least not when one accounts for software quality. This discrepancy between theory and practice has been observed before by others and has been described before by Joel Spolsky who called it “the law of leaky abstractions”\textsuperscript{27}. Practically we found it easier and more practical to explain eSciDoc to new developers by introducing them to the concepts of Fedora first than by completely leaving out Fedora from the vocabulary.
   eSciDoc is highly coupled with Fedora, which reveals itself in a many of aspects of eSciDoc (e.g. terminology, architecture, features and performance). We found that understanding of Fedora improves the quality of the work done using eSciDoc.
   There are also a few disadvantages to this abstraction that need to be addressed. eSciDoc's object patterns, for example, are mandatory, which means that if those object patterns do not fit with your requirements at all (though because of its generic makeup that is unlikely) it might be better to use Fedora directly, which renders it impossible to use eSciDoc services.
   Also Fedora has a few features that eSciDoc does not have yet. Since eSciDoc does not allow direct access to Fedora’s features, they need to be ‘tunneled’ through eSciDoc services before they may be accessed.
The fact that Fedora is always accessed through eSciDoc’s services does however make it possible for eSciDoc to ensure its own (more advanced) authentication and authorization model.

The discussion on the advantages and disadvantages of this abstraction layer may need further elaboration, yet it is clear that there is not an effective symbiosis between eSciDoc and Fedora. From an end-user’s point of view the two systems are in fact competing instead of complementing each other. The benefits the eSciDoc community reaps from the Fedora community and vice versa are far more limited then they could have been.

6. In conclusion – evolving vision of repository services

The eSciDoc project has shown that a repository service in terms of being generic is able to exist on several levels by offering high-level services like workflow management, image manipulation and citation management next to low-level services like object storage and semantics. And while there is a definite need for high level services, Fedora seem to grow more low-level, more networked, more interoperable, offering the possibility to create high-level services, but not entertaining any or describing any method to collaborate between them.

Our work to build an online institutional repository serves a practical purpose today, tomorrow, however, it might be too much. As repositories evolve to be more interconnected high-level services are taken over by aggregating services. Eventually our classic institutional repository might be stripped down to being nothing more than another server part of a research network.

On the other hand we need the high level services of today to be part of that aggregating service of tomorrow and there is no end to the amount of high level services are needed and wished for. Therefore we need repository middleware, like the eSciDoc project, to form the platform from which to operate repositories.

The potential of interconnectedness of repository services and the internet are still largely untapped. Before we will see worldwide collaboration between repositories we have to see more standards emerging and being adapted, standards like the Java Portlets Standard. Yet often these standards address too few people’s needs and therefore accomplish too little. It is only logical to see the repository community shaping itself from the bottom up: low-level services first (e.g. persistent identification, metadata registries, metadata harvesting). It seems that a collaborative repository system that presents itself through decentralized web controllers and that is extensible on all layers as envisioned in this paper remains a challenge for the future.

References

1. DANS Research & Development factsheet (2008, June)
2. Fedora open source repository software. (2005, Oktober 28)
   Retrieved from http://www.fedora.info/pdfs/WhitePaper.10.28.05.pdf
5. History of the "Fedora Project" and "Fedora Commons"™ Names (2008)
   Retrieved from http://www.fedora.info/about/history.php
   Retrieved from http://www.fedora.info/download/2_1_1/userdocs/server/cmd-line/index.html#rebuild
8. “Gordon and Betty Moore Foundation Awards Fedora Commons $4.9M to start non-profit organization and develop open-source software for collaborative communities” (2007, August 13)
   Retrieved from http://www.fedora-commons.org/about/news.php#moore-grant
11 NCore website  http://ncore.nsdl.org/
12 eSciDoc website  http://www.escidoc.org/
14 PubMan website  http://colab.mpdl.mpg.de/mediawiki/PubMan
15 The Fedora Content Model Architecture (CMA)  (2007)
18 The dataverse factsheet
20 Jetspeed website  http://portals.apache.org/jetspeed-2/
21 Liferay website  http://www.liferay.com/
22 Boye, J.  (2008, 24 June) JSR 286: The last portlet standard?
23 Wicket website  http://wicket.apache.org/
25 JiBX website  http://jibx.sourceforge.net/
27 Spolsky, J.  (2002, November 11) The law of leaky abstractions