Towards format repositories for web archives

Niels H. Christensen

Dept. of Documentation & Digitalisation, Royal Library of Denmark[1],
Postbox 2149, DK-1016 Copenhagen K, Denmark.
nhc@kb.dk

Phone: (+45) 33 47 46 90
Fax: (+45) 33 93 22 18

Web archives face a formidable challenge regarding the handling of file formats. It is the thesis of this paper that this challenge could and should be met through the development of format repositories fit for that purpose. The "format challenge" for web archives - and its relation to software for viewing and converting digital objects - is analyzed in detail using methods from the field of programming language implementation. As a result of the analysis, we are able to list a number of specific requirements to a format repository. A format repository that satisfies these requirements can be integrated with a web archive's software and thereby provide it with automatic support for handling formats.

Keywords: web archiving, digital archives, format repositories, programming languages, T-diagrams

Introduction

Of all the kinds of digital archives in existence, web archives arguably face the largest challenge when it comes to the handling of file formats. Web material is usually harvested from any number of external sites that may each have their own ideas and policies about which formats to use. The web archive will often have no influence on these policies and must therefore accept material in any and every form. While every harvested digital object is certainly a bit stream that can be stored and preserved as such, the bit stream is of little value to anyone if the producer's intended semantics – as defined by its format - can not be determined and applied.

This challenge expresses itself concretely when we attempt to access an object contained in a web archive. We may want the object to be rendered on screen, to be manifested through other media (e.g. loudspeakers), or we may just want certain features extracted from it, but in any case we need our system to help us figure out

[1] This paper was prepared as part of the netarchive.dk project.

which applications are appropriate for processing the bits that make up the object. The choice of application will often be determined by the local setup of the computer that is used for accessing the object (the access machine). On the basis of a file extension or a mime-type, the local operating system or browser will pick a preferred application and use that for interpreting the given bit stream.

Such ad-hoc solutions to the format challenge will not stand the test of time. Formats and standards evolve over time, processing applications are regularly published in new versions, and new formats appear with their own new tools. If we want the ability to access both old and new digital objects from the web, we need to register current and past formats and applications; we need *format repositories*. And, conversely, format repositories should assist our web archives in solving the format challenge.

About this paper

In this paper we focus on how a format repository can be of use to the automated processes in a web archive, i.e. as an information system that can be used by programs running as services in the archive. Most format repositories (e.g. [Pronom,Ic,Wotsit,Mff]) contain detailed textual descriptions of formats that may also be useful to web archivists, but this aspect will not be discussed here. The contribution of this paper is a list of requirements to format repositories on the basis of an analysis of the format challenge for web archives. We hope that the resulting requirements will be useful input not only to format repository developers but also to organizations like IIPC [Iipc] that aim to coordinate developments of benefit to web archives. Because of the focus on processes – and because the requirements may be satisfied by many different designs – we do not formulate specific solutions in e.g. UML or ER-diagrams.

Terminology

As mentioned above there are many modes of access that can manifest a digital object to a human user; rendering a picture, playing a sound, printing part of a document to name a few. For simplicity we shall refer to all such processes as either *displaying* or *rendering* the digital object to the user, regardless of its content. The particular way in which the system chooses to display an object is referred to as a *display scheme*. Any application that renders an object to a human user will be called a *viewer*. *Processing applications* will mean a collection of viewers and converters.

Overview of the paper

This introduction is followed by three sections and a conclusion. In the first section we discuss how a web archive might look if it were completely independent of format repositories of any kind, i.e. if it was format-unaware. This leads to a clarifying formulation of the challenge that we would like format repositories to help us meet. The second section introduces a visual notation called T-diagrams. The diagrams are used as a tool for analyzing the format challenge in more details, and the section concludes with a functional description of "what is lacking" in a format-unaware web archive. Based on this, the third section formulates a number of requirements to a

format repository that aims to solve the format challenge for web archives. That sections ends with a very brief review of three existing designs for format repositories.

The rôle of formats

We now consider a web archive that is independent of format repositories. The archive contains a (potentially very large) number of digital objects, each of which is associated with an identifier and some metadata. The object's identifier should be unique in the given archive. The metadata could be structured in many ways, but for an archive of harvested web material it will often contain at least the http header that was sent with the object. As the archive was assumed to be independent of format repositories, no format is assigned to an object, although we as humans may infer format information from e.g. a filename suffix or a mime-type field.

To be of any use at all, the archive must support ways of human access to the stored objects, i.e. it must register a number of viewers. An archive with a very simple access strategy might only use e.g. the Unix *cat* program, which will print any bitstream as raw ascii, while a more ambitious archive could have hundreds of viewers including browsers, word processors, media players and text extractors. The viewers will often reside at a different piece of hardware than the stored objects – on an access machine rather than a storage server – but we can nonetheless assume that the archive associates each viewer with a unique identifier and some metadata. The viewers themselves are typically stored as executable files on the access machine. A viewer's identifier is often a path to the executable file. Viewer metadata may, as with the digital objects, be structured in many ways, but as a minimum it might encode how the executable file is invoked to view a given object ("scripting"). An example of these associations is the "Helper Applications" table in the Mozilla browser where viewers like Acrobat Reader and Real Audio are registered. In addition to the above information the Mozilla table also relates viewers to formats, thus fusing the above table with its own, local format repository. In the above we have not associated formats with viewers, as we are describing the format repository-independent web archive.

A final, relevant element of the web archive is its population of converters. A converter is an application that transforms a given digital object into a new one that potentially can be viewed with the same experience as the original. An example converter is the *pdf2ps* application that transforms *Portable Document Format* documents into an equivalent document in the *PostScript* format. Some converters deliberately reduce quality in the process, e.g. the *ps2ascii* converter deliberately removes fonts and figures that have no representation in the *ascii* format. A web archive does not necessarily need to employ any converters at all, but conversion can help limit the number of viewers needed and make more objects viewable. As in the case of viewers, each registered converter should be associated with a unique identifer and appropriate metadata. It is sometimes discussed at which time conversion should

be applied to digital objects: at the time of the object's ingest, at the time of a user's request for the object, or sometime in between? In this paper we shall not address the issue of conversion time but refer the interested reader to [Hff]. In the following discussion, conversion will be presented as if done "on-demand" at the time of a user's request for the object (also known as "migration on request"), but the results apply to other conversion strategies as well.

Summing up

To sum up the above, our format repository-independent web archive contains digital objects, viewers and converters. Each of these is associated with metadata and an identifier. When a user requests that a given object should be displayed, the archive has a large number of choices:

- Example A: It could invoke any of the known viewers and apply it to the object.
- Example B: It could apply any of the known converters to the object and then apply any of the known viewers on the resulting object.
- Example C: It could choose two of the known converters, apply the first converter to the object, apply the second converter to the output of the first converter, and finally apply any of the known viewers to the output of the second converter.

And so on – theoretically this list goes on forever. But note that in an archive with, say, 20 viewers and 4 converters the first bullet represents 20 possible display schemes, while the second bullet already represents 80 additional possibilites. The total amount for examples A, B and C is 420 possible display schemes.

The above description may seem surprising, as it implicates that e.g. Adobe's *Acrobat Reader* is a possible choice for displaying a *wav* sound file. But this is a direct reflection of the lack of formats: there is no coupling of objects, viewers and converters, so any combination is thinkable even though some of them will leave the requesting user far more satisfied than others.

So formats should be introduced in a web archive as a concept to somehow assign a preferred viewer to each object? Yes, but there may be more than one good viewer per object. One example is that *html* files may be equally well displayed by *Internet Explorer*, *Mozilla* and *Opera*. Also, there may be degrees of satisfactory display; e.g. *Notepad* would give an intelligible but far from perfect rendition of *html* files. So for the purposes of this paper we conclude the following:

The rôle of format repositories is

- to group objects that are expected to have similar behaviours wrt. viewers and converters, and
- to relate groups of objects to viewers/converters and asscoiated notions of quality.

T-diagrams for web archives

In the previous section we pointed to the strong connection between formats and applications for processing digital objects. In this section we shall look more carefully at this relation. Specifically we shall introduce an existing visual notation for the relationship.

Motivation: programming languages are formats

There is a strong similarity between the task of displaying a digital object from a web archive and the task of running a program. In order to run a program we must first establish the programming language, call it *L*, in which it is written in. When looking at it, a program is really just a collection of lines of words and symbols. In order to run the program it is necessary to establish what semantics should be given to these words and symbols, i.e. what the (intended) programming language is. Once this *L* is determined we need to pick a way of running the program as an *L*-application. Two popular ways are:

- To load the *L*-program into an interpreter for the language *L*. A programming language interpreteter is an application that is able to execute programs in its language. Interpretation is standard for scripting languages like *Javascript* and *Perl*.
- To compile – i.e. translate by automatic means - the *L*-program into a new program in another programming language, call it *M*, then execute the new *M*-program using an *M*-interpreter. This is standard for languages like *Java* and *Macromedia Flash.*

Interpreters and compilers are tools that support a given language, and any language can be supported by interpreters as well as compilers, although most languages have a standard execution approach. The important point is that as a process, there is no difference between interpreting a program and viewing a digital object. The subjective difference is that programs are generally far more complex than other digital objects. Also, as a process there is no difference between compiling a program and converting a digital object. The first bullet above matches perfectly to the pattern of our Example A. Likewise, the second bullet above matches perfectly to the pattern of our Example B. As with digital objects, the compilation chain could in theory be of any length. So to conclude on the above, the task of running a program mirrors the task of displaying a digital object.

The T-diagram notation

The design space for running programs has been analyzed extensively in the field of programming languages. For several reasons a wider spectrum of solutions have had to be considered than we do in this paper; among other things, the element corresponding to emulation is much more common in programming language implementation than in digital archives. To illustrate the design space, a visual notation was developed in the 1960's, called *T-diagrams* [Bra61,ES70]. We now introduce part of the T-diagram notation.

We shall need three symbols: one representing a digital object, one representing a viewer, and one representing a converter. The three symbols are shown in Table 1.

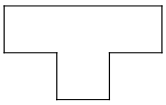
Digital object	Viewer	Converter
		

Table 1: Symbols used in T-diagrams.

In a T-diagram, these three symbols are combined into a figure that represents a process for displaying an object. For example, the diagram in Table 2(a) would represent displaying the digital object *report.pdf* using the viewer *acroread*. Viewers must always be placed under the object that is being rendered. Displaying the same object by first applying the converter *pdf2ps* to it, then rendering the result using *gv* would be represented by the diagram in Table 2(b). When an object is placed on the left side of the converter symbol, a new object is put on the right-hand side of the converter symbol. This anonymous, new object represents the output of the conversion. No viewer should be put under the original object because no viewer was applied to this object in the process. As a final example, the diagram in Table 2(c) represents the following display scheme for the same object: First convert *report.pdf* using *pdf2ps*, then convert the resulting object using *ps2ascii*, finally display the output of the latter conversion using *Notepad*.

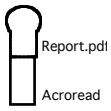
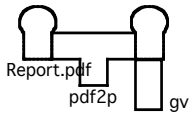
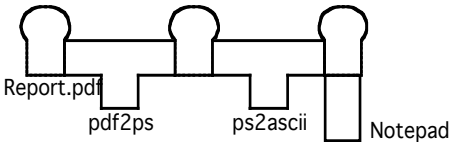
(a)	(b)	(c)
		

Table 2: Three display schemes for the object report.pdf, drawn as T-diagrams.

Tagging and Domino rules for T-diagrams

The three diagrams visualize our examples A-C of solutions for displaying digital objects in the previous section. And indeed, if our archive has 20 different viewers registered, we could draw 20 different diagrams in the form of Table 2(b).

To avoid diagrams that do not correspond to reasonable display schemes, the T-diagram notation has additional rules for how the symbols may be combined. Each

digital object appearing in the diagram must be assigned a format. This is done by tagging the object with a format identifier. Each viewer symbol must be tagged with the identifier of a format that it is a viewer *for*. Finally, each converter symbol must be tagged with two format identifiers: one for a format that it converts *from*, and one for a format that it converts *to*. The rules for combining symbols are exactly like in the game of *Domino*: adjacent symbols should have matching tags. Table 3 shows the diagrams from Table 2 with proper format annotations.

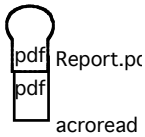
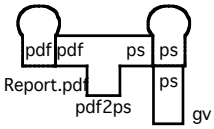
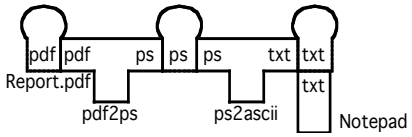
(a)	(b)	(c)
		

Table 3: The same three display schemes for report.pdf as in Table 2, with added format annotations.

The Domino rules should be no surprise. They simply amount to the following:

- An object in format *F* can only be rendered by a viewer for *F*.
- An object in format *F* can only be transformed by a converter that converts from *F* to some other format *G*.
- An object in format *G* can only be the output of a converter that converts from some format *F* to *G*.

There is one major point to this. Once a format has been assigned to the requested "original" object, we must tag the occuring viewers and converters appropriately. If we disallow *acroread* to be tagged with the *wav* format (as is reasonable), then a number of display schemes are disqualified by the Domino rules. In conclusion: a resonable tagging policy for objects, viewers and converters combined with the rules for constructing T-diagrams together allows an archive system to compute every reasonable way of displaying a given object.

Facilities of format repositories

In the two previous sections we established the following points:

- From the point of view of a web archive's automatic systems, a format repository should assist in grouping objects that are expected to have similar behaviours wrt. viewers and converters, and in relating the groups to these applications.

- The system can automatically compute every reasonable way of displaying a given object, if it is supplied with a resonable policy for associating formats to objects, viewers and converters.

On this basis we are now ready to specify some of the technical requirements that we would like a format repository to satisfy for the benefit of web archives. To support the automatic choice of display schemes, a format repository should provide programming interfaces to the following:

- **A list of acknowledged format identifiers**. Formats should of course be registered with official identifiers. It is important for a repository to establish an authoritative list that settles issues regarding consistency and level of granularity, e.g. are Word 5.0 documents and Word 7.0 documents of the same format?
- **A procedure for determining the format of a given object**. This is the "input" to the T-diagram. The format should be determined on the basis of the object's metadata, or the object itself, or both. It would of course be useful if the procedure was expressed as an actual program that computed the format, but a programmed procedure would also be harder to integrate into individual archives. Moreover, a programmed procedure may also be harder to keep up to date.
- **A table of acknowledged viewer-format combinations**. This table expresses the reasonable tagging policy for viewers. For example, this table could include the combination of *acroread* and the *pdf* format, but should not include the combination of *acroread* and the *wav* format. When the table is to be used in a web archive, the viewer identifiers of the format repository must be integrated with the viewer identifiers of the archive.
- **A table of acknowledged converter-format-format combinations**. This table expresses the reasonable tagging policy for converters. If, for example, the table includes the combination of *pdf2ps*, the *pdf* format and the *ps* format, this expresses that *pdf2ps* is able to convert a *pdf* file to a *ps* file. When the table is to be used in a web archive, the converter identifiers of the format repository must be integrated with the converter identifiers of the archive
- **Updates of all lists, procedures and tables on a regular basis**. New formats, viewers and converters appear all the time, so it is important to keep the authoritative lists up to date.

The quality rating space of "acknowledged" vs. "not acknowledged" is, however, not fine-grained enough. For example, a list of "acknowledged combinations" fails to suggest which of the schemes in Table 3 should be preferred for *pdf* files. A hard-and-fast rule like "use a minimum number of conversions" may not only be misguiding in certain cases, it also fails to point out a preference when there are two acknowledged viewers for a given format. We therefore suggest that each row in the two tables required above is extended with a rating of the expected quality of the listed combination. The set of possible values for this rating may be large or small but it should definitely be more fine-grained than "acknowledged"/"not acknowledged". The values may simply be numerical (e.g. 0%-100% quality) or they may be structured (evaluating different aspects of the digital objects separately). These issues

are discussed elsewhere in the literature [RR04], and we simply state the following, final four requirements to a format repository's programming interface:

- **An appropriate space of values for display quality ratings**. As discussed above, the value space should be rich enough to express different acceptable quality reductions in viewers and converters.
- **Ratings of viewer-format combinations**. Each combination should be assigned a quality rating expressing how well the given viewer renders the given format.
- **Ratings of converter-format-format combinations**. Each combination should be assigned a quality rating expressing how well the given converter conserves content quality when converting between the two given formats.
- **A procedure for "adding" several ratings into one rating**. As an example, look at the T-diagram in Figure 3(b). The ratings of the *pdf2ps-pdf-ps* converter combination and the *gv-ps* viewer combination should somehow be composed ("added") to produce a quality rating of the entire process. The design of this procedure for composing rating values should of course be closely coordinated with the design of the value space itself.

The requirements vs. three existing designs

The GDFR is a multi-institutional project, the aim of which is to construct a global registry of digital formats. The GDFR data model (version 3, see [Gdfr]) provides a list of formats and advanced features for identifying the format of an object. It is also possible to associate to a given format any number of applications "using" this format, but there does not seem to be a defined way of realizing a converter-format-format table. An implementation inside the data model would probably need to use proprietary instances of the *FormatRelation* type. Implementations of quality rating are also not part of the standard model.

PRONOM [Pronom] is an advanced format repository published on the web by the National Archives in the UK. The system is able to provide format lists as well as tables of viewers and converters with associated evaluations of quality. At the moment, PRONOM does not seem to offer a mechanism for determining the format of a given digital object. It is also not clear to us whether there could be a procedure for adding PRONOM's quality ratings in order to asses the quality of an entire display scheme.

The Preservation Layer Model (PLM) was presented in [Ibmkb]. A central concept in the model is *view paths*, a notion very similar to our display schemes. A view path is an ordered list of layers like *operating system layer* or *application layer*. Every digital object must be rendered "on top" of a compatible view path – a tower of viewers. In this paper, we chose to leave out this "vertical aspect" of T-diagrams to focus instead on interactions between viewers and converters. We feel that the two concepts should be united in future work. The PLM provides lists of formats and viewer-format tables. Format identification is external to the model and there are no explicit structures for converters. The model does not seem to have a structured quality measure, although this may be implementation-dependent.

Conclusion

We have argued that web archives face a great challenge when it comes to the handling of file formats. The challenge and its relation to software for viewing and converting digital objects has been explained, and we demonstrated how the situation mirrors the task of running a program. On this basis we analyzed the challenge using methods from the field of programming language implementation. As a conclusion we were able to specify a list of requirements that we would like to see satisfied by a format repository. Such a repository could be integrated with a web archive and thereby provide it with automatic support for handling formats.

Future work

To keep the presentation simple, we did not present all elements of the T-diagram notation. When extended with the one symbol missing in our presentation, the diagrams can not only be used to describe all display schemes involving emulation and layers of abstraction, they can also serve as a vehicle for discussing the software and hardware dependencies of a given strategy for logical preservation. Giving a full presentation of T-diagrams in the context of web archives would be interesting future work. We would also like to develop examples of detailed technical formulations of our requirements in the form of programming interfaces.

Acknowledgements

The author would like to thank Steen Sloth Christensen and Tue H. Larsen (Royal Library of Denmark), Lars Clausen (State and University Library, Denmark) and Neil D. Jones (DIKU) for useful discussions on the subjects of this paper. The author also thanks the anonymous referees for constructive comments.

References

[Pronom] Public Record Office Pronom, URL: http://www.records.pro.gov.uk/pronom/
[Ic] "The top file extensions Windows site", URL: http://www.icdatamaster.com
[Wotsit] "Wotsit's Format", URL: http://www.wotsit.org
[Mff] "My File Formats – the programmers file format collection",
URL: http://www.myfileformats.com

[Iipc] Internation Internet Preservation Consortium, URL: http://www.netpreserve.org/

[Hff04] Lars Clausen: "Handling File Formats", URL: http://www.netarchive.dk/website/publications/FileFormats-2004.pdf

[Bra61] Harvey Bratman: "An alternate form of the 'uncol diagram'", Communications of the ACM, 4(3):142, 1961.

[ES70] Jay Earley, Howard Sturgis: "A formalism for translator interactions", Communications of the ACM, 13(10):607-617, 1970.

[RR04] Carl Rauch, Andreas Rauber: „Towards an analytical evaluation of preservation strategies", URL: http://www.erpanet.org/www/products/vienna/slides/erpaTrainingVienna_Rauber.pdf

[Gdfr] Global Digital Format Registry, URL: http://hul.harvard.edu/gdfr/

[Ibmkb] "Preservation requirements in a deposit system", report no. 3 in the IBM/KB long term preservation study, http://www-5.ibm.com/nl/dias/preservation.html