Adobe Advanced Annotations (A^3)

Erik Wilde (mailto:pdf@dret.net)
Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology (ETH Zürich)
ETH-Zentrum, 8092 Zürich, Switzerland

May 2002

Abstract

PDF in its current for has rather weak support for annotations. This paper describes a usage scenario and how advanced annotations support could make using PDF (and possibly other Adobe applications) more productive. Starting from these observations, different problems are described which could be solved based on different evolutionary steps of the annotation architecture, which has been dubbed "Adobe Advanced Annotations (A^3)". Following this scenario, some design approaches and a number of implementation issues are discussed.

Contents

1 Introduction

This document is intended as a motivation to improve the annotation support in PDF (and maybe other applications of the Adobe product line). Starting from a scenario described in Section 2, it is described how different evolutionary steps of advanced annotation support could improve the usefulness of PDF. From a more conceptual point of view, Section 3 describes some of the concepts which could be of interest. Section 4 then investigates what kind of implementation issues would be raised when implementing these concepts. Finally, Section 5 gives some concluding remarks.

2 Scenario

Recently, I worked with a publisher on a book, which I authored with David Lowe [20], a colleague working in Sydney. We were using the LaTeX formatting system, which was a choice we made because of its availability on a multitude of platforms and our personal preferences. In the following list, I shortly describe the steps of the publication process:

1. *First Submission of the manuscript.* I prepared a version of the manuscript for review and submitted it as PDF to the publisher. The publisher forwarded the PDF to five technical experts who reviewed the book. In return, I received five Word documents, containing some general remarks as well as very specific comments (using page/paragraph numbers to roughly identify the content). Integrating these comments therefore involved five passes over the document, and resulted in a lack of control which comments were addressed and which not.

2. *Submission of the final manuscript.* After the review phase, I prepared a double-spaced version of the manuscript for review and submitted it as PDF to the publisher. The publisher forwarded printed copies to the copy editors, because they claimed that the copy editors did not have the software for annotating the PDF.

3. *Review of the copy edited manuscript.* The publisher photocopied the copy edited pages, and sent a set to each of the authors (using a messenger service). The publisher requested to get back only one set of reviewed pages, and because of the time constraints we had to find a way to make this possible. In the end, my co-author created simple text files with his reviews (containing page/remark descriptions), and I scribbled these reviews as well as my reviews on the pages, which I then sent back (again using a messenger service) to the publisher.

4. *Creating the formatted version.* Based on the reviewed and copy edited pages, a typesetter created the final version of the book (using a new set of LaTeX style files and making all changes by hand). From this final version, a PDF was created and sent to the authors by Email.

5. *Review of the formatted version.* The publisher requested that we print out the PDF, and make the changes on the pages and then send them back using a messenger service. Again, the publisher claimed a lack of software support for working with annotated PDF. Based on our reviews, the final version has been prepared and printed.

Looking at this scenario, first of all life could have been a lot easier if we simply had used the annotation features available in PDF today. The reason for not doing this was the publisher's claim that not all people involved in the production (the publishing house, as well as external copy editors and typesetters) were able to work with annotated PDF. Because the workflow in this particular scenario involves a lot of changes to the underlying document (preparing the reviewed and final manuscript, submitting a double-spaced version, creating a published version), the overall gain of using advanced annotations is smaller

than it could be, because annotations are tied to the PDF. If we assume that annotations are supported throughout the whole publication chain (as described in more detail in Section 3.1), then annotation support would be even more beneficial. The following sections discuss all possible alternatives.

2.1 Current Annotation Support in PDF

Partly, the cumbersome publication process was not a result of technical limitations of the current version of PDF, but of the failure to use these features properly (or the lack of required software). Looking at the current support of annotations in PDF (based on Acrobat 5.0 features), the following way of interactions in the publishing scenario demonstrate the benefits and limitations:

1. *First Submission of the manuscript.* The submitted version could be reviewed in PDF, and all reviewers could submit their annotations as FDF files to a central annotation server maintained by the publisher. The annotations could then be merged into one reviewed PDF containing all annotations. This merged PDF could then be sent to both book authors.

   - *Solved Problems:* With the merged annotations, there is no need to process the manuscript five times, all annotations can be addressed using one pass. The publisher has control over the annotations as individual FDF files.
   - *Remaining Problems:* The annotation model itself is rather limited, there are no concepts for different classes of annotations (such as critical, important, peripheral, or other user-defined criteria), there is no generalized annotation model (for annotations including any number of resources, such as two URIs and a comment saying "did you look at these pages?", or an annotation pointing to two parts of the PDF, saying "these two parts seem to be contradictive"). Also, it is not possible to filter annotations according to user-defined criteria, such as author or class. Annotations of annotations are not possible as well, making it impossible to address a reviewer's comment by adding an annotation to it.

2. *Submission of the final manuscript.* Proofreading (checking spelling, punctuation, and grammar) is a task that has a lot of tradition, uses its own syntax[1] (which varies among countries and users), typically creates a lot of annotations per page, and therefore is hard to support electronically. Copy editing, however, in many cases can be better handled by using electronic tools, so typical copy editing remarks could be submitted electronically.

   - *Solved Problems:* If copy editing annotations are submitted electronically, they can be collected and distributed as FDF files.
   - *Remaining Problems:* The lack of support for annotations semantics makes it impossible to create different kind of copy editing annotations (for example identifying a special kind of annotation which absolutely requires to be addressed by the authors, while many other annotations may be left as they are). More generally speaking, it is my guess that proofreading will remain paper-based for a long time to come, so the basic problem in this step is whether it is possible to separate proofreading as being paper-based, and copy editing as being PDF based. It is not always possible to clearly separate proofreading and copy editing.

3. *Review of the copy edited manuscript.* Getting back the copy edited pages in annotated PDF rather than hard copies would have made it much easier to find annotations, to remove processed

[1] http://www.nisus.se/proof/ lists some of the symbols together with rather interesting thoughts about proofreading symbols for electronic documents.

annotations, and to exchange information with my co-author. We could both have worked in parallel, and then send our annotations of the copy editing version back as FDF to the publisher. The publisher could then merge the FDF files with the copy edited manuscript.

- *Solved Problems:* By distributing the copy editing annotations electronically, both authors would have been able to work on the copy edited manuscript in parallel, and the resulting FDF files would make it possible to synchronize this parallel work.
- *Remaining Problems:* PDF currently does not allow annotations of annotations, and this is a serious limitation, because many copy editing annotations need to be reviewed and annotated. Furthermore, because FDF files enforce a coarse granularity, parallel work is likely to produce conflicts, which will only show up after the author's FDF files have been merged together.

4. *Creating the formatted version.* If the typesetter had the reviewed copy edited pages in electronic from, it would be easier to go through the PDF version and make the necessary changes in the typesetting program. Remaining questions could easily be solved by creating annotations, rather than faxing pages.

   - *Solved Problems:* Using PDF's annotation features, it would be easier to keep track of the reviews. Furthermore, remaining questions could be easily addressed by exchanging additional FDF files containing questions.
   - *Remaining Problems:* It is still necessary to exchange FDF files in case of remaining questions (and PDF's current limitation of not being able to make annotations to annotations again is a problem), rather than using a linkbase as central point where questions can be submitted and actions may be generated in response to the questions. Furthermore, typesetting requires the use of two program, PDF for looking at the reviews, and the typesetting program for creating the formatted version.

5. *Review of the formatted version.* Instead of getting PDFs and sending back hard copies of the pages where we wanted some changes, we could have created PDF annotations and send them back as FDF files to the typesetter. The typesetter could then go through the annotations and make the final changes before publication.

   - *Solved Problems:* By sending FDFs instead of hard copies, it would be easier and faster to get the final reviews to the typesetter.
   - *Remaining Problems:* Synchronization with the co-author while reviewing the formatted version is hard, but would be useful, to see the changes he has made, and thus to avoid repeated or conflicting annotations. Furthermore, the restricted nature of PDF's annotations makes it impossible to make ceratin kinds of annotations, such as "make these three parts of the text look identical", and attaching three pointers into the document to that annotation. Finally, the typesetter must work in parallel with the reviewed PDF and the formatting program, because the annotations are only available in the PDF.

It can be seen that PDF's current annotation support can already provide a lot of benefits, when being used whenever possible. However, there also remain a lot of problems to be solved, in particular problems related to workflow issues, where PDF's current model of exporting and submitting FDFs is problematic, because there is not central place where the workflow could be controlled or managed.

2.2 Advanced Annotation Support in PDF

If we assume that PDF's annotation model is extended to implement an advanced annotation model (which is limited to PDF only) as described in Section 3, then we can expect the following improvement in the scenario:

1. *First Submission of the manuscript.* Advanced annotation support would enable reviewers to create a much richer set of annotations, such as multi-ended annotations and annotation classes. Furthermore, if linkbase access for the publisher's linkbase would be enabled, and the reviewers work online, then the annotations of other reviewers would become visible as soon as they have been submitted to the linkbase[2]. When the reviewers have finished their review, the linkbase is locked for write access for the reviewers.

   - *Solved Problems:* The advanced annotation model is more dynamic for reviewers, since annotations can be submitted on a per annotation basis to the publisher's linkbase. Even if they are still submitted in one batch, the advanced annotation features can be very useful, by creating annotations which are structurally and semantically richer than today's annotations.
   - *Remaining Problems:* Even though the annotations are using a richer model, I still have to scan the PDF for annotations, and then make appropriate changes in my manuscript, using a different application.

2. *Submission of the final manuscript.* While proofreaders very likely will not be supported by electronic means, copy editors may use advanced annotations to create annotations of different classes, or annotations using advanced structure (such as connecting two part of the manuscript and making a comment about repetitions or contradictions). Additionally, the annotations can be uploaded to the publisher's linkbase, and the authors can use the linkbase to work through the reviews. The linkbase may provide additional support for controlling the process of reviewing, by checking whether all annotations have been addressed by the authors.

   - *Solved Problems:* Copy editors could create better annotations, and the control over the copy editing annotations can be centralized in the publisher's linkbase.
   - *Remaining Problems:* If the annotation features are only part of PDF, then the choice of tools is limited. This is a problem for the proofreaders, because they most likely require highly specialized tools (if they will use electronic tools at all) to accommodate the large amount of small annotations that proofreading produces.

3. *Review of the copy edited manuscript.* Rather than reviewing the copy edited manuscript in separation, using a linkbase it would be possible to achieve annotation-level granularity, so that each annotation becomes visible to the co-author as soon it has been submitted to the linkbase. This is likely to avoid many conflicts.

   - *Solved Problems:* Because of the linkbase as synchronization point, conflicting annotations are much less likely to appear. The publisher also has better control over the overall progress of the review process and may even implement automated control and reminder features in the linkbase.

[2] For external reviewers, this might not be realistic. However, for inhouse reviewing and a more tightly coupled review process, this feature might become very helpful.

- *Remaining Problems:* In offline scenarios, the advantages of the synchronization provided by the linkbase do not apply. Thus, conflicts are still possible and there must be ways to detect and handle them.

4. *Creating the formatted version.* If the reviews of the copy edited pages were stored in a linkbase, it would be easier track the review process. Furthermore, because annotations can be annotated, the typesetter could easily create annotations, if there were any remaining questions regarding the final version.

   - *Solved Problems:* Because of the more powerful linking model, the author's review annotations can be annotated by the typesetter, which is very useful for creating follow-up questions. The linkbase, hosted by the publisher, could be used to implement workflow support, for example by creating email messages to the authors for the questions submitted by the typesetter.
   - *Remaining Problems:* Because the review process is limited to PDF, the typesetter still has to use two applications when typesetting the document, PDF for viewing the reviewed version, and the typesetting application for creating the formatted version. Additionally, it will necessary to keep track of the proofreading annotations, which will probably exist on hard copies only.

5. *Review of the formatted version.* Advanced annotations would make it easier to collaborate during the reviewing process, by jointly using a linkbase and utilizing the linkbase as the synchronization point. Advanced annotation features such as threaded annotations could be used to discuss certain annotations, and mark them as finalized when the issue is resolved. The publisher may track the progress by extracting the appropriate information from the linkbase.

   - *Solved Problems:* Collaboration becomes much easier if features such as annotations to annotations and threaded annotations become available. Furthermore, certain kinds of annotations would be possible which currently cannot be used, such as multi-ended annotations.
   - *Remaining Problems:* Because the annotation features are limited to PDF, the typesetter is forced to work with two applications simultaneously, the PDF application and the typesetting application.

While a number of improvements can be achieved with an advanced annotation model in PDF, there still are some problems left, in particular concerning the integration of other applications which are also part of the workflow, and do not have access to the annotation information.

2.3 Advanced Annotation Support in Publishing Chain

If we assume that the advanced annotation model not only is a feature of PDF, but implemented in a way which makes it truly open and interoperable across various platforms, then an even better integration of annotation functionality into the workflow becomes possible. An example of what this could mean is to think of annotation support in FrameMaker, where annotations can be handled inside the application and exported to PDF. Conversely, annotations could be imported by using the XML support of FrameMaker, so that annotations could be exported from PDF and imported back into FrameMaker, where the document originated.

1. *First Submission of the manuscript.* Using the publisher's linkbase, I could access the annotations from within my authoring publication (instead of a PDF viewer) and review the annotations in the original manuscript. If required for offline work, I can download all annotations from the linkbase to my authoring application.

- *Solved Problems:* Not only can I review all annotations at once, I can also make the changes to the manuscript, thus keeping all the relevant information together.
- *Remaining Problems:* I need to have an application supporting the linkbase's link model, and the annotations must be robust across application boundaries.

2. *Submission of the final manuscript.* Submission and copy editing of the final manuscript could probably be handled in PDF without problems, but if proofreading is done electronically, it will very likely require specialized tools.

   - *Solved Problems:* If the proofreading tools support annotation export, the proofreading annotations can be imported into the PDF manuscript (for most views of the manuscript, however, it will be necessary to hide these annotations, because there are a lot of them and they will probably clutter the display).
   - *Remaining Problems:* It is unclear if (and when) electronic proofreading will become a reality.

3. *Review of the copy edited manuscript.* Because this step only uses PDF for reviewing the copy edited manuscript, there are no additional benefits of supporting PDF in the publishing chain.

4. *Creating the formatted version.* If annotations were supported by the typesetting application as well, the author's review annotations could be imported into the typesetting application.

   - *Solved Problems:* There would be no need for the typesetter to use two applications for creating the formatted version in the typesetting application.
   - *Remaining Problems:* As long as proofreading is done paper-based, there still is the need to use the proofreaders' hard copies to go through the proofreading annotations. Furthermore, if the authoring applications used by the authors and the typesetter do not share a common document model, there still is the need to import the text into the typesetting application using rather simple mechanisms. However, if the text is imported by cut and paste or import from PDF, then the links could be imported as well.

5. *Review of the formatted version.* The authors' reviews can be exported from the PDF and imported into the typesetting application, and the typesetter can work with that application only to create the final version of the the formatted manuscript.

   - *Solved Problems:* By sharing a common link model, users can switch applications without using the annotations, in this case switching from PDF to the typesetting application.
   - *Remaining Problems:* The typesetter needs to use an application that supports the import of annotations. As a fall-back solution, the typesetter has to use the PDF as well as the typesetting application when preparing the final version of the the formatted manuscript.

The vision of a shared annotation model among multiple applications clearly makes more interesting workflows possible than advanced annotation support in PDF only. It is clear, however, that a shared annotation model will have its shortcomings and limitations, but at least as option for further evolution of a PDF-only annotation model it should be kept in mind.

3 Design Issues

The design of A^3 is a non-trivial task, in particular if the annotation model should be general enough to allow later extensions, and specific enough to allow application-specific handling of certain features. In the following sections, I give a short sketch of the work that is necessary to design A^3.

3.1 Document Model

The *Extensible Markup Language (XML)* [4] is the most important standard for structured data today. While it is perfectly possible to use other kinds of data structures (which may even be isomorphic to XML), using XML offers many advantages with regard to available tools, advanced standards using XML as their foundation, and the possibilities of integration into existing XML-based solutions. While PDF probably will not become XML-based in the near future, the annotation model could still use PDF by building on top of a XML-like view of PDF. It is, however, essential, that the mapping from PDF to an XML-like view is well-defined and respects at least the core XML standards such as XML Namespaces [3] and XML Base [13]. Basically, there should be a view of PDF that either uses the XML Infoset [6] or the DOM [12].

Building on top of a cleanly defined document model becomes essential when working with annotations. Annotations to PDF documents have locators pointing into the PDF to a certain fragment of the content, and great care should be taken when creating these fragment identifiers. If we think of an annotation model shared by multiple applications, it must be possible to import the annotations from a PDF document into the FrameMaker from which the PDF was produced. If this import operation should be as robust as possible, there must be a way to deduce from the PDF's content from which part of the FrameMaker content it has been produced. This could be done in form of application-specific hints, where FrameMaker assign IDs to PDF objects, which can then be used to backtrack the fragment identifiers from PDF to FrameMaker.

This certainly is not a trivial problem, but its clean and extensible solution would make workflow applications possible which cannot be implemented today. I therefore think that a well-designed document model as the foundation of A^3 is essential.

3.2 Link Model

Currently, two W3C technologies exist which can be used to implement links. One is the fairly general *Resource Description Framework (RDF)* [11], which is a representation for metadata of any kind. Viewing link information as a special kind of metadata, it is possible to represent link information as RDF data.[3]

There is another W3C standard which has been created explicitly to represent link information, which is the *XML Linking Language (XLink)* [9]. This standard describes how to represent link information in XML. A complementary standard is *XPointer* [10], which describes how to point into XML documents, in essence describing the format for XML fragment identifiers [15]. XPointer could be used for fragment identification if the document model would be based on XML, otherwise a proprietary fragment identification scheme could be defined[4].

XLink defines a basic link model with a few semantic attributes, but it leaves a lot of room for extensions, and this room should be used to describe the links in a semantically richer way that is possible by XLink only. This way, A^3 would still conform to XLink, but would provide a lot of benefits through a richer semantic model (for example, describing authorship and access restrictions for annotations). The following list describes some of the features that the A^3 link model should provide:

- *Multi-ended annotations.* Currently, PDF annotations are links that connect two resources, a part of the PDF document, and a piece of text. Multi-ended annotations are based on a link model that

[3] There even is a W3C note [8] describing how XLink data can be converted to RDF, by interpreting the XLink information and generating RDF statements from it.

[4] XPointer does not contain any information to check for a fragment identifier's consistency, which could be implemented using checksums of documents or fragments. I therefore would vote for not using XPointer, or at least for extending XPointer (which is possible), but this issue should be investigated in great detail, because it is one of the most important questions when thinking of workflows and annotation export/import.

allows any number and any kind of resources to be associated, where different classes of resources can be regarded specially:

  - *Parts of the PDF the annotation is presented with.* It may make sense to treat PDF locators to the current presentation context specially, such as displaying a note that the annotation has several anchors in the document (these anchors can be on separate pages for an annotation making a statement such as "did you notice that these two paragraphs contradict each other?").
  - *Parts of other PDF documents.* Since Acrobat will be the presentation platform of choice for A^3 documents, special care should be taken of locators into other PDF documents than the current presentation context of the link.
  - *Embedded Text.* Currently, this is the only resource type supported by PDF annotations. Because is this case the resource is an integral part of the annotation, it must be handled in a special way. The link model must make it possible to use the text (or parts of it) of an embedded text resource as the anchor for another link.
  - *Other resources.* Other resources should be allowed as well, and treated in a way that is specific to the presentation platform. Maybe A^3 should provide a way to package other resources with the PDF and/or links they belong to, such as images that have been associated with an annotation (if they are packaged with the annotation, they can be transmitted with it, otherwise they need to be have an URI and must be accessible using that URI).

  This list of different classes of resources is purely informational, technically speaking the A^3 link model should be multi-ended, and allow resources of any kind to be associated. This is the most general linking model defined by XLink, and A^3 should take it and make some specific decisions about the handling and presentation of these links.

- *Threaded annotations.* Since one of the main selling points of A^3 should be workflow and collaboration support, it would be a very useful feature to allow annotation threading, as for example implemented by W3C's Annotea.[5] In this case, annotations and annotations of annotations and so forth are being viewed in the same manner as mailing list archives, representing the relationship of annotations by providing a thread-based view. The thread graph can be constructed from the dependencies of annotations, and the feature should be directly supported in the GUI (for example by providing a "create reply" menu point when viewing an annotation).

- *Annotation access rights.* XLink does not contain any information about a link's owner of the access rights for the link. A^3 could introduce ownership and access right semantics, where the access rights could be set in an annotation-specific way, such as "read-only", "may add locators", "may create follow-up", "may change embedded text", and similar rights. The ownership and access right model should be accompanied by a role model, which assigns access rights to individuals or roles, and roles may be played by several individuals (this is particularly useful in workflow scenarios). It is not required that all applications support this model, but it should be possible to include this information in annotations, so that scenarios requiring this level of control may use it.

This is a first list of features which should be part of A^3. It is important to make a clean separation between conforming to standards (such as XPointer for XML fragment identifiers and XLink for links), and A^3-specific extensions, such as PDF fragment identifiers and A^3 link semantics, which are specific to the A^3 architecture. This way, it will be possible to enable open interchange between applications (for

[5] A shared annotation platform based on Web technologies such as RDF and XPointer, more information can be found at http://www.w3.org/2001/Annotea/.

example, by simply exporting PDF annotations as XLinks), as well as to support advanced functionality, such as access control and threaded annotations.

3.3 Distribution

So far, I have only discussed the links themselves, and it is has not yet been described where they should be stored. While embedding annotations into a document is the easiest way to use them, and to guarantee consistency of the annotations and the document, it also severely limits the possible application scenarios. One of my central research interests in the last years has been the question of a distributed way of handling links and documents, and when applying this scenario to A^3, it becomes obvious that the concept of an *annotation server* (from here on called a *linkbase server*, because this is the term that I use throughout my publications) is an interesting one.

The basic idea is to treat annotations as first-class objects, which are handled as self-contained entities. They may be embedded within documents, but this is not necessary. XLink supports this architecture, but does not say anything about how links will actually be distributed and retrieved by an application. The master's thesis of Christian Stillhard [17], one of my former students, describes requirements and a first approach for a linkbase access protocol. We have modelled linkbase access as a Web service, using SOAP [14] for server/client communications. Basically, linkbase access is the same as database access, but it is based on specialized data model, which are links. We plan to push linkbase access as an action item within W3C, and I believe that linkbase access will become a reality.[6] Our proposed linkbase access protocol is extensible, so that not only XLink-specific features may be part of a query, but also linkbase-specific features, such as the extensions defined by the A^3 architecture.

Linkbases and distributed annotations require a lot of work and standardization is only starting, but at least the scenario should be kept in mind, so that later version of A^3 may support linkbases, while the initial release could implement the advanced link features, but not link distribution.

4 Implementation Issues

The features described in Section 3 will not be very easy to implement, and require an extensive design phase to make sure that later evolvements such as distribution and cross-application annotations will be possible. Assuming that the document model has been designed in a way that A^3 is appropriately supported, the following sections shortly describe some of the implementation issues of the A^3 architecture.

4.1 Server Side

A linkbase server basically is a specialized database providing a service through a specialized access protocol [22, 23, 21]. All that needs to be specified is the access protocol, and from the architecture point of view, it is not important whether the database serving the linkbase protocol requests is using SQL, XML, or some other data and query format. Mapping primitives of the access protocol to other access mechanisms is not very hard, because the linkbase query mechanism is rather simple. We have done this in a prototype by mapping the queries to XSLT [5] executing the queries, but creating a mapping to SQL or other popular query languages (such as the upcoming XQuery [1] language for querying XML) will be a very manageable problem.

Thus, implementing a linkbase becomes a problem of implementing the Web service described by the linkbase access protocol, and this Web service is implemented by a mapping between the access protocol and the actual database that is being used for the linkbase server. Since A^3 is likely to define semantics

[6] For example, http://www.w3.org/2001/Annotea/User/Protocol.html describes the protocol used by W3C's Annotea for server/client communications.

that go beyond XLink's simple semantics, it would be a good idea to provide a reference implementation of a linkbase access server offering full A^3 support. This reference implementation should be based on a SQL database, because this is the most widely used type of database.

4.2 Client Side

While the server side only needs to implement the linkbase access protocol, the client side is harder to implement, because it has two interfaces, one to the server (the linkbase access protocol), and another to the user (the GUI through which the user access the A^3 functionality). In the following sections, both issues are discussed.

4.2.1 Client Logic

The client needs to implement the client side of the linkbase access protocol, and support the semantics of XLink (such as links pointing to linkbases) and A^3. The client must be configurable so that user interactions can be forced to be either local (for example, when adding an annotation, it will be embedded in the currently open document) or remote (user interactions are mapped to linkbase access, for example by creating a new annotation on the linkbase server). For workflow scenarios, the client must be sufficiently equipped to support user logon, user roles, and required user actions (for example, before a document can be forwarded to the next role in the workflow pipeline, all annotations of a certain kind must have been marked as finished).

4.2.2 GUI

Annotation GUI design has two major facets. One is the display of the actual annotation. PDF's current model uses a separate window pane for this, but XLink has a more general model, for example defining a way to transclude content, which means dynamic inclusion for presentation purposes. This model makes presentation of links a non-trivial problem, and so far there is no agreed-upon presentation model for XLink; there only is a W3C note about link presentation issues [18]. Publications by Bouvin et al. [2], Weinreich [19], and Oberholzer and Wilde [16] have investigated various ways to present XLink in their most general form, but so far no generally accepted model has emerged. It is important for A^3 to define a presentation model (which may disallow some forms of presentation for specific formats such as PDF, for example embedding).

Apart from the presentation of the annotation itself, additional GUI concepts are required to make other features of A^3 accessible, such as access rights, annotation editing, and linkbase access parameters. In particular the latter may become rather complicated, because it should be possible to execute any linkbase query (such as "give me all the annotations that have been created before last Friday from all users having the roles proofreader or copy editor"). Inspired by P3P's APPEL [7], I think that it would be a good idea to create a separate format for server access preferences, which could then travel with the user from client to client. Furthermore, server access preferences could be created using a nicely designed Web interface, and could then be downloaded in this server access preferences format and installed in the client. Designing a good GUI for accessing the complex functionality of annotations in a distributed and workflow-oriented scenario certainly will be a challenging exercise is usability engineering.

4.3 Phases and Costs

Looking at the A^3 architecture as a whole, it certainly is a challenging task. However, it is rather easy to split it into separate pieces, making the development and thus the costs more manageable. A natural split would be to separate the link model from the linkbase access protocol, and to simply assume that

in the first phase, annotations will still be embedded, or can only be imported or exported as files. In this case, GUI design would be the center of interest, as well as the relationship with the underlying document model. A large fraction of the costs would have to be dedicated to conceptual work, making sure that the first phase of A^3 will not contain anything that will make evolution difficult. Another large fraction of costs will go into GUI design and implementation, and this will require a major redesign of Acrobat's current interface (which is based on many assumptions which will no longer be true in the world of A^3 PDF).

The second phase of A^3 is based on the distribution scenario and the extension of Acrobat as a linkbase access protocol client. It should be considered to take over the leadership of linkbase access protocol development at W3C, which would make it possible to, on the one hand, create early implementations, and, on the other hand, shape the future of the linkbase access protocol in a way which is fully compatible with A^3. Costs in this phase would go into linkbase access protocol conceptual work, creating a reference server side implementation, adding client side functionality to Acrobat, and GUI design and implementation for linkbase access and workflow control.

A third phase of A^3 could start to extend A^3 beyond PDF and introduce A^3 support in other applications, such as FrameMaker. By using open standards, other companies could also be enabled to go this way, making integration of other products with PDF easier, and thus making Acrobat the product of choice for distributed and workflow-controlled collaboration.

5 Concluding Remarks

This is just the first version of a position paper for the A^3 architecture, and I am sure that it contains numerous errors and omissions. It should be sufficient to show the potential of advanced annotation features in PDF, and the approach to base them on open standards, so that they can be used across multiple applications.

The scenario of the book publication process is not ideal, because it involves many loosely coupled parties (authors, publisher, reviewers, proofreaders, copy editors). In a more tightly coupled scenario, where the roles, tools and workflows are more integrated, the potential benefits of the concepts described in this position paper become more significant. I believe that the benefits of advanced annotations are directly dependent on a well-defined workflow (with appropriate support in the linkbase), and a well-defined set of applications (or rather, applications with a well-defined and powerful set of import/export features).

This document is not intended to be technically accurate or to give a complete technical description of the features discussed in the scenario. Even though advanced annotations (most often referred to as *Open Hypermedia*) have been investigated in hypermedia research for some time now, integration into a production environment requires careful considerations of robustness and compatibility, which in the case of PDF (and maybe FrameMaker's XML) are beyond the scope of the position paper.

References

[1] Scott Boag, Donald D. Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML Query Language. World Wide Web Consortium, Recommendation REC-xquery-20070123, January 2007.

[2] Niels Olof Bouvin, Polle T. Zellweger, Kaj Grønbæk, and Jock D. Mackinlay. Fluid Annotations Through Open Hypermedia: Using and Extending Emerging Web Standards. In *Proceedings of the Eleventh International World Wide Web Conference*, pages 160–171, Honolulu, Hawaii, May 2002. ACM Press.

[3] Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in XML. World Wide Web Consortium, Recommendation REC-xml-names-19990114, January 1999.

[4] Tim Bray, Jean Paoli, C. Michael Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0 (Second Edition). World Wide Web Consortium, Recommendation REC-xml-20001006, October 2000.

[5] James Clark. XSL Transformations (XSLT) Version 1.0. World Wide Web Consortium, Recommendation REC-xslt-19991116, November 1999.

[6] John Cowan and Richard Tobin. XML Information Set. World Wide Web Consortium, Recommendation REC-xml-infoset-20011024, October 2001.

[7] Lorrie Cranor, Marc Langheinrich, and Massimo Marchiori. A P3P Preference Exchange Language 1.0 (APPEL 1.0). World Wide Web Consortium, Working Draft WD-P3P-preferences-20020415, April 2002.

[8] Ron Daniel. Harvesting RDF Statements from XLinks. World Wide Web Consortium, Note NOTE-xlink2rdf-20000929, September 2000.

[9] Steven J. DeRose, Eve Maler, and David Orchard. XML Linking Language (XLink) Version 1.0. World Wide Web Consortium, Recommendation REC-xlink-20010627, June 2001.

[10] Paul Grosso, Eve Maler, Jonathan Marsh, and Norman Walsh. XPointer Framework. World Wide Web Consortium, Recommendation REC-xptr-framework-20030325, March 2003.

[11] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. World Wide Web Consortium, Recommendation REC-rdf-syntax-19990222, February 1999.

[12] Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Thomas Nicol, Jonathan Robie, Mike Champion, and Steven Byrne. Document Object Model (DOM) Level 3 Core Specification. World Wide Web Consortium, Recommendation REC-DOM-Level-3-Core-20040407, April 2004.

[13] Jonathan Marsh. XML Base. World Wide Web Consortium, Recommendation REC-xmlbase-20010627, June 2001.

[14] Nilo Mitra. SOAP Version 1.2 Part 0: Primer. World Wide Web Consortium, Recommendation REC-soap12-part0-20030624, June 2003.

[15] Makoto Murata, Simon St. Laurent, and Dan Kohn. XML Media Types. Internet proposed standard RFC 3023, January 2001.

[16] Glenn Oberholzer and Erik Wilde. Extended Link Visualization with DHTML: The Web as an Open Hypermedia System. Technical Report TIK Report No. 125, Computer Engineering and Networks Laboratory, ETH Zürich, Zürich, Switzerland, January 2002.

[17] Christian Stillhard. Methods for Accessing Linkbases. Master's thesis, Computer Engineering and Networks Laboratory, ETH Zürich, Zürich, Switzerland, March 2002.

[18] Norman Walsh. XML Linking and Style. World Wide Web Consortium, Note NOTE-xml-link-style-20010605, June 2001.

[19] Harald Weinreich, Hartmut Obendorf, and Winfried Lamersdorf. The Look of the Link — Concepts for the User Interface of Extended Hyperlinks. In *Proceedings of the 12th ACM Conference on Hypertext and Hypermedia*, pages 19–28, Århus, Denmark, August 2001. ACM Press.

[20] Erik Wilde and David Lowe. *XPath, XLink, XPointer, and XML: A Practical Guide to Web Hyperlinking and Transclusion.* Addison Wesley, Reading, Massachusetts, July 2002.

[21] Erik Wilde and Christian Stillhard. Openly Accessing Linkbases. Technical Report TIK Report No. 134, Computer Engineering and Networks Laboratory, ETH Zürich, Zürich, Switzerland, January 2002.

[22] Erik Wilde. Linkbase Access Protocol Design. In *Poster Proceedings of the Eleventh International World Wide Web Conference*, Honolulu, Hawaii, May 2002.

[23] Erik Wilde. Protocol Considerations for Web Linkbase Access. Technical Report TIK Report No. 143, Computer Engineering and Networks Laboratory, ETH Zürich, Zürich, Switzerland, July 2002.