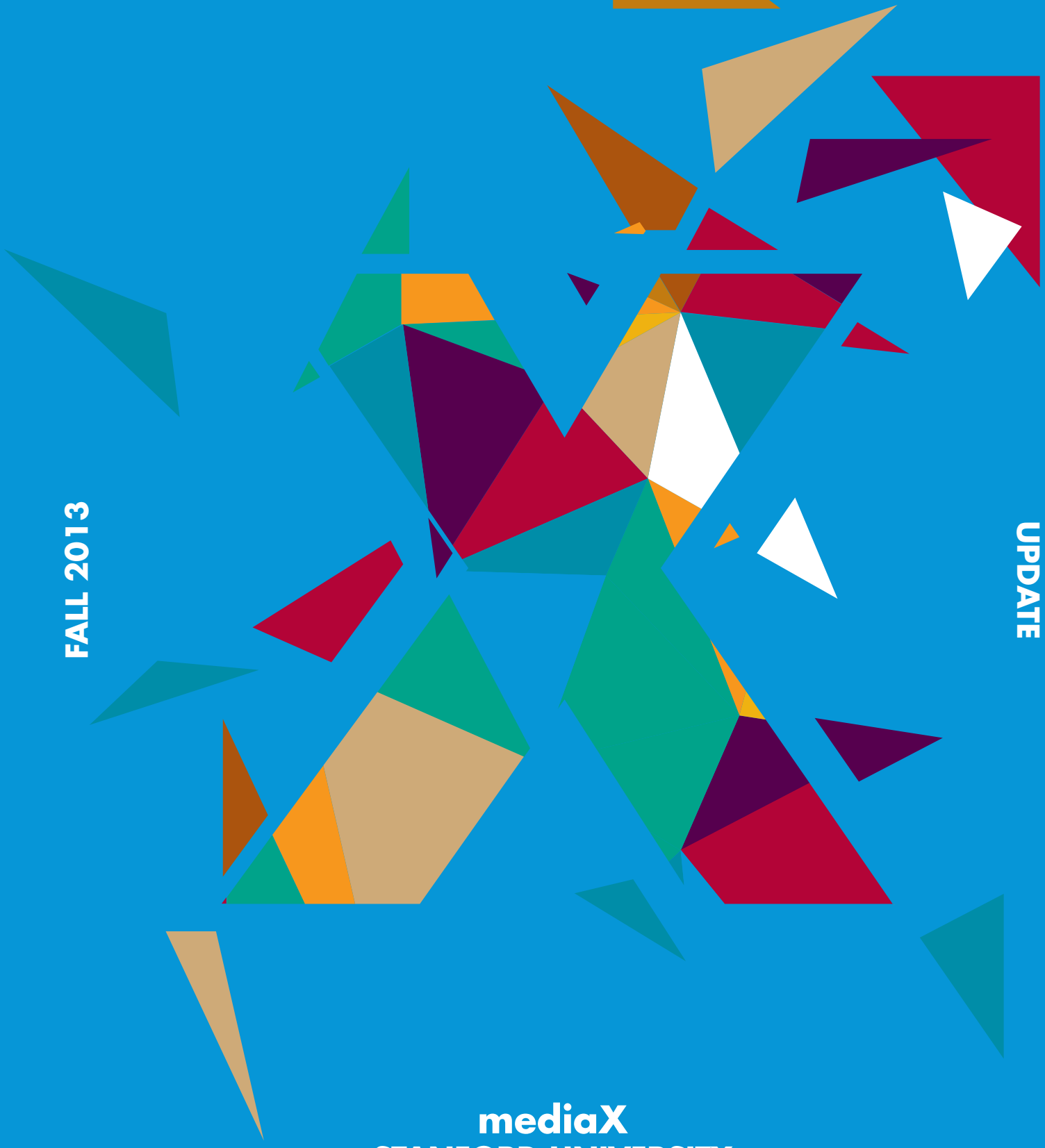
PUBLISH ON DEMAND

Smarter Scholarly Texts



FALL 2013

UPDATE

mediaX
STANFORD UNIVERSITY

mediaX connects businesses with Stanford University's world-renowned faculty to study new ways for people and technology to intersect.

We are the industry-affiliate program to Stanford's H-STAR Institute. We help our members explore how the thoughtful use of technology can impact a range of fields, from entertainment to learning to commerce. Together, we're researching innovative ways for people to collaborate, communicate and interact with the information, products, and industries of tomorrow.

media at Stanford University

Smarter Scholarly Texts: Phase One Completion Report

Research Team: John Willinsky, Khosla Professor of Education; Alex Garnett, Data Curator, Simon Fraser University Library; Juan Pablo Alperin, Researcher, Stanford Graduate School of Education

This is a progress report on the Smarter Scholarly Texts MediaX project, which received mediaX funding from Autumn 2012 through Spring 2013. Our goal was to develop a functionally complete, scholarly article parsing Toolchain and make it available as a web service. We sought to use open-source technologies wherever possible to facilitate rapid, industry-standard development and to integrate our service with Public Knowledge Project platforms, such as Open Journal Systems.

As of mid-June 2013, we have largely accomplished our preliminary goals. In the first month we hired a Vancouver-based developer who worked closely with Alex Garnett, the project lead. We formed a support agreement with the Manchester, UK-based developers of a toolkit called 'pdfx,' which formed the basis of our parsing engine. With these two elements in place, we were able to work quickly, adding additional layers of parsing functionality (e.g., accepting various common file formats, most notably Microsoft Word, and letting users select any known citation style from an open-source database not maintained by us to automatically format their submissions). We sourced translation expertise, free, from the Public Knowledge Project user community, permitting non-English-language submissions. We were also able to provide metered, subscription-based access to our webservice, letting us anticipate future scalability and cost-recovery plans. In all, we successfully completed a functional webservice (with scriptable API for production use; Fig. 1), and released an open-source plugin for Open Journal Systems that integrates our parsing functionality into existing scholarly journal workflows (Fig. 2).

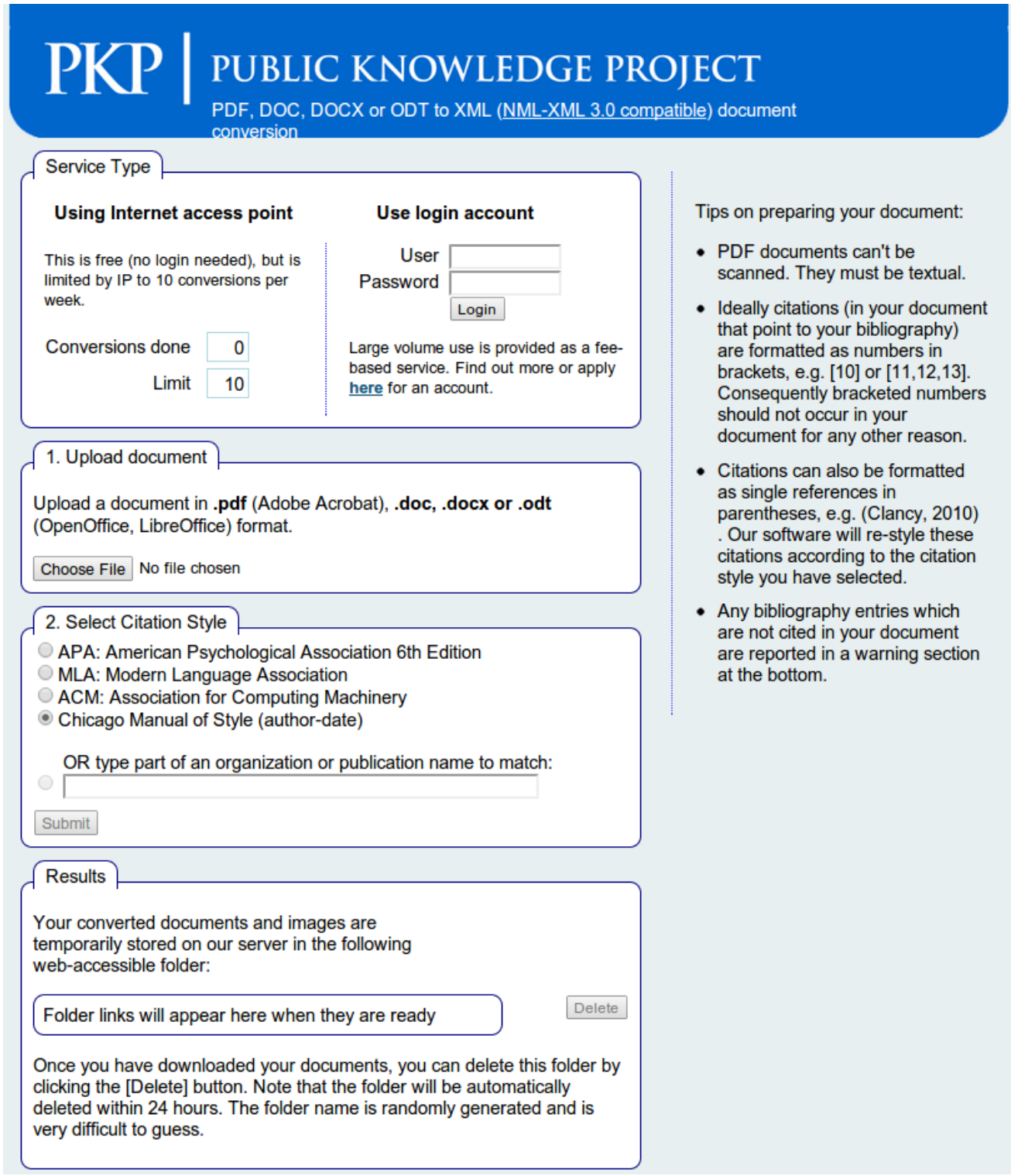


Figure 1: Parsing service standalone web frontend. Allows up to ten conversions by guest users, and infinite use by registered users (paying clients, development partners, or open-access publisher. Citation style selector allows finding-as-you-type.

Our over-all motivation: academic article production, in technological competence or resource sustainability, is deficient. Authors are presumed to write using any means that suits them (often the ubiquitous but little-loved Microsoft Word), and submit finished product to a professional editor, who must then transform the author's likely idiosyncratic use of headers and other textual elements into a production-quality document – ideally, and fairly frequently, the National Library of Medicine's XML, used in PubMed Central (Fig. 5). Because input to this process generally varies too widely for publishers' high standards, transforming an author's work into an archival or journal copy is almost entirely manual and is for this reason outsourced by most large publishers to one of a few production shops in the developing world. They do good work, but the failure to automate more of this process reflects the fact that human labour is cheaper for most developed-world publishers than technical progress might be, and those technical solutions that are implemented are virtually always proprietary for competitive reasons. This system fails publishers in the developing world, as well as academic-library-as-publisher scenarios, common to the Public Knowledge Project (PKP)'s open-source platforms Open Journal Systems and Open Monograph Press, respectively. PKP platforms have for a decade provided a scalable, sustainable approach to scholarly workflows, but have remained agnostic vs. text contents, relying on semi-professionalized journal editors to manually facilitate production.

In the same decade, open-source solutions for the component parts of article parsing – citation formatting, author disambiguation and conversion between office-document formats, to name a few – have made significant progress. Bits and pieces of these can be seen in successful open-source projects intended for academic audiences, such as Mendeley. Nobody, however, has yet assembled them for the purposes of article production.

Realizing that one of the primary project goals is to facilitate and improve editors' work, we have implemented copyediting-like functionality to improve text value. References cited in the article body but absent in the bibliography (or vice versa) are flagged for review; references that sync properly are automatically turned into hyperlinks that query Google Scholar for the source material, letting readers easily navigate through texts. We plan to supplement this functionality by letting our system automatically seek references, so that the hyperlinks point not to queries but to definitive (DOI-looked up, using technology from the industry-standard indexing services Sherpa-ROMEO and CrossRef) open- access versions of cited articles. We anticipate being able to verify quotations against cited open- access texts, flagging mis-parses the same way that missing references are currently flagged.

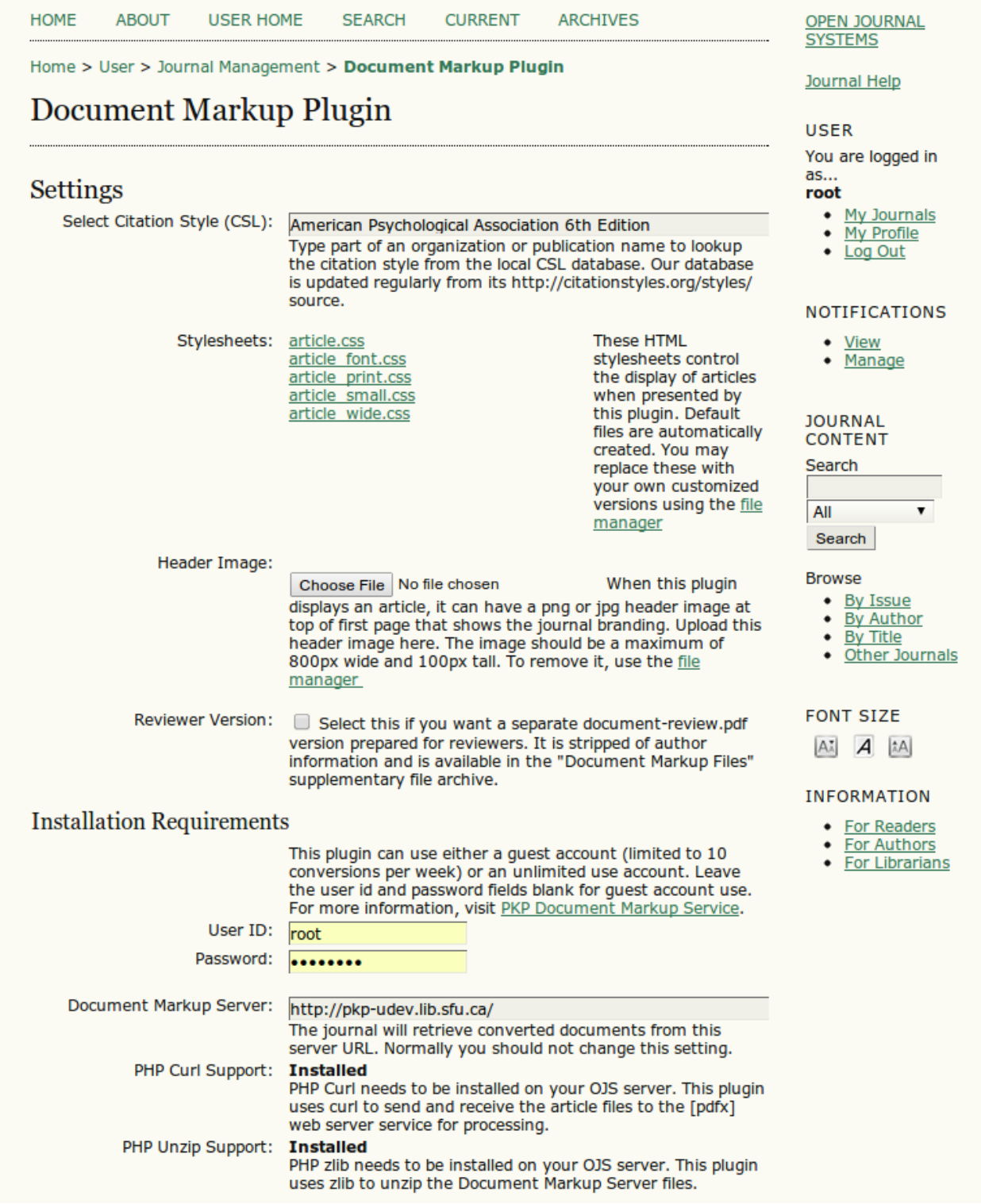


Figure 2. OJS Plugin configuration screen. Contains the same find-as-you-type citation style dialog as standalone web service. Allows users to replace CSS and automatically generates an anonymous document for blind review.

Some text elements require significant additional support (for example, mathematical formulae) that we would like to provide. While we have achieved conformity to the National Library of Medicine XML standard for archiving documents and transforming them into user-desirable formats like PDF and HTML (Figs. 3+4), we do not yet use any additional semantic markup layers – for example, structuring the relationship between individuals and experiments detailed in the text and exposing this structured data for text mining. Exciting work is being done in this area and we would like to both ensure compatibility with ongoing efforts and contribute our own.

Our parsing service is accessible from http://pkp-udev.lib.sfu.ca; the code will remain closed while we assess our licensing interests and negotiate the use of pdfx. The Open Journal Systems plugin is available at https://github.com/ddooley/ojs-markup, and is planned for inclusion in the next release of the software. We continue to use Github and a bug tracker to manage our small team and are prepared for the project to grow through additional funding and new personnel. We have three main development priorities:

1) Improve parsing quality. This will be an ongoing goal, as there are an infinite number of 'edge cases' when dealing with article layouts (and the lack thereof). We have effectively outsourced many individual components of the parsing to other open-source projects (exception for pdfx, which remains closed-source). We would like to contribute to these projects, in the interests of the community, as well as improving our own implementation.

2) Improve process usability. This includes contracting design work in discussion with our community, ensuring that we have attractive, functional layouts for various scholarly genres (for example, biomedical articles have different expectations than humanities articles). This also includes gracefully handling articles that do not parse entirely correctly – providing an elegant interface for authors and editors to correct minor errors without falling into the trap of menial XML editing, which we engineered this project to avoid. Our partners at the University of Heidelberg will help us undertake this work.

3) Advance support for Public Knowledge Project platforms. Our Open Journal Systems (OJS) 2.4 plugin is now functionally complete and will serve as a model for plugin development for both Open Monograph Press (OMP) and OJS 3.x. Work remains to adapt our production pipeline to monograph-length works. This will continue to grow our user community,

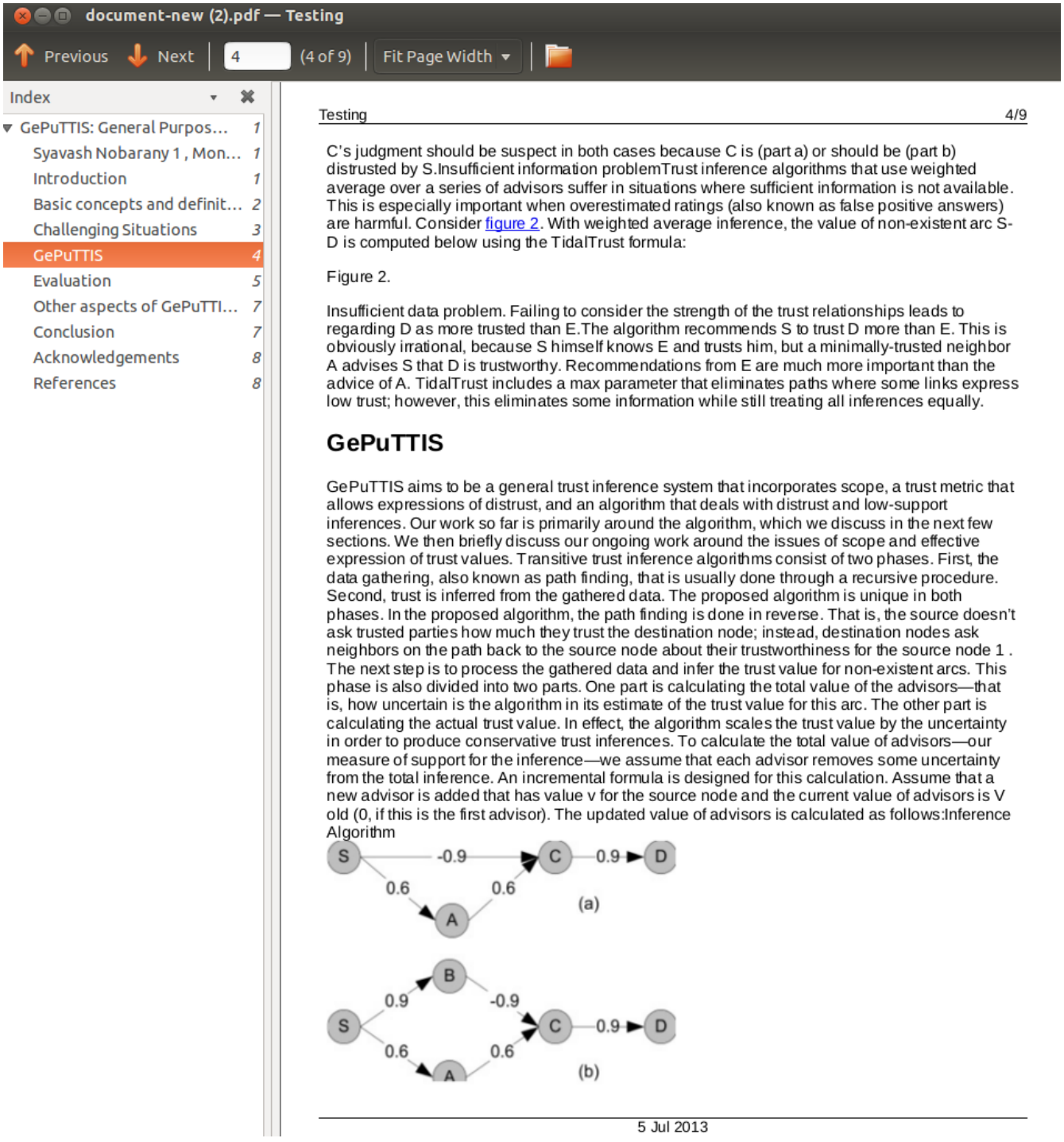


Figure 3. Example PDF output. Embeds table of contents (shown) and PDF metadata in article for ease of navigation. Currently viewed in Ubuntu; functions identically in OSX Preview or Adobe Reader on Windows.

and add wanted functionality to Public Knowledge Project platforms new and old.

Our relationship with the Public Knowledge Project has provided us bridge funding to continue work and retain personnel through the summer months. This time will be spent reaching necessary conclusions on issues of intellectual property, continuing to work on parsing issues and planning for future sustainability. Over all, we are very satisfied that we've built a completely functional service using the allotted funding, and eager to develop a community of users and stakeholders.

Figure 4. Example HTML output of same document. References automatically have links generated to Google Scholar queries for ease of navigation. We are currently using off-the-shelf open-source CSS and plan to redevelop our own.

```
▼<article xmlns:mml="http://www.w3.org/1998/Math/MathML" xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 ▼<front>
  ▼<article-meta>
   ▼<title-group>
    ▼<article-title>
       GePuTTIS: General Purpose Transitive Trust Inference System for Social Networks
     </article-title>
    </title-group>
   ▼<pub-date pub-type="epub">
     <year/>
     <month/>
     <day/>
    </pub-date>
    <article-id pub-id-type="doi"/>
  </article-meta>
  ▼<journal-meta>
    <journal-id/>
   ▼<journal-title-group>
     <journal-title>Journal of Problems</journal-title>
    </journal-title-group>
    <issn/>
   ▼<publisher>
     <publisher-name/>
    </publisher>
  </journal-meta>
 </front>
▼<body>
 ▼<sec>
    <title>Syavash Nobarany 1 , Mona Haraty 1 , Dan Cosley</title>
    1 School of ECE, University of Tehran, Tehran 14395-515, Iran 2 HCI Lab, Cornell University, Ithaca, NY 14850, USA
    {nobarany, haraty}@ gmail.com, drc44@cornell.eduAbstract Recent work has explored the idea of using trust networks to
    supplement ratings information in community-based information systems, including algorithms to infer missing values in
    the trust network. Current trust inference algorithms sometimes make undesirable inferences because they do not fully use
    information about distrust and sometimes make inferences based on weak support. Further, many algorithms do not consider
    the problem of trust scope, where one may trust someone's opinions about movies but not books. We present GePuTTIS, a
    trust inference system that reasons about support levels, distrust, and trust scope. We demonstrate that it improves
    prediction performance in a collaborative filtering dataset.
  </sec>
 ▼<sec>
    <title>Introduction</title>
    Web-based social networks have attracted millions of people; the growth of these communities has created new social
    outlets but weakened modes of community (Putnam, 2000). This gives computer scientists the opportunity and responsibility
    to design effective models of social living that bring value to these communities and their members. One of the most
    valuable concepts in people's real world interactions and relationships is trust. System designers use trust-inspired
    algorithms in applications such as ad-hoc networking and electronic commerce where entities must interact in highly
    dynamic and unpredictable environments in which traditional security measures are not effective. For example, mapping
```

Figure 5. Sample NLM-XML output. Archival quality; serves as the basis for transformation into HTML or PDF.

Appendix A

Software Used in Smarter Scholarly Texts

Software	**License**	**Purpose**
pdfx	Closed-source, by agreement	Generates structured XML from article PDF.
ParsCit	Open-source	Parses plain-text, variously-formatted references into usable author-title-source structure.
CSL	Open-source	Allows "reflowing" of parsed references into any desired style.
Pandoc	Open-source	Links inline references to bibliography and performs CSL transformations.
LibreOffice	Open-source	Converts .doc/.docx/.odt submission to PDF for parsing. Strips problematic formatting.
JATS Validator	Open-source	Ensures that the XML produced conforms to NLM standard.
phantomjs	Open-source	Produces PDF from production-quality HTML.
ExifTool	Open-source	Fills PDF metadata.
Angry Frog	Open-source	PHP login/user-management script

Appendix B

The Smarter Scholarly Text XML Issue Tracker: Future Developments

Category	Issue	Description	Solution Notes
Parsing	XML meta content validation	XML structure is already validated with https://github.com/PeerJ/validator, but content isn't checked -- should we do additional checks on PDFX output to see whether names are actually names, etc.?	Concerned that this involves developing on top of PDFX as though we're not relying upon them to get these things right in the first place. Would want to see their approach before trying to approve it.
Parsing	Use additional XML elements	We currently use just about everything that PDFX outputs (with the exception of equations, and errant text snippets that it indicates low confidence in), but this is a small subset of JATS. Can try to do our own parsing on top of PDFX output (both syntactic and semantic tags, though JATS is mostly the former) to add addt'l descriptive layers.	JATS tags are here: http://jats.nlm.nih.gov/archiving/tag-library/1.0/. I'm less concerned about this running opposite to PDFX development as this doesn't try to correct their output per se, it just does things they don't currently do. That said, I'm not sure how far we should go down the value-add-layer rabbit hole until we have requests from users.
Parsing	Add syntax layers (RDF, named entity recognition)	In short: do other things that aren't expressed by JATS. Adds more overhead and probably necessitates the development of reading tools to e.g. toggle these layers on and off while viewing the document inline, but would be sophisticated and forward-thinking.	Need good examples of people actually using these markup layers before we go to the trouble of implementing them. I tried to hack a named entity recognition layer together using Python's NLTK and DBpedia, but it needs better justification.
Parsing	Careful with special characters	We know PDFX is dropping a handful of special characters here and there -- bullet points are the worst offender -- and PDFX knows too. This needs fixing (and again, I'm not sure we can do a better job post-hoc than they can), but it's also one of the implict problem of image-style parsing.	This is probably the most prominent example of "we're not comfortable with it as it is, but we almost certainly can't do it any better than they can."
Parsing	Allow non-Latin text blocks	Though we're not planning on supporting anything more than left-to-right UTF-8 for the article body, we have a user who occasionally works with documents that contain East Asian language text chunks. We need to be able to, at minimum, preserve these 1:1 within the document itself.	The most naive but reliable way of doing this would obviously be to bitmap these text chunks, turn them into images, and insert them inline where the text way. But hopefully we can do better than this.

Parsing	Work on formula text	PDFX outputs formula text as <disp-formula class="DoCO:FormulaBox"> <content class="DoCO:Formula" id="56" page="3" column="1">t SD = t SA t × SA t AD = 0 . 1 0 × . 1 0 . 9 = 0 . 9</content> </disp-formula> Note that they're uniquely identified as formulae (good), but most useful structure is dropped (bad, particularly for math, which we really need to represent 1:1 as it is in the source).	Seems the smartest way to handle math formulae would be to use the MathTypeSDK on a word .docx upload to turn Word's embedded OMathXML into MathML (http://bit.ly/10KXOzR), which OJS has the ability to render. This would only work with .docx submissions and possibly .odt -- formulae are too sensitive to try to parse unstructured out of a PDF, and .doc doesn't contain OMathXML. We'd have to be able to figure out a way to then replace the formula strings in the pdfx output with our neatly parsed MathML, but probably doable. However, many authors who are apt to write formulae in research publications are also apt to use LaTeX rather than word, and .tex support would introduce a huge amount of other dependencies and fairly specific solutions.
Layout	Get professional CSS work done	We need, at minimum, a biomedical (PeerJ/PMC/E-Life derived) and a humanities layout, primarily for HTML but which will also work well for our PDF generation mechanism.	Designer contacted and requirements outlined; waiting for funding.
Layout	Be wary of page breaks	PDFX currently inserts XML elements to indicate where page breaks are in the original text; we ignore these because a) we're already pushing input through LibreOffice and presumably throwing the pagebreaks off before the document even gets to pdfx, and b) we go from pdfx to HTML, where page breaks don't necessarily matter. However, we do then turn the HTML to PDF, and our PDF creation device doesn't do a good job on page breaks currently (images can get cut in half).	I think we're probably happy enough with our PDF generation solution at this point to do the requisite hacking against it to try to anticipate how many lines/px tall a given page will output as, and probably employ some CSS hacks to make it work. Should try to involve designer in this process as best we can.
Layout	Be wary of paragraph breaks	Paragraphs are often too long and not especially readable in the current output. This is due at least in part to the current stylesheet, but we may need to try to somehow automagically break up paragraphs better.	Getting CSS work probably comes first.

Citations	Send ParsCit results to lookup service	ParsCit isn't perfect; should optionally query output strings against Google Scholar/PubMed and replace parsed result with any matches (over a certain threshold), on the assumption that they will contain more complete data.	We already generate links to search Google Scholar for any bibliographic entry. Of course, to actually extract data from Scholar, you need to screen-scrape, which gets gnarly. We'd also have to design our own system for assigning confidence ratings to results as we wouldn't want to sub in the fetched data for our parser output unless there's only one "right answer." Not trivial, but editors really like getting citations correct. May be able to leverage partnership with CrossRef instead of Google, could delegate a significant chunk of this.
Citations	Verify quotations	Assuming we can look up the full content of cited articles in a free database, fetch this text and verify direct quotations.	Dependent on many other components; nicely complicated. Not going to happen anytime soon but a very nice example of auto-copyediting.
XML editing	Outline WYSIWYG requirements	Heidelberg folks appear to need usability guidance; we need to make sure WYSIWYG actually works for non-professional XML editors	Already linked them to https://github.com/Annotum/textorum (though mostly on word of mouth; haven't really explore it deeply ourselves yet). Need to work pretty closely on this one.
XML editing	Design feedback loop	Design WYSIWYG cleanup step in such a way that article can be re-parsed after having been manually corrected, therefore minimizing the amount of manual correction that needs to be done. Somehow design cues for the XML editor to follow.	This is gonna be really tough for a number of reasons, not least of which is that PDFX doesn't really execute fast enough as things stand now to be able to provide helpful iterative feedback while manually editing. But exciting.
Mutilingual	Contribute more parsing dictionaries to PDFX	PDFX needs document section headers outlined in dictionary files to work effectively; we started out only having these in English, but have been working with OJS translators to prepare them for other languages.	We currently have Spanish and Portuguese done, with German coming. I'd like to get French too, and after that we can probably rely on users to approach us to do translation on their own.

Multilingual	Ensure ParsCit works properly on multilingual data	ParsCit seems to have trouble working on multilingual documents, so citations don't get parsed correctly.	Verified that ParsCit's ML engine has actually been trained properly on multilingual corpora, so the problem seems to be that it's not properly identifying the bibliography section within document. This may be an issue with how we've wrapped it but we're having trouble fixing.
Output	Support embedding supplemental data iFrames in HTML output	Requested by Dataverse / CoAction partners; example: http://figshare.com/blog/Embeddable_figshare_content/87	This isn't about parsing per se and would have to be an intermediated process, but is germane to what we're doing, as our project currently represents the majority of work being done on OJS HTML article production/viewing
Output	Generate BibTex citation file for article itself	We currently generate a BibTex file of the article's bibliography, but it doesn't include metadata for the article that's actually being parsed.	Easier to do from OJS/OMP plugin, as metadata is often missing when using standalone service (similar situation to PDF XMP, which is already implemented).
Output	Handle monograph output differences	Ensure current parsing engine is adaptable from articles to monograph-length works	Relatively straightforward as far as NLM is concerned -- there's a <sub-article> tag in which individual monograph chapters can be nested. Waiting on ePub support from mPach. Should check ONIX metadata requirements against what's already in OMP.
OJS	Release OJS 2.4.x plugin	Completed code review and basically ready to go; Alec suggested we make a video walking through the functionality.	Would like to include in main OJS distribution after next point release
OJS	Develop OJS 3.x plugin	Self-explanatory	No hurry until we have users and OJS 3.x is approaching release
OMP	Develop OMP plugin	Self-explanatory	Committed to this for Heidelberg; good opportunity to compare+contrast OJS plugin workflow

Additional Reading:

Statement of the Publish On Demand Research Theme
http://mediax.stanford.edu/POD/concept

For more information:

- membership
- research themes
- events (conferences, seminars, workshops etc.)

Please visit our website - *http://mediax.stanford.edu*

Like us on Facebook -
https://www.facebook.com/MediaXatStanford
Follow us on Twitter -
https://twitter.com/mediaXStanford
Join us on LinkedIn -
http://www.linkedin.com (search for MediaX at Stanford)
Watch us on YouTube -
http://www.youtube.com/user/mediaxstanford

or contact:

Martha Russell, Executive Director - marthar@stanford.edu

Jason Wilmot, Communications Manager - jwilmot@stanford.edu

Adelaide Dawes, Program Manager - adelaide@stanford.edu

Acknowledgements

Many thanks to mediaX Members and Partners, whose contributions have supported this Research.

Special thanks to the professors and researchers who provided these research updates. Thanks also to Addy Dawes, Program Manager, and Jason Wilmot, mediaX Communications, for editing and putting this report into the format you are reading.