RefUTU–Automatic Bibliography Database Generation for Freely Formatted Reference Listings

Johannes Holvitie, Ville Leppänen

Abstract: *This paper presents the RefUTU framework for automatic bibliography database generation for freely formatted reference listings. Its three stages of reference extraction, partitioning, and formation are explained, implemented, and trialed on test data. The results showcase detection accuracy and discuss the possibility of using pre-existing solutions independently at each stage. It is concluded that the framework provides an improvement over the laborious and error-prone work of manual reference handling often required for example in the conduction of systematic literature reviews.*

Keywords*: software frameworks, information sciences, computational bibliometrics*

INTRODUCTION

Modern research practice holds searches conducted in electronic publisher and indexer databases as a necessary step for surveying previous research. For a majority of fields, databases have become the primary and even sole way of accessing information [7]. The results returned by these databases include electronic copies of articles, article reference extractions (for articles with more restricted access), or single reference entries (in the case of a fully restricted or non-electronic article). Often, the enquirer wants to store the—hundreds or thousands of—returned references, for example as part of familiarizing a new research area, a systematic literature review, or a mapping study.

Knowledge received from data is proportional to the level of structure present in it [3]. Applying this for references: in order to be informative, there needs to exist structural information that describes what parts of the reference capture what fields—e.g. all letters up until first semi-colon or number capture author names. This raises a problem as most encountered references—be it search engine results or articles themselves—carry no meta-information about the references' structure. More precisely, the meta-information is not extractable, as the references do follow one of many reference formats that impose a structure on to the entries but this structure is implicitly present in the formatting.

To extract this structure one may opt to do the conversion from a formatted reference to a structured database entry by hand. While manual reference extraction allows for any type of reference format, it requires plenty of mechanical repetition which is tedious and hence error-prone. That is why, especially for larger conversion undertakings, one requires an automated approach. Two components are distinguished for an automated approach that aims at forming a bibliography database from a freely formatted reference listing: 1) the approach must analyse and understand the used reference format in order to impose structure onto the data and 2) the approach must normalize the structured data (e.g. over different accuracies) so that it may be collated to form a database. There are solutions available that solve the issue for single components. The problem is that their interoperability, which is scarce, is required to achieve the desired result.

Hence, as a unique solution, this article proposes the *RefUTU* framework for automatic bibliography database generation for freely formatted reference listings. As a component based framework, a separate solution can be used to solve each sub-problem. This allows

CompSysTech'15, June 25-26, 2015, Dublin, Ireland

 ISBN 978-1-4503-3357-3/15/06...$15.00
http://dx.doi.org/10.1145/2812428.2812469

generating combinations that support desired input and output formats while retaining high detection accuracy.

RELATED WORK

Regarding similar frameworks, Tkaczyk et al. [18] introduce a tool named *CERMINE* which is capable of automatically extracting both metadata and references from "born-digital" scientific literature. They apply k-means clustering and Conditional Random Fields (CRF) to extract and partition references. They reach very high recall for extracted-field-correctness. The lack of normalization however makes the method inapplicable for managing material covering multiple different reference formats. Further, Bergmark introduces *D-Lib* [2] which implements reference extraction and linking from online documents. She provides a detailed discussion on the related problems and overcoming them. Here, it is noted that references must be presented in HTML or plain text format in order to be analysed. Again, the *D-Lib* achieves high recall, but is limited to singular reference formats.

Regarding tools, *Zotero* [8] and *Mendeley* [13] are amongst the largest electronic article managers. While both of them are able to extract meta-data from an input PDF (Portable Document Format) and querying *Google Scholar* [9] with this information, they do not provide support for analysing the PDF reference lists. *EndNote* [17] is the only tool capable of this, but only for references with DOIs (Document Object Identifiers). *inSPIRE*'s *Reference Extractor* [5] extends this to references that either have *arXiv* identifiers or follow certain report numbers or journal references, and is hence format constrained. Finally,*cb2bib* [6] inputs plain text reference lists (supports their user assisted extraction from PDFs) and parses them according to a reference format regular expression library. While *cb2bib* is not format constrained, it requires pre-defining each format by hand.

Regarding research adopted herein, Kern & Kampfl [10] discuss loss of information when using only plain text for reference extraction and they suggest a layout based method. This approach is implemented in *CrossRef*'s *PDFExtract* [1, 16] which is capable of extracting reference lists from PDFs. The previously mentioned Conditional Random Fields (CRF) method is implemented in the *FreeCite* [4] system by Brown University and it is used together with *PDFExtract* herein. CRFs have been introduced in [12], applied to the PDF extraction problem in [15] and to the context of Reference Partitioning in [14].

FRAMEWORK DESCRIPTION

This section describes the *RefUTU* framework for automatic bibliography database generation for freely formatted reference listings. The framework is depicted in Figure 1. It can be considered as a pipeline formed from an optional pre-processing step and two components. A subsection is dedicated for describing each step and linking it to the pipeline.

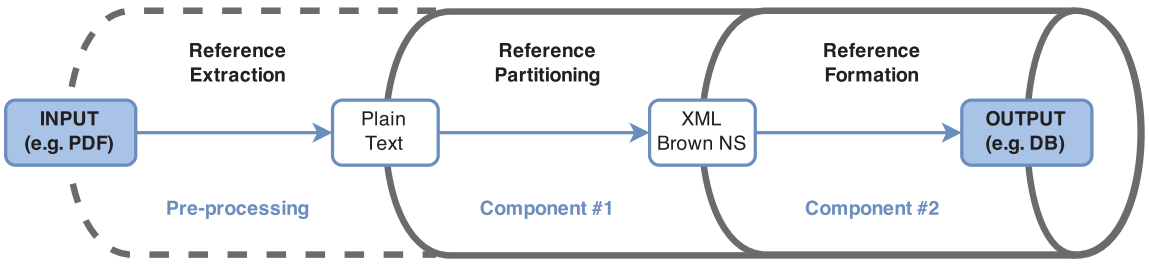

Figure 1: The *RefUTU* framework

Reference Extraction

Reference extraction is required when the references are not in plain text format, which is the only requirement for the framework's input. Commonly, the references are part of a larger set of data, such as an article. Possibly saved in a non-transparent format, like an encoded page content stream in a PDF file. In these cases, reference extraction is responsible for overcoming the transparency issue by 1) extracting the data in plain text format, and overcoming the data obfuscation issue by 2) extracting only the references from the data.

As the references can exist in a variety of input formats, it is quite possible that a number of reference extraction components will be co-used. Here, however, the user is responsible for data integrity, meaning that the framework is not capable of distinguishing and filtering non-reference data prior to executing Reference Partitioning and formation.

Reference Partitioning

The Reference Partitioning component forms the first half of the framework. It is responsible for imposing structure on to the input data. In practice, the plain text reference data is parsed using the chosen tool(s) and the expected output, for each reference, is the reference's data incorporated with meta-data that describes the reference's structure—i.e. fields like author, title, and year and their position. We assume the structural description to be partially correct. That is, received partitioning (from one or multiple methods) must correctly capture at least some of the reference's fields. The Reference Formation component takes the partial correctness into account.

The output from the Reference Partitioning is a structured list of references where the structure tries to describe for each reference what fields are present in it and what are their contents. Since the Reference Formation component is dependent on the output's structure, it must be fixed. We chose XML (Extensible Markup Language) as the structure retaining medium and we fixed the structure's component specifications by choosing an XML name space defined by the Brown University for describing references [4].

Following a specific name space is beneficial for component interconnectivity. First, if we want to use a Reference Partitioning component that is incapable of producing XML in the Brown name space, we can provide an adapter for it by implementing a translator that converts the component's output to the desired XML. Similarly, as the name space describes all fields possible to be present in the XML, we can use any Reference Formation component incapable of accepting this XML as long as we can provide an adapter for it.

Reference Formation

The Reference Formation component forms the second half of the framework. It is responsible for taking in structured reference data and normalizing it with respect to being entered into the bibliographical database. This normalization means that the component finds, for each structured reference, a uniform representation.

Entry into the database requires that the component is capable of producing an identifier that is unique with respect to the uniqueness of the publication behind the reference. To achieve this, the component exercises a bibliography database specific search strategy. The strategy consists from multiple search definitions, each describing a level of search accuracy for that database. Each definition tells which search options are active and which combinations of fields from the references' structured XML representations will be used in the query. The search is started from the most strict, and hence most accurate, definition and the strategy will iterate over all specified definitions in a descending order of accuracy until a match is found. The first match is used for producing the uniform bibliographical entry.

If we recall that the Reference Partitioning component is not expected to fully understand the reference format and hence to not be able to produce a perfect XML partitioning,

we understand why a search strategy defining a descending order of search accuracy is required. In the best case scenario, the XML partitions contain correct data for each of the reference's fields and the most strict search strategy yields a match for it—given that the queried database contains an entry for this reference. In the worst case scenario, the XML partitioning has managed to extract only one field from the reference correctly. Now, the search strategy must be executed until only the correct field is considered alone and a match is found for it. In the latter case, this the only way to produce an entry for the reference that can be considered field-wise as uniform and complete as the best case scenario.

The unique identifiers and the special case of a search strategy failing to produce a single match for a reference are considered. In general, the unique identifier for a reference is formed as a combination of the fields in the reference. The framework follows the *Google Scholar* BIB-identifier schema [9]: `[1st author surname]<year><1st title word>`. The identifier "`brown2010managing`" in Figure 4 is an example of an identifier constructed this way. In cases where the search strategy fails the fallback plan for this reference is to build the identifier from the fields present in the XML.

Having executed the search strategy or the fallback strategy for each reference, the Reference Formation component has generated an entry for each reference. Disregarding possible fallbacks, these entries are uniform in terms of their identifiers and fields present and they are ready to be entered into the database. While the database is separate from the framework, for purposes of retaining from storing duplicate information—i.e. references with identical identifiers—and for example generating cross-reference graphs, relational databases are considered a viable option.

CASE-STUDY

This section documents a case-study with the *RefUTU* framework. We describe the concretization of the framework via an example conversion. At each stage of the framework, we describe the pre-existing solution fitted for the purpose and its function on the example input. Finally, we discuss results when the attained framework is applied to a large data set.

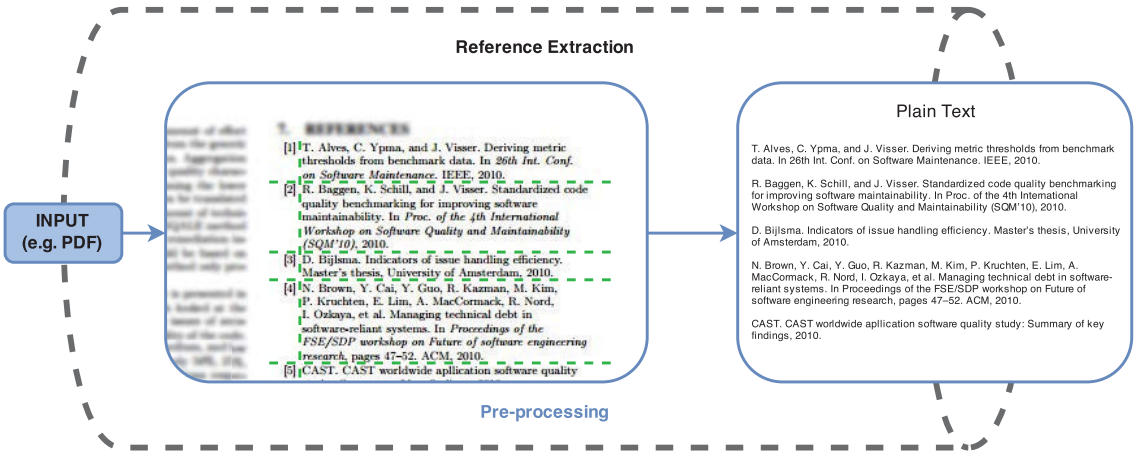

Figure 2: The Reference Extraction pre-processing stage

Reference Extraction

The example input for our framework is a scientific article that is distributed electronically as a PDF file. At the end of this scientific article is a reference list—that follows the *IEEE Transactions* reference format—which we are interested in obtaining. In order to apply the framework for it, it must be converted to plain text format. This requires that we execute the pre-processing step of Reference Extraction on it.

There are a number of tools intended for extracting elements from a PDF file. These tools function differently. Some mine the raw text data in order to distinguish partitions from it, while others analyze the layout of the file in order to reach the same result. As discussed, the latter is a viable approach when trying to distinguish reference lists from PDFs and the *PDFExtract* [16] supports this.

Figure 2 showcases the functionality of the pre-processing stage. Here, a tool like *PDFExtract* is applied to the PDF in order to distinguish and extract a list of references from the paper. The resulting output for this stage is expected to be a plain text listing, where each line represents a single reference. Note, that the references still follow their own formatting. Only the context specific reference data—like paper-specific reference list numbering—should be excluded.

For removing context specific reference data, a library of regular expressions can be used. For example, a risk free (in terms of not matching fields within the reference) regular expression to match *IEEE*-formatted reference list numbers is "`^\[\d+\]\s+`" (matches a number in brackets if it is the first characters after a new line and followed by a white space character). A library of these expressions can be accumulated and ran on each reference as a pre-processing phase as long as their match-safety is ensured (see *cb2bib* [6]).

Reference Partitioning

After attaining the references in plain text format, they can be input into the framework. The first half of the framework is the Reference Partitioning component which is responsible for analysing the reference, identifying fields from it, and delivering this information as XML output which follows the Brown University naming scheme.

Again, a number of tools exist which try to identify fields from singular references. As the framework must provide support for all plain text entries, the discussed Brown University *FreeCite* parser [4] is used. *FreeCite* uses the *CRF++* (Conditional Random Fields) [12] machine learning library and comes pre-trained with a number of reference formats. Due to this approach, *FreeCite* can be trained and modified to learn and hence to better partition new reference formats on the fly.

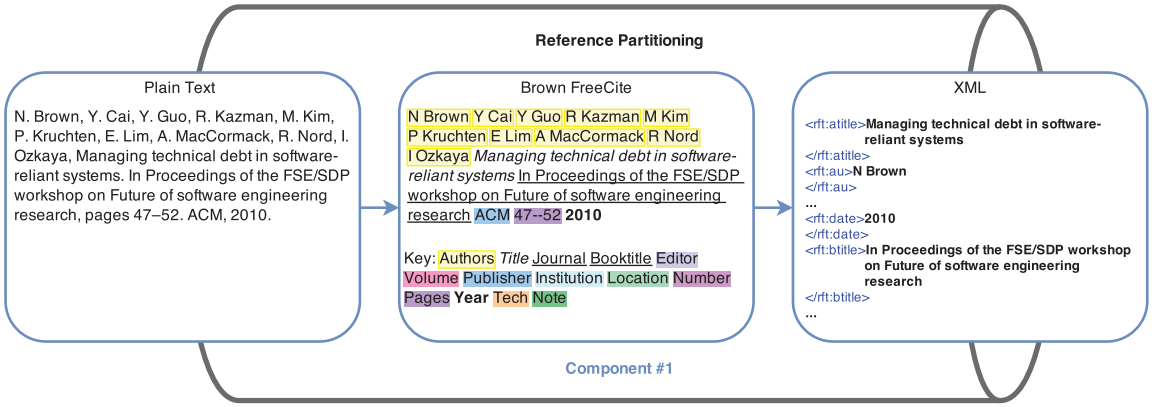

Figure 3: The Reference Partitioning component

From Figure 3 we see that the library is quite successful in distinguishing and partitioning fields from the input reference. As discussed, some errors could still be present at this stage—e.g. the title field could partially contain the name of the conference. Hence, we can not expect a perfect identification and we must be ready to work with partially erroneous data. To accommodate this, we implement the search strategy in the next section.

Reference Formation

The second half of the framework is implemented by the Reference Formation component which is responsible for the normalization of reference data so as to form bibliographical database entries. The normalization is concerned with field accuracy and field uniformity. As we expect the XML partitioning to possibly be erroneous, the field accuracy pursues completing the incorrect or missing fields based on the correct ones. Field uniformity is interested in retaining a shared filling pattern throughout all fields in all references. This ensures that for example if two references are made to the same article but with differing formats the imposed uniformity will reveal the target article to be the same for both references.

Use of the Reference Formation component makes the framework a non-stand-alone application as we make queries to external databases. There is a wide variety of reference databases, some are implemented by publishing bodies to organize access to their own material, while some are implemented by third parties to collate and index them. *Google Scholar* [9] is an example of the latter and due to its multi-disciplinary nature we wanted to access it in the Reference Formation component. Direct access to these databases is often not offered as most of the orchestrators fund their operations via offering library services. There however exists a third party one for the *Google Scholar* in the form of an open-source project called *scholar.py* [11]. Hence, the Reference Formation component becomes one that executes the search strategy via this project.

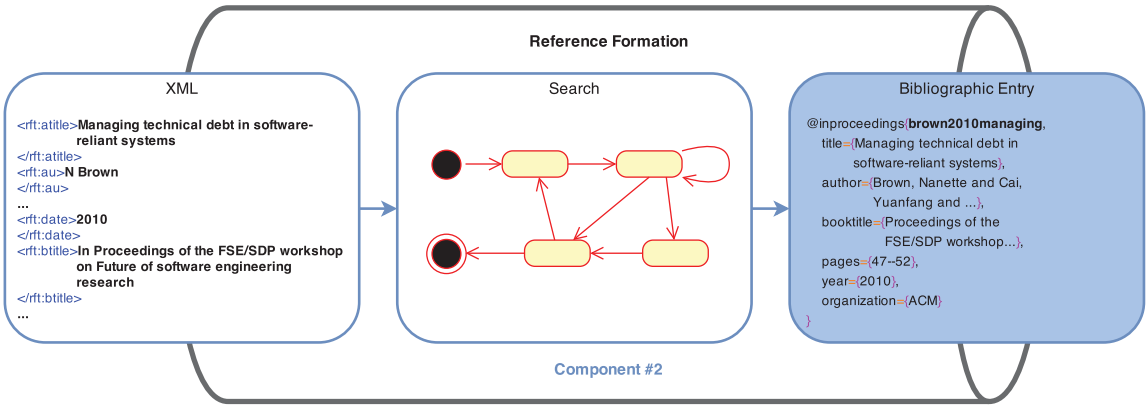

Figure 4: The Reference Formation component

Figure 4 depicts the conversion from the partially accurate XML partitioning into a complete and correct bibliographical entry. The entry can then be served to a database provider or in this case output directly into a file in BibTeX format. The conversion is the end result of the component executing the search strategy. Figure 1 depicts the search strategy composed for querying the *Google Scholar* database, while Figure 5 captures the state machine representing the query execution logic. In the search strategy example, the fields "title", "author" and "year" are fetched from the input XML structure. In the Brown University name space, these correspond to the values present in the reference's XML components `<rft:atitle>`, `<rft:au>`, and `<rft:date>`. Functionality of the state machine is straightforward: after receiving the XML input, it starts executing the search strategy. If no match is received for the query, the next most accurate query is used. In case of a match, a database entry is prepared from the match's fields. In case of no matches for the entire strategy, a fallback plan must be used, in which case a database entry is constructed directly from the XML fields.

Results

We applied the aforedescribed framework to a set of references discovered as part of systematic literature review. In this review, we started with an initial set of 97 papers from

Accuracy (ACC)	Query (title, year, author)
1	--phrase "title" --after "year" --author "author"
2	--phrase "title" --after "year"
3	--title-only --all "title" --after "year" --author "author"
4	--title-only --all "title" --after "year"
5	--title-only --some ...
...	...

Table 1: *Google Scholar* search strategy

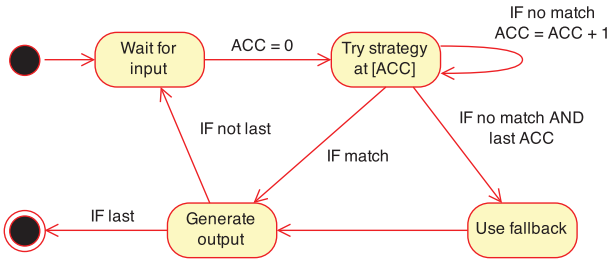

Figure 5: The *RefUTU* Query Logic

which we wanted to extract references for further inspection. We had access to PDF versions of all the initial papers and hence we could input all of them directly into the Reference Extraction stage. After the pre-processing step, we were left with 1590 raw references.

For the Reference Partitioning component, a worst case performance can be defined as the output XML indicating only one field for a reference—comprising the full entry. This is possible for example in the case of referencing URLs (Universal Resource Locator) with no accompanying information. In these problematic cases, all identified fields are cycled as title-fields, so that the search strategy is still applicable to them.

The Reference Formation component ran on the 1590 input XML structures, each describing a reference. While the XML transformation took 22 seconds to complete, this component took 12 minutes and 15 seconds. This is purely because we needed to query *Google Scholar* whilst keeping their anti-spam measures in mind. In practice, this meant waiting a couple of seconds between each query and in cases where further queries where refused, wait a longer period of time before retrying. From each query, the first match was taken, as per decreasing accuracy expected from the search strategy. Table 2 documents the results.

Table 2: Identification results

Input (Pre-processing)	No. of papers (PDF)	97
	Formats encountered	~25
Output (Comp. #1: XML)	References found	1590
	Time taken	~22sec
Output (Comp. #2: Scholar)	References matches	996 (Recall 0.626)
	Time taken	~12min 15sec
	Incorrect matches	594
	non-existing	157
	identification failures	437 (Recall 0.725)

Reviewing Table 2 we see that the initial recall is 63% which leaves 594 papers for manual reprocessing. Closer look at thenon-identified papers reveals that 157 of them are "non-existing" references. That is, they are references for which we can not expect to find a match from most databases—e.g. the previously described URLs. This makes the actual recall for this particular implementation of the framework 73%. Considering that [2] reported 83% for the non-querying and format-constrained framework, this a very promising first result.

CONCLUSIONS AND FUTURE WORK

This paper introduced the *RefUTU* framework for automatic bibliography database generation for freely formatted reference listings. It consists of an optional pre-processing phase called Reference Extraction and two components: Reference Partitioning and Reference Formation. Reference Extraction is responsible for taking the original source for the references and extracting them in a plain text format. Reference Partitioning takes in the plain text references and tries to identify and partition fields for them. This information is delivered in an XML format to the final component of Reference Formation. Here, a database-specific search strategy is executed for the identified fields in order to complete and normalize the

references so that they may be entered into a bibliographical database.

The framework allows different solutions to be fitted for each stage so that desired input and output formats as well as search engines and accuracies can be supported. As there is a multitude of excellent solutions to overcome each of these phases, we introduce this framework to enable their interoperation to produce higher quality results whilst decreasing the amount of manual, mechanical, and error-prone labor.

We foresee a number of improvements for the framework. Firstly, an improved user interface is under construction. Also, an on-memory database will be included for easy manipulation of results—especially for cross-referencing incorrect matches. Further, we intend to define adapter interfaces for each stage so that integration of new solutions becomes easier but also to allow multiple solutions to be co-executed at each stage. Finally, we invite the reader to follow our progress, to contribute, and most importantly to use our tool by visiting the *RefUTU* homepage at `http://soft.utu.fi/refutu/`.

REFERENCES

[1] Ø R Berg, S Oepen, and J Read. Towards high-quality text stream extraction from pdf. In *Proceedings of the ACL-2012 Special Workshop on Rediscovering*, volume 50, pages 98–103, 2012.

[2] D Bergmark. Automatic extraction of reference linking information from onlinedocuments. Technical report, Cornell University, 2000.

[3] M R Berthold, C Borgelt, F Höppner, and F Klawonn. *Guide to intelligent data analysis: how to intelligently make sense of real data*, volume 42. Springer Science & Business Media, 2010.

[4] Brown University and Public Display. Freecite citation parser. *URL: http://freecite.library.brown.edu/*, 2015.

[5] CERN, DESY, Fermilab, and SLAC. inSPIRE. *URL: https://inspirehep.net/textmining/*, 2015.

[6] P Constans. cb2bib Reference Extractor. *URL: http://www.molspaces.com/d_cb2bib-overview.php*, 2015.

[7] C Creaser, Y Hamblin, and Eric D J. An assessment of potential efficiency gains through online content use. *Program*, 40(2):178–189, 2006.

[8] Roy Rosenzweig Center for History and New Media. Zotero reference manager. *URL: https://www.zotero.org/*, 2015.

[9] Google Inc. Google scholar - scholarly literature database. *URL: https://scholar.google.com/*, 2015.

[10] R Kern and S Kampfl. Extraction of references using layout and formatting information from scientific articles. *D-Lib Magazine*, 19(9):2, 2013.

[11] C Kreibich. scholar.py, google scholar parser. *URL: https://github.com/ckreibich/scholar.py*, 2015.

[12] J Lafferty, A McCallum, and F CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[13] Mendeley Ltd. Mendeley reference manager. *URL: https://www.mendeley.com/*, 2015.

[14] M Ohta, D Arauchi, A Takasu, and J Adachi. Crf-based bibliography extraction from reference strings focusing on various token granularities. In *Document Analysis Systems (DAS), 2012 10th IAPR International Workshop on*, pages 276–281. IEEE, 2012.

[15] F Peng and A McCallum. Information extraction from research papers using conditional random fields. *Information processing & management*, 42(4):963–979, 2006.

[16] Inc. (PILA) Publishers International Linking Association. Crossref labs pdfextract. *URL: http://labs.crossref.org/pdfextract/*, 2015.

[17] Thomson Reuters. Endnote x7, reference manager. *URL: http://endnote.com/*, 2015.

[18] D Tkaczyk, P Szostek, P J Dendek, M Fedoryszak, and L Bolikowski. Cermine–automatic extraction of metadata and references from scientific literature. In *Document Analysis Systems (DAS), 2014 11th IAPR International Workshop on*, pages 217–221. IEEE, 2014.

ABOUT THE AUTHOR

Johannes Holvitie and Ville Leppänen, TUCS - Turku Centre for Computer Science, Software Development Laboratory (SwDev) and Department of Information Technology, University of Turku, {jjholv, ville.leppanen}@utu.fi

ACKNOWLEDGEMENTS

J. Holvitie is supported by the Nokia Foundation Scholarship and the Finnish Foundation for Technology Promotion, the Ulla Tuominen Foundation, and the Finnish Science Foundation for Economics and Technology grants.