Transforming large collections of scientific publications to XML

H. Stamerjohanns, M. Kohlhase, D. Ginev, C. David and B. R. Miller

Abstract. We describe an experiment transforming large collections of LaTeX documents to more machine-understandable representations. Concretely, we are translating the collection of scientific publications of the Cornell e-Print Archive (arχiv) using `LaTeXML`, a LaTeX to XML converter currently under development.

While the long-term goal is a large body of scientific documents available for semantic analysis, search indexing and other experimentation, the immediate goals are tools for creating such corpora. The first task of our arXMLiv project is to develop `LaTeXML` bindings for the (thousands of) LaTeX classes and packages used in the arχiv collection, as well as methods for coping with the eccentricities that TeX encourages. We have created a distributed build system that runs `LaTeXML` over the collection, in part or entirely, while collecting statistics about missing bindings and macros, and other errors. This guides debugging and development efforts, leading to iterative improvements in both the tools and the quality of the converted corpus. The build system thus serves as both a production conversion engine and software test harness.

We have now processed the complete arχiv collection through 2006 consisting of more than 400,000 documents (a complete run is a processor-year-size undertaking), continuously improving our success rate. We are now able to convert more than 90% of these documents to XHTML+MathML. We consider over 60% to be successes, converted with no or minor warnings. While the remaining 30% can also be converted, their quality is doubtful, due to unsupported macros or conversion errors.

1. Introduction

The last few years have seen the emergence of various XML-based, content-oriented markup languages for mathematics and natural sciences on the web, e.g. OpenMath [BCC+04], Content MathML [ABC+09], or our own OMDoc [Koh06]. The promise of these content-oriented approaches is that various tasks involved

in "doing mathematics" (e.g. search, navigation, cross-referencing, quality control, user-adaptive presentation, proving, simulation) can be machine-supported, and thus the working mathematician can concentrate in doing what humans can still do infinitely better than machines.

On the other hand, LaTeX is and has been the preferred document source format for thousands of scientists who publish results that include mathematical formulas. Millions of scientific articles have been written and published using this document format. Unfortunately, the LaTeX language mixes content and presentation and is extensible through macro definitions — a mixed blessing in this context. Therefore machines have great difficulties processing LaTeX documents to extract enough information to represent the written formulas in a XML representation.

In this paper, we will present an experiment translating a large corpus of mathematical knowledge to a form more suitable for machine processing. The sheer size of the arχiv [ArX07] poses a totally new set of problems for Mathematical Knowledge Management (MKM) technologies, if we want to handle (and in the future manage) corpora of this size. In the next section we will review the translation technology we build on; the following sections present the corpus-level build system which is the main contribution of this paper.

2. TeX/LaTeX to XML Conversion

The need for translating LaTeX documents into other formats has been long realized and there are various tools that attempt this at different levels of sophistication; see [SGD+09]. We will disregard simple approaches like the venerable `latex2html` translator that can deal only with simple user macros, and produces only html with minimal semantic structure. The remaining ones fall into two categories that differ in the approach towards processing the TeX/LaTeX documents.

Romeo Anghelache's Hermes [Ang07] and Eitan Gurari's TeX4HT [TeX] systems leverage the TeX engine for dealing with the intricacies of the TeX macro language. They use special TeX macros to seed the `dvi` file generated by TeX with semantic information. This `dvi` file is then parsed by a custom parser to recover the text and semantic traces which are then combined to form the output XML document. An advantage of these TeX-based systems is that they can directly use the standard (ie. TeX) implementations of any macro packages used by the documents, usually without errors. Any semantic intent of macros defined in those packages is lost, however, unless the macros are modified to add the seeding; this can be a tricky proposition potentially requiring modification to both the TeX macros and the `dvi` parsing. While Hermes attempts to recover as much of the mathematical formulae as Content-MathML, it has to revert to Presentation-MathML where it does not have semantic information. TeX4HT directly aims for Presentation-MathML.

In contrast, the `LaTeXML` [Mil07] system and the SGLR/Elan4 system [vdBS03] re-implement the TeX engine for a large fragment of the TeX

language. This has the distinct advantage of bringing the power of more conventional programming languages to bear on both the parsing process and XML generation. In particular, we want to expand abbreviative macros (i.e. convenience macros that just abbreviate token sequences) and recursively work on the resulting token sequence, while we want to directly translate semantic macros (i.e. macros that stand for higher-level concepts or objects), since they directly correspond to the content representations we want to obtain. See [Koh08] for a discussion of semantic and abbreviative macros; for instance $\mathcal{C}^\infty(\mathbb{R})$ could be marked up with the macros in Listing 1. The first two are semantic, whereas the last is abbreviative.

Listing 1. Semantic and Abbreviative Macros

```
\def\Reals{\mathbb{R}}
\def\SmoothFunctionsOn#1{\mathcal{C}^\infty(#1)}
\def\SmoothFunctionsOnReals{\SmoothFunctionsOn\Reals}
```

A disadvantage in this approach, however, is that these systems tend to *require* their own implementations of LaTeX packages (see Section 4). Even if they can successfully process the low-level TeX coding of packages, this usually generate constructs that pass below the semantic layer of markup.

For both categories, however, a challenge is dealing with the types of markup and usage patterns found 'in the wild'. TeX is, among other things, a complex macro processing engine; LaTeX itself is, in fact, implemented as a macro package on top of TeX. Moreover, all of TeX's capabilities are available to authors, allowing them to develop non-standard usages and to exploit quirks in the processing model — authors are notorious for finding peculiar ways to get a specific desired effect in print that completely obscures the semantic intent of the document. Even when the the processing engine is able to cope with the computations, the semantic intent can easily be lost, unless care is taken in the conversion.

In our conversion experiment we have chosen the `LaTeXML` system, whose LaTeX parser seems to have largest coverage in the second "semantics-aware" category. The `LaTeXML` system consists of a two programs: the TeX emulator `latexml`; and the post-processor `latexmlpost`. `latexml` processes the TeX document, emulating TeX's processing, loading the `LaTeXML` bindings for any LaTeX packages referred to, and generates an intermediate XML output in the LTXML format. The LTXML is based on the implicit document model used by LaTeX, intended to lose as little semantics and document structure as possible. For instance LTXML supplies `<theorem>` and `<proof>` elements as XML counterparts to the `theorem` and `proof` environments in LaTeX. Any mathematics within the document is parsed, to the extent possible, preserving any semantics or structure implied by the markup.

In the post-processing phase, this LaTeX-near representation is transformed into the target format by the `latexmlpost` program. This program applies a pipeline of intelligent filters to its input, performing a variety of tasks such as storing and filling-in information for cross-references and bibliographies, generation of images for mathematics (if needed) along with included graphics or picture

environments. Finally, the filters perform the necessary conversions to various formats including the main one of interest here: XHTML +Presentation-MathML.

Other filters such as transformations to OpenMath and Content-MathML are under development, although they rely on inferring more of the mathematical semantics than is currently done. The parsing of mathematics mentioned above uncovers the structure of the expressions, but not necessarily the meaning. It is obviously essential, but not sufficient, for the eventual generation of content-oriented mathematical formats, such as Content-MathMLand OpenMath. However, the mathematical structure is also essential to the generation of quality Presentation-MathML; for this reason failure to recognize the mathematical structure yields warnings and such documents are not considered fully successful conversions (see the categories in Figure 2). Further projects are underway to develop semantic filters to enhance the content, disambiguate notations, and carry out type and part-of-speech analysis [GJA+09].

3. The Build System

The main contribution of this article is the arXMLiv build system. In the experiment we report on we transform the articles in the arχiv collection from LaTeX to XHTML+MathML.

The huge number of more than 400,000 documents in this collection made simple manual handling of conversion runs impossible. In a first attempt, we tried to mechanize the conversion process itself (invocation of `latexml` and `latexmlpost`) by a set of script-generated `Makefiles`. But even using distributed extensions of `make` (e.g. `dmake`) or other grid tools that support distributed builds to run distributed jobs on several hosts — a feature that is essential to be able to massively convert thousands of documents in one day — proved infeasible as they did not give us the necessary control over the build process. It turned out that for determining the improvements on the `LaTeXML` system and the package bindings necessary for increasing translation coverate we need a automated analysis of the conversion results. In particular, we need to cluster specific error patterns and gather statistics of failure causes.

The arXMLiv build system (see figure 1) distributes `make` jobs among several hosts and also extracts and analyzes the conversion process of each document and stores results in its own SQL database. The usage of a database allows us to cluster documents which include a specific macro that is only partially supported or to gather statistics about specific errors in the document processing.

The arXMLiv build system consists of a shared file system, a queue manager, a build manager, and a relational database, which stores a work queue and results statistics about each single converted file. The file system contains all the documents ($\approx$ 150 Gigabytes), classified by topic and each one located in its own sub-directory. The file system is exported via NFS to all hosts which take part in the build process.

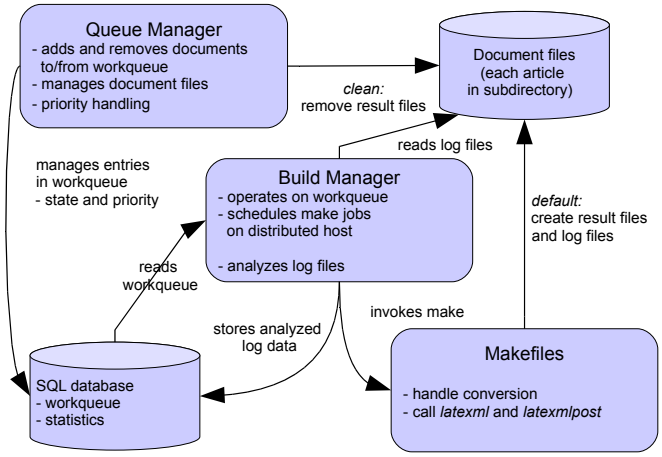

FIGURE 1. Schematic overview of the arXMLiv build system

The build manager is implemented in PHP, where SQL databases as well as process control functions can be easily used. It keeps an internal list of available 'worker' hosts, reads the files to be converted next from the workqueue and distributes jobs to remote hosts. For each document to be converted, the build manager forks a new child process on the local machine. The child sets a timer to enable a limiting timeout of 180 seconds for the conversion process and then forks a (grand-)child process to call `make` on a one of the worker hosts via remote `ssh` execution. The `make` process will then invoke `latexml` and `latexmlpost` to convert a TEX file to XML and then to XHTML, respectively.

After the completion of the conversion process, or if the time limit was exceeded, the build manager is notified via regular Unix signal handling. The build manager then parses and analyzes `LaTeXML`'s log files, extracts information about the resulting state of conversion and collects the names of package files not found and missing macros not yet supported. If the conversion has failed, the error messages given by `LaTeXML` are also extracted. For each processed document, the analyzed result data is then stored into the database for later use. With such stored result data it is possible to instruct the queue manager to rerun selected subsets of documents, such as those using a certain macro or to those which resulted in a fatal error in the conversion. A command like `php workqueue.php default cond-mat` will add all documents inside the `cond-mat` subdirectory — arχiv's *condensed matter* classification — to the current work queue. This has been especially useful when changes to the binding files have been applied or when an improved version of `latexml` becomes available.

Developers can retrieve the results via a web-interface available at `http://arxmliv.kwarc.info`. The main page gives an overview of the conversion process: it describes the number of documents, the number of converted files in the last

24 hours, the current state of the build system, version metadata and the hosts involved in the conversion. For our experiment we have used 24 processors on 13 different hosts.

A further table (see figure 2) gives detailed information about the results of the conversion process. The most important states are *success* (`warning` and `no_problems`) where latexml has only issued some minor warnings, *missing macros*, where the conversion has been successfully completed, but some macro definitions could not be resolved. In this case the rendered layout may contain unexpected elements or not properly displayed elements.

Detailed return values

return value	count	%	marked for rerun
unknown	399	n/a	
no_latex	25346	n/a	3
missing_errlog	25300	n/a	
fatal_error	19177	5.48	87
timeout	10673	3.05	
error	36683	10.48	29
missing_macros	71060	20.31	52
warning	165482	47.29	218
no_problems	46826	13.38	29

FIGURE 2. Conversion status

All these states are hyperlinks leading to a list of recently converted files with the specified status. In these lists, the file name is also a hyperlink leading to the source subdirectory for the document where all the document-related files can be investigated, such as the TEX source, the intermediate XML file that `LaTeXML` produces or the final XHTML+MATHML form. Also the full log file containing detailed error messages can be easily be retrieved via the web browser.

Top Fatal Errors

No.	Count	Error Message
1	10603	Too many errors!
2	1035	Unbalanced $ or } while ending group
3	1018	Missing $ closing display math.
4	976	Unbalanced $ or } while ending mode inline_math
5	820	The token T_PARAM[#] should never reach Stomach!
6	565	Unbalanced $ or } while ending mode text
7	471	Can't call method "currentColumn" on an undefined value at /soft/arXMLiv/dan/rep
8	426	Package
9	421	Input file appears to be binary.
10	411	Can't call method "currentColumnNumber" on an undefined value at /soft/arXMLiv/d

FIGURE 3. Top fatal errors

The back end behind the web interface is also able to analyze the database content and create detailed summaries on-the-fly, such as lists of *Top Fatal Errors* (see Figure 3) and *Top Missing Macros* (see Figure 5). These lists have proven to be particularly useful to the developer of the `LaTeXML` system and the implementers of the needed binding files to deal with some of the issues discussed in sections 2 and 4. One can easily determine the most severe bugs in the still evolving conversion tool, as well as isolate specific documents 'abusing' markup, or using peculiar TEX idioms.

Again, each entry in these tables is a hyperlink leading to a list of relevant documents. For example, Figure 6 shows the documents causing a specific error condition. Discovering and analyzing these usage patterns has led to the elimination of a large number of bugs and mistaken assumptions in `LaTeXML` (see Figure 4 for a time line).

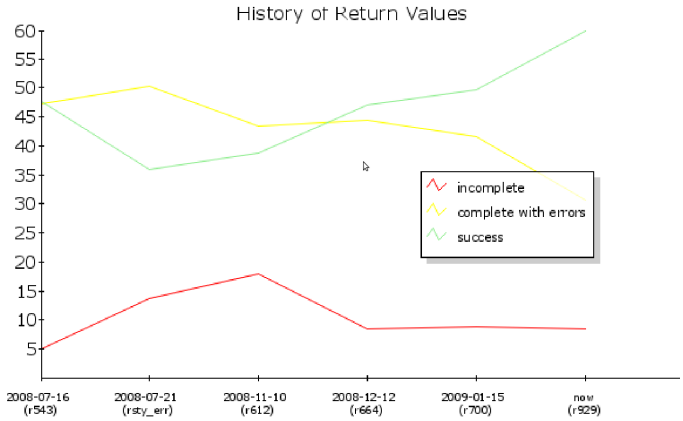

FIGURE 4. The history of return values in our conversion experiment

Occurences of missing macros

No.	Macro (and used in files...)	Count	Defined in
1	keywords	7687	defined in files...
2	affil	6224	defined in files...
3	email	4972	defined in files...
4	references	4643	defined in files...
5	acknowledgments	3923	defined in files...
6	Align	3778	defined in files...
7	reference	3640	defined in files...
8	institute	3475	defined in files...
9	apj	3155	defined in files...
10	address	3084	defined in files...
11	acknowledgements	3044	defined in files...

FIGURE 5. Top missing macros

The summaries also provides statistics on which macros and style files are heavily used so that development can focus on those which are not yet implemented but will have the most impact. Again, this has led to implementation of a number of interesting style and class files that are widely used, but were initially unfamiliar to the developers. The main remaining task in the arXMLiv project is now to enhance the coverage of package implementations. We describe the tools and workflows to implement these in the next section.

4. Creating `LaTeXML` binding files

As discussed above, we need to supply `LaTeXML` bindings, that is, a `LaTeXML`-specific implementation, of the various LaTeX style files used in the documents in the corpus to enhance coverage of the translation process. LaTeX packages tend to fall into one of two groups: those that merely adjust the formatting to fit, say, a particular journal style; and those that define additional macros, often with implied semantics. Moreover, in a collection as diverse as the arχiv, many authors have created personal 'variants' of more common packages, which they have adjusted for their own preferences.

The arXMLiv translation is supported by a working group of undergraduate students at Jacobs University who create `LaTeXML` bindings for the LaTeX packages used in the arχiv. To be able to support thousands of different style files the group has developed a set special of tools and workflows which we now describe.

Files that have given fatal error: *Unbalanced $ or } while ending mode inline_math*

No.	Date	Files	Errmsg
1	2009-05-18 20:37:14	/math-ph/papers /0612034	Unbalanced $ or } while ending mode inline_math
2	2009-05-18 20:37:06	/math-ph/papers /0612004	Unbalanced $ or } while ending mode inline_math
3	2009-05-18 20:36:42	/math-ph/papers /0512094	Unbalanced $ or } while ending mode inline_math
4	2009-05-18 20:36:39	/math-ph/papers /0512017	Unbalanced $ or } while ending mode inline_math
5	2009-05-18 20:36:25	/math-ph/papers /0504068	Unbalanced $ or } while ending mode inline_math
6	2009-05-18 20:36:18	/math-ph/papers /0507032	Unbalanced $ or } while ending mode inline_math
7	2009-05-18 20:36:13	/math-ph/papers /0505040	Unbalanced $ or } while ending mode inline_math
8	2009-05-18 20:35:48	/math-ph/papers /0410029	Unbalanced $ or } while ending mode inline_math
9	2009-05-18 20:35:47	/math-ph/papers /0412081	Unbalanced $ or } while ending mode inline_math
10	2009-05-18 20:35:45	/math-ph/papers /0501018	Unbalanced $ or } while ending mode inline_math

FIGURE 6. Documents with a specific fatal error

The build system already gives us information about which packages are used most extensively. To cope with package variants we have developed a package analyzer that creates a similarity matrix of all the packages files that are being used in the arχiv (see http://arxmliv.kwarc.info/sty_sim.php for the result). This allows the variant packages to be treated by simply loading the common base package:

```
RequirePackage("BaseStyleFileName");
```

The purely style oriented packages can be effectively ignored; we can formalize this simply by creating an empty binding file.

The remaining packages define additional macros and thus require additional attention. The binding files can define abbreviative macros using the `DefMacro` form supplied for this purpose by `LaTeXML`. For semantic macros we need to associate associate an XML fragment with the TeX markup using the `DefConstructor` form. For instance the macros from Listing 1 would have the following bindings:

```
DefConstructor('\Reals',"<ltx:XMTok name='Reals'/>");
DefConstructor('\SmoothFunctionsOn{}',
        "<ltx:XMApp><ltx:XMTok name='SmoothFunctionsOn'/>#1</ltx:XMApp>");
DefMacro('\SmoothFunctionsOnReals',"\SmoothFunctionsOn\Reals");
```

Note that the semantics of `\Reals` is preserved by encoding it as a `XMTok` element that is the LTXML version of a symbol (aka. logical symbol), while the semantics of `SmoothFunctionsOn` is marked up as the application of a function to its argument. `LaTeXML` offers a rich declarative infrastructure for implementing `LaTeXML` bindings; for more sophisticated constructs, the binding author can fall back to

the LaTeXML API in PERL. For example, the Babel package required a deep understanding of how characters are composed in the Unicode standard and to get to know most of the names for the different types of diacritical marks; the RevTeX package required understanding of what the original LaTeX macros did and trying to mimic that specific behavior.

The arXMLiv working group has created more than two thousand binding files to support packages that are being used by authors of the arχiv. Some sophisticated packages are not completely handled yet; for example usage of pstricks suite would ideally generate Scalable Vector Graphics (SVG), this is implemented only in-part; a fall-back to image generation is supported, however.

5. Conclusion and Outlook

Our arXMLiv build system has allowed us to apply the LaTeXML conversion tool across a large collection of scientific documents; we have been able to successfully convert more than half of the more than 400,000 LaTeX articles of the arχiv to a semantically enriched XHTML+MathML representation. The build system has enabled us to cope with the conversion process of this huge collection of documents while providing statistics and reports to guide improvement of the binding files needed to support various LaTeX style files, and also to improve the LaTeXML tool, itself.

We have been able to improve the success rate to close to 61% of the corpus being converted with significant fidelity. Another 30% have also been converted and are available as XHTML+MathML, although they suffer from unrecognized macros and other inaccuracies; we do not currently count them as full successes since the representation and layout may not be correct and the rendering in a web browser might not be fully appropriate.

This has expanded our collection of scientific MathML documents which we need for further studies in semantic analysis, search indexing and other experimentation by more than 240,000 (real-world) documents. Having access to the documents in XML, with all TeX-specific processing already handled, but with minimal loss of semantics, opens the door to many additional applications. We are currently working on linguistic/semantic analysis of the corpus [GJA+09] with the goal of enabling MKM services like our semantic search engine [KŞ06, Mat08]. But the XHTML +MathML conversion already has shown completely unexpected benefits: the MathML formulae in the transformed corpus can be read out e.g. by Design Science's MathPlayer system [Mat], thus opening it up to sight-impaired "readers".

With the high success rate — and with diminishing returns on further binding development — we turn our focus in two new directions: applying this experience to additional corpora, and enhancing the mathematical semantics of the document conversion. For the first, we are currently working to acquire additional corpora, e.g. Zentralblatt Math [ZBM07]. Preliminary tests show that due to the careful

editorial structure of this collection and the limited set of macros that need to be supported, our system can reach nearly perfect translation rates.

The next steps in the analysis of the arχiv corpus will be to improve the `LaTeXML` post-processing, and in particular the OpenMath/MathML generation. A much wider variety of mathematical notations is encountered in the arχivcorpus than was used in the original development of `LaTeXML`; data-mining and additional support from the build system should help expose these notations and will help drive improvements to the core mathematical grammar used by `LaTeXML`. Beyond that, further work in disambiguation is necessary to select which of several interpretations should be applied. Some of these improvements will be integrated directly into `LaTeXML`. Other improvements will require more involved approaches based on heuristics or linguistic and semantic analysis. Rather than relying on a single tool like the `latexmlpost` processor for this task we plan to open up the build system and compute farm to competing analysis tools.

The build system described here is open source software and can be obtained from the authors upon request. The conversion tool, `LaTeXML`, is also open source and available from [Mil07]. Additionally, we allow access to the build system through a web interface (available at `http://tex2xml.kwarc.info`); LaTeX files can be manually uploaded and converted, making use of the many additional binding files created to support the arXMLiv work (because of limited resources, this system only supports a few concurrent users).

References

[ABC+09] Ron Ausbrooks, Bert Bos, Olga Caprotti, David Carlisle, Giorgi Chavchanidze, Ananth Coorg, Stéphane Dalmas, Stan Devitt, Sam Dooley, Margaret Hinchcliffe, Patrick Ion, Michael Kohlhase, Azzeddine Lazrek, Dennis Leas, Paul Libbrecht, Manolis Mavrikis, Bruce Miller, Robert Miner, Murray Sargent, Kyle Siegrist, Neil Soiffer, Stephen Watt, and Mohamed Zergaoui. Mathematical Markup Language (MathML) version 3.0. W3C Working Draft of 4. June 2009, World Wide Web Consortium, 2009.

[Ang07] Romeo Anghelache. Hermes - a semantic xml+mathml+unicode e-publishing/self-archiving tool for latex authored scientific articles. web page at `http://hermes.roua.org/`, 2007.

[ArX07] `arXiv.org` e-Print archive, seen December2007. web page at `http://www.arxiv.org`.

[BCC+04] Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaetano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The Open Math Society, 2004.

[GJA+09] Deyan Ginev, Constantin Jucovschi, Stefan Anca, Mihai Grigore, Catalin David, and Michael Kohlhase. An architecture for linguistic and semantic analysis on the arXMLiv corpus. In *Applications of Semantic Technologies (AST) Workshop at Informatik 2009*, 2009. in press.

[Koh06] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer Verlag, 2006.

[Koh08] Michael Kohlhase. Using LaTeX as a semantic markup format. *Mathematics in Computer Science*, 2008.

[KŞ06] Michael Kohlhase and Ioan Şucan. A search engine for mathematical formulae. In Tetsuo Ida, Jacques Calmet, and Dongming Wang, editors, *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2006*, number 4120 in LNAI, pages 241–253. Springer Verlag, 2006.

[Mat] Mathplayer: Speech instructions and examples. web page at http://www.dessci.com/en/products/mathplayer/tech/accessibility.htm.

[Mat08] Math Web Search. http://kwarc.info/projects/mws/, seen Dec. 2008.

[Mil07] Bruce Miller. LaTeXML: A LaTeX to xml converter. Web Manual at http://dlmf.nist.gov/LaTeXML/, seen September2007.

[SGD+09] Heinrich Stamerjohanns, Deyan Ginev, Catalin David, Dimitar Misev, Vladimir Zamdzhiev, and Michael Kohlhase. Mathml-aware article conversion from LaTeX, a comparison study. In Petr Sojka, editor, *Towards Digital Mathematics Library, DML 2009 workshop*. Masaryk University, Brno, 2009.

[TeX] TeX4ht: LaTeX and TeX for hypertext. web page at http://www.tug.org/applications/tex4ht/mn.html.

[vdBS03] Mark van den Brand and Jürgen Stuber. Extracting mathematical semantics from latex documents. In *Proc. Intl. Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2003)*, number 2901 in LNCS, pages 160–173, Mumbai, India, 2003. Springer.

[ZBM07] Zentralblatt MATH, seen December2007. web page at http://www.zentralblatt-math.org.

H. Stamerjohanns
Computer Science, Jacobs University Bremen
e-mail: h.stamerjohanns@jacobs-university.de

M. Kohlhase
Computer Science, Jacobs University Bremen
e-mail: m.kohlhase@jacobs-university.de

D. Ginev
Computer Science, Jacobs University Bremen
e-mail: d.ginev@jacobs-university.de

C. David
Computer Science, Jacobs University Bremen
e-mail: c.david@jacobs-university.de

B. R. Miller
National Institute of Standards and Technology
e-mail: bruce.miller@nist.gov