Final Technical Report E.5.1
Methodology and Costs for Creation of Dual-Use Grammars: Lessons Learned

Date of report: January 26, 2009
Information current as of: January 26, 2009

Contact Information:
Anne David
301-226-8969
adavid@casl.umd.edu
www.casl.umd.edu

Authors:
Anne David, Mike Maxwell, Evelyn Browne, Jessica Shamoo, David Zajic, Nathanael Lynn,
Christopher Taylor, Pamela Kling

CDRL: A021
DID: DI-MISC 80508A
Contract No. H98230-07-D-0175
# Table of Contents

1. Introduction ...................................................................................................................................... 3
2. Descriptive and Formal Grammars .................................................................................................. 5
  2.1 Descriptive Grammar ................................................................................................................. 5
    2.1.1 Use of DocBook .................................................................................................................. 5
    2.1.2 Formatting the Grammars .................................................................................................. 6
    2.1.3 Example Sentences ............................................................................................................. 9
    2.1.4 Word Annotation ............................................................................................................... 10
    2.1.5 Morphological Complexities ............................................................................................. 13
    2.1.6 Variant Representations .................................................................................................... 14
    2.1.7 Fallible Resources ............................................................................................................. 15
    2.1.8 svn Server ......................................................................................................................... 15
  2.2 Formal Grammar .......................................................................................................................... 16
    2.2.1 Conversion to Parser .......................................................................................................... 17
    2.2.2 General Processing Issues .................................................................................................. 19
    2.2.3 Coordination ...................................................................................................................... 20
    2.2.4 Editorial Issues .................................................................................................................. 21
    2.2.5 Technical Issues ............................................................................................................... 21
    2.2.6 Implications ....................................................................................................................... 22
3. Software Tools ................................................................................................................................... 23
4. Costs ............................................................................................................................................... 25
  4.1 Salaries ......................................................................................................................................... 25
  4.2 Direct payments to language experts ......................................................................................... 25
  4.3 Relevance and Applicability of this Data .................................................................................... 25
1 Introduction

In creating the dual-use grammars of Bangla and Urdu, our goal has been not merely to produce grammatical resources for these languages, but to articulate a methodology for developing subsequent grammars, a methodology which could be used for other languages by researchers at CASL, or contracted out in part or whole. Presented here are our methodological findings, relating to the descriptive and formal grammars and to the process of coordinating their development. We also describe the ways in which our vision of grammar development changed as we created these grammars: the lessons we learned along the way.

As stated from the beginning of this project, our intent was to build two grammars in parallel: a descriptive grammar of the sort a linguist would use, and a formal grammar suitable for automatic conversion into the programming language of a parsing engine. The formal grammar was to be broken into pieces—“fragments”—at a suitable level, e.g. a single phonological rule, or a paradigm of affixes. These formal grammar fragments were to be embedded into the descriptive grammar at the points where they were being described in words, so that the verbal description elucidated the formal description, while the formal grammar disambiguated (where necessary) the descriptive grammar. The formal grammar could be automatically re-constituted, giving an application-independent but computer-readable form of the grammar.

At this level of detail, nothing has changed; but at a more detailed level, we have made a number of choices, as well as some changes. This report describes the choices and changes; decisions which were made early on and which worked out well are also documented. We also describe some remaining problems.

The report is organized into five sections: we first describe the methodology for creating the descriptive grammar, and then the formal grammar development.

We then turn to how the development of these two parallel but different grammars was coordinated among different authors. This points to one of the major outcomes that we did not anticipate when we began this project: while the formal grammar is dependent on the descriptive grammar (since the former encapsulates the grammatical information in the latter), and while the
descriptive grammar is likewise dependent on the formal grammar (because the process of writing the formal grammar inevitably turns up deficiencies in the descriptive grammar, which must be addressed), the two grammars are independent in the sense that writing them requires substantially different skills—skills which are seldom found in a single person. While the authors of the Bangla and Urdu descriptive and formal grammars worked down the hall from each other, we have come to realize that this is not necessary: we could have been half a world apart, a fact whose implications we draw out in that section.

The fourth section of the report lists our software tools and their sources.

Finally, the last section of this report describes how we spent our time in this project, including both the development of the infrastructure that supported our work, and the time devoted to grammar writing itself.
2 Descriptive and Formal Grammars

2.1 Descriptive Grammar

2.1.1 Use of DocBook

At the beginning of the project, we had already elected to use DocBook, an XML-based representation widely used in industry for documentation, particularly (but not exclusively) of software. At the same time, it seemed inadvisable to use a programmer’s editor, or even a standard XML editor, to create this representation, since that would expose all the XML tags, a result which we felt would be confusing when we wanted to concentrate on the content, not the format.

Fortunately, there was a commercial product that answered our need: XMLMind (XXE, see http://www.xmlmind.com/xmleditor/). It entailed a learning curve, but because it is similar to standard word editing conventions, this learning curve was not too steep. (Indeed, one team member later suggested that it might make sense to write some of the TTO planning documents using this tool, modularizing the documents so as to make re-use of the parts easier.) Our experience with this tool has been almost entirely positive during the course of the project; in part this is attributable to the good support the company that developed this tool has given. The only drawback is that the company is located in France, thus preventing our subcontracting them when we needed modifications; this resulted in some delays, partially compensated for, however, by the fact that we developed in-house knowledge of the tool and of the DocBook formalism, knowledge which may serve us in the future.

The DocBook schema (http://www.docbook.org/) underwent a major revision between our work on the Bangla and Urdu grammars. As a result, the Bangla grammar is written in DocBook version 4 (which uses a Document Type Definition, or DTD, to specify the possible contents of documents), while the Urdu grammar is in DocBook version 5 (which uses a schema instead). The conversion did require modification of some of the local modifications we had made (see below re Literate Programming); these were accomplished without major problems, and indeed
the new schemas are much easier to modify than the DTD was. It would be possible to convert the Bangla grammar to the newer formalism, but this is not high on our list of priorities.

One addition required by our project to the standard DocBook XML schema was the ability to use Literate Programming (LP) techniques to embed the formal grammar into the descriptive grammar. This had already been done for a previous version of DocBook by Normal Walsh, the principle author of DocBook (see http://nwalsh.com/docs/articles/xml2002/lp/paper.html). With some help from mailing lists (this was our first foray into modifying DocBook), we were able to adapt these LP techniques to the version of the DocBook DTD used for the Bangla grammar, and later to adapt them to the DocBook schema used for the Urdu grammar. This approach has operated well, enabling us to embed fragments of the formal grammar at appropriate places in the descriptive grammar, and to extract the formal grammar, re-arranging the order of the fragments to match that required for processing.

One final, if more minor, lesson that we learned in this process is that the DocBook schema is more complex than we need. This showed up when an unwanted XML element (intended for use in descriptions of computer hardware) was accidentally used in our grammar. The result would not have been disastrous, but it highlighted the fact that the grammar writers were being forced to deal with more detail than they needed to, and that their task could therefore be eased if we eliminated the unwanted XML elements. This is in fact quite easily done; in fact, DocBook was designed with this need in mind (see the description in the section entitled “Deleting elements” at http://www.docbook.org/docs/howto/). We have accordingly eliminated over eighty unneeded elements in our local modification of the DocBook schema. (There is a working group that is building a subset of DocBook, called Simplified DocBook; for a somewhat dated version of this, see http://www.docbook.org/schemas/simplified. In the future, CASL’s localization of DocBook may be based on that subset.)

2.1.2 Formatting the Grammars

Since the grammars are written in XML, they must undergo a formatting process before they can be printed or used as PDFs. There are two mechanisms built into the XXE editor for formatting DocBook documents for publication: conversion to XML Formatting Objects, and conversion to one of several Microsoft Word formats.
We elected not to use XML Formatting Objects, as this technology is new and does not handle exotic scripts well. Instead, the draft and final versions of the Bangla grammar were created using XXE’s tools for conversion into Word doc (and later Word XML) format. However, this process required considerable post-editing in Microsoft Word to produce an adequate PDF. This was particularly true for formatting tables, some of which were too wide to easily fit on the page. While the ultimate output was satisfactory, we decided to explore alternatives that would not require post-editing. Moreover, for the Urdu grammar, we needed to use a Nasta’liq font (Nafees Nastaleeq, see http://www.crulp.org/software/localization/Fonts/nafeesNastaleeq.html). While Word normally handles multiple fonts well, the Nasta’liq script is so exotic that it does not display well in Word. Conversion to OpenOffice was no more promising.

The same computer scientist who invented the Literate Programming paradigm, Donald Knuth, had faced an analogous problem decades earlier with formatting his documents, which were heavy with mathematical notation; in response, he had developed the TeX typesetting system.¹ For many purposes, TeX was considered too low-level, so a higher level system called LaTeX was later developed.² TeX and LaTeX have since become widely used for typesetting journals and books, particularly technical ones.

However, TeX and LaTeX had two serious deficiencies for our purposes. The first deficiency is that both are format markup languages, whereas XML (including DocBook) is a content markup system. For our purposes, the use of content markup was crucial, since we wanted to be able to extract parts of the grammar (such as examples) for other purposes, and format markup languages do not lend themselves to this text (since various types of text might be formatted similarly). We therefore needed to convert from the XML content markup we used for development into the formatting markup we needed for output. Fortunately, there is open source software to do this; we chose dblatex (http://dblatex.sourceforge.net) for this. The author was helpful in explaining its use, and later (at our suggestion) in improving dblatex to produce output suitable for XeTeX (see below).

The second deficiency of TeX and LaTeX is that both work with the older 8-bit character encodings, whereas we were using Unicode. This was a serious limitation, as the use of Arabic (and Bangla) scripts would have made the use of 8-bit encodings very difficult. Fortunately, an SIL linguist, Jonathan Kew,\(^3\) had developed a Unicode-aware version of these programs which he called XeTeX (http://scripts.sil.org/xetex), and in fact used for typesetting Urdu. Some experimentation revealed that it was possible to produce output suitable for XeTeX from the dblatex program, and we had the core of our solution for typesetting (both in print and in PDF) the Urdu grammar. The addition of a set of LaTeX macros for interlinear text (made available by Michael Covington, of the University of Georgia (http://www.ai.uga.edu/mc/) added most of the additional tools we needed.

We still had to make some local modifications in order to make all these tools work together to typeset the Urdu grammar. Chief among the modifications was a way to convert our XML-based interlinear text structures into the form needed by Covington’s macros; David Zajic developed these.

The result worked so well that when we later needed to produce revised PDFs of the Bangla grammar, we decided to use the new dblatex/ XeTeX formatting tools in place of Microsoft Word. This, however, resulted in a problem: the difficulties we had encountered with fitting wide tables bit us again. We did not wish to repeat our earlier exercise of post-editing the output (this time in XeTeX instead of Microsoft Word). Fortunately, David Zajic and Chris Taylor were able to develop techniques to specify, in XML, how to make the wide tables fit by passing parameters to the XeTeX table commands. While this represents a slight mixing of content and format markup, it is probably the best that can be done, given the current state of computer-based typesetting.

There are some limitations for which we still want to find fixes. One of these is the fact that we do not as yet have the ability to bold particular words in interlinear text, which would allow us to draw attention to the parts of example sentences that illustrate a particular point. This can surely be implemented, but it will require some additional work.

\(^3\) Jonathan Kew has since left SIL.
2.1.3 Example Sentences.

We have also not had time to develop ways of nicely formatting the formal grammar in our outputs; the result is that the formal grammar appears as XML. Again, we know this is doable, but it will require additional work.

As with any descriptive grammar, we needed to include example phrases and sentences of Bangla and Urdu to illustrate the grammatical constructions we were describing. The example sentences in the Bangla grammar were constructed using the tools available in XML Mind's DocBook, primarily the “informal table” construct. These tools proved cumbersome for many reasons, chief among them the tables' opacity to copy and paste commands: as Urdu examples could not be constructed in other applications and pasted in, the team's Urdu experts and descriptive linguists could not work simultaneously on the same paradigms, creating bottlenecks; and as blank template tables could not be created and pasted in, team members had difficulty maintaining consistent formatting across chapters. Constructing Urdu and interlinear examples within the DocBook program itself was complicated by the need to combine right-to-left with left-to-right text. Moreover, the tables were ill-suited to displaying long sentences, nor did they provide an easy way to format the examples.

For the Urdu grammar, we therefore decided to use a different mechanism to create and store examples, one which would allow us to create interlinear text of the sort linguists commonly use for this purpose. In languages with inflectional morphology (which of course includes both Bangla and Urdu), such interlinear text typically uses three lines: one line for the sentence in the target language (Bangla or Urdu), written more or less normally; one line with the words broken into glossed morphemes; and one line for the free translation (generally into English). The first and second lines are generally aligned at the word level. However, since Urdu is written with a right-to-left script (Nasta’liq, a special Arabic script), it would be effectively impossible to align with a gloss line. Since we wanted to include the Arabic script, we included a fourth line at the top for the Arabic script, and wrote the same text in a Romanization for the first aligned line. This would have made the use of tables for displaying interlinear text even more inconvenient. We thus needed a way to create interlinear text, store it in our descriptive grammar, and display it.
For the creation of interlinear text in the Urdu grammar, CASL originally investigated SIL’s FLEx (Fieldworks Language Explorer), eventually installing it on "virtual" PCs, as it was not on the list of CASL-approved software. (It is open source, and has non-US citizen involvement.) But in the end, the 1308 team concluded that using FLEx was more difficult than it was worth (particularly as we considered the necessity of exporting the interlinear examples from FLEx and importing them into our XML documents), and elected to simply edit the interlinear text in place in the XXE DocBook editor.

Fortunately, SIL had also developed an independent XML-based grammar formalism, which included constructs for interlinear text. Since we wanted to remain with the industry-standard DocBook schema as much as possible, we adapted the constructs for interlinear text from the SIL formalism, modifying them slightly for our use, and added them to the DocBook schema. (DocBook, in the meantime, had grown from the DTD-based version 4.x, which we used for the Bangla grammar, into the RelaxNG schema-based version 5, which made it much easier to add the interlinear text schema to the native DocBook schema.) This schema gave us the mechanism for representing interlinear examples in DocBook. SIL had also developed a Cascading Style Sheet (CSS) for XXE, which we again borrowed and modified, allowing the interlinear text to be edited in XXE.

The final requirement was a mechanism for converting interlinear text to PDF, within the overall process for converting DocBook text to PDF. Team member David Zajic solved this part of the puzzle by converting the DocBook representation to a XeTeX format, which was further converted into PDF using open-source LaTeX macros developed by Michael Covington.

The final result was a relatively easy-to-use method for creating interlinear text, store it in our XML grammar, and format it into pleasing output, including the parts written in the difficult Nasta’liq script.

**2.1.4 Word Annotation**

The Bangla and Urdu morphological parsers need to be tested on annotated texts. In principle, this could be text-tagged in XML with the morphological information the parser is designed to elucidate. However, it seemed likely that it would be difficult to produce significant quantities of
text tagged at this level of difficulty: marking each affix for its meaning is tedious at best, and can be impossible for native speakers to do. While the CASL team could do this tagging, we felt it would be a poor use of our time (particularly since none of the CASL team is a native speaker of these languages, meaning that we would continually need to refer to dictionaries); and in any case would represent a conflict in the sense that we would be doing markup to test our own parser.

Accordingly, we decided on a simpler, but still useful, form of annotation: each “token” (roughly, word) of the test text was to be annotated with the citation form of the word. (In cases where there was no citation form, such as proper names, numerals, or punctuation, the word would be marked “none.”) This, we felt, would be within the capabilities of educated native speakers of the languages, just as it doubtless would be for even high school-educated speakers of English.

With this in mind, we recruited speakers of Bangla and Urdu to do the annotation. An initial issue was that if the speakers we found were not US citizens, they were required to pass a National Agency Check (NAC), a process which was said to take up to three months. In fact, the process took much longer than that, and in the end was completely impractical. The implication was that we needed to recruit educated fluent speakers who were US citizens. This process turned out to be difficult as well, although not as slow for the languages we needed as the NAC process.

Our first recruit was a speaker of Bangla who had received the equivalent of a Bachelor’s degree from a university in Bangladesh. Despite considerable coaching from us, he proved unable to perform the annotation at an acceptably accurate level (as judged by us, referring to dictionaries). We were surprised; in retrospect, we believe the problem can be attributed to several factors:

1) the inherent complexities in Bangla and Urdu morphology;

2) the lack of orthographic standardization (i.e., variant spellings);

3) the fact that the task was designed to work with existing resources (electronic texts provided by the Linguistic Data Consortium (LDC) and electronic versions of the dictionaries), but without making modifications to or fixing errors in these resources; and
4) (perhaps) to the schooling system in Bangladesh, which does not appear to emphasize dictionary lookup as much as schools in the US do.

We believe that the difficulties to some extent validate the goals of this project: dictionary lookup for inflected languages is difficult, even for native speakers.

Fortunately, we were able to find another speaker of Bangla who could do the work. At the end of the task, she commented that this was the most interesting job she had ever had. We doubt that most English speakers would say this about dictionary lookup of an English text; again, the difference might be due to the greater challenge in dictionary lookup in inflected languages, a challenge which this annotator relished.

We also found a speaker of Urdu to do the corresponding task for that language.

The Urdu speaker did the annotation work on her own computer, and emailed us the results. The Bangla speaker wanted to do the work on her Macintosh; however, there was a problem: the Macintosh cannot display Bangla text using Microsoft Word (which was the tool we were using for annotation). After we considered sending her a loaner laptop (this consultant lives in California), CASL’s IT department suggested the use of a “virtual” PC. This is in essence a computer running Microsoft Windows (and hence capable of correctly displaying Bangla text), but isolated from most of CASL’s internal network for security reasons. Using remote desktop software provided by CASL’s IT department, the annotator was able to log in to this virtual computer remotely and complete the task. We expect to use this technology for annotation in the future.

The obvious lessons in our annotation task are that the task is difficult; finding qualified speakers who can do the task is time-consuming; and therefore searching for annotators needs to be started early in the course of working on a particular language. Indeed, assuming the next language CASL works on is Pashto, finding suitable annotators is likely to be even more difficult, due to the low level of literacy in Pashto, the lack of standardization in Pashto writing,
and the expected difficulty in finding Pashto speakers willing to work with us.4

The question of ensuring consistency in annotation is a related issue that we are still struggling with; this is discussed below, in the section on Variant Representations.

2.1.5 Morphological Complexities

The Bangla and Urdu morphological parsers are meant to focus on inflectional morphology, or grammar-driven changes to a word's form. Inflectional morphology involves relatively productive and predictable patterns, such as the plural –s suffix for English nouns, or the variant between the object pronoun us and the subject pronoun we. Derivational morphology, on the other hand, deals with meaning-driven changes, such as the generation of directive, direction, and director from direct. This type of variation is less predictable—not every affix is attested combining with every compatible word stem—and the resulting word forms are generally stored as unique words in the lexicon, and often classified as different parts of speech than the word or stem from which they are derived.

Unfortunately, the distinction between inflectional and derivational morphology, and hence decisions regarding which derivational processes to incorporate into the morphological parsers, are not always straightforward. For example, what are known as “derived transitive” or “single causative” verbs in Urdu are derived by adding –aa at the end of a “plain” (often with intransitive meaning) verb stem: e.g. pakaRaanaa “to cause to catch, grab” from pakaRnnaa “to catch, grab”. Although somewhat productive, such causative/transitive verb stems are generally considered a part of derivational morphology and are listed separately in the Urdu dictionary. In the above example, however, only the “plain” form pakaRnnaa appears in the dictionary, and occurrences of the causative form pakaRaanaa had to be left unannotated. This problem, of missing entries in the dictionary, might be overcome by allowing the –aa verb stem suffix to be accounted for by the morphological parser; however, this would require searching for and re-annotating the derived verb forms. In any case, the use of this particular affix is not documented in a systematic manner in the Urdu dictionary. Similar problems emerged with other morphological processes, such as adjectives derived via the suffix –ii.

4 Pashto “linguists”—we believe this term is used elastically—are in extremely high demand at present, and command correspondingly high salaries, according to job announcements we have seen on the Internet.
Another major problem involved variant representations of various compounding processes in Urdu. In addition to compound nouns similar to those of English, Urdu has a rich system of compound prepositions and verbs. Certain verbs in these compound constructions carry the semantic function of adverbs, tense marking, etc. For example, in the form jaa raha hai (“he is going”), the main verb jaana (“to go”) is combined with a form of the verb honaa (literally “to be”), marked for 1st person singular in this case, and with the verb rahnaa (literally “to remain”), marked for masculine, to create the masculine-singular present progressive tense form. For the purpose of morphological parsing, we have favored treating these as separate units (in which case the annotations for these special occurrences are identical to annotations of the same verbs when they are used in their literal senses). However, in some texts, these constructions appear as single words, and so we have had to decide whether to treat such occurrences as un-annotatable, or to allow for variant spellings and train the parser to account for both forms. Although issues of this sort are resolvable, they have required considerable interaction with contracting annotators, and this has posed its own problems of ensuring consistency of annotation throughout the texts.

A related issue is that of spelling variation. The spelling of Bangla and Urdu is not as standardized as it is for English, so the same word may be spelled in two or more ways, and annotators may disagree with the spelling in either dictionary or in texts. We have treated the latter problem by considering the dictionary to be the authority (in part because our application is dictionary lookup), but this does not resolve the problem of divergent spelling in texts and in the dictionary. At its simplest, our parser can only look up words if they are spelled the way the dictionary spells them (apart, of course, from alternations caused by the morphology and phonology).

There are, however, some work-arounds for the spelling variation problem. The parsing engine we are currently using (the Stuttgart Finite State Transducer, SFST) has the capability of searching for words which are within a specified “edit distance” of the expected form, where “edit distance” refers to the number of changes which are required to change the expected form into a lookup form. In the simplest form, edit distance is defined as the number of deletions, insertions, or single letter changes required.
For example, the mis-spelled form *cip* is within an edit distance of one from the words *cap, cop* and *cup*, since the change of the vowel *i* to one of the other vowels will result in one of these other words. It is also within an edit distance of 1 from *dip, hip, lip, nip, pip, rip, sip, and tip*, based on a change to the first consonant; and the same distance from *chip*, since insertion of an *h* results in the latter word. As this example illustrates, edit distance can be a blunt instrument for spell correction: it can result in a very large number of potential corrections (although this number is likely to be lower for longer words). A better solution—if a more costly one, in the sense of the amount of work required—would be to investigate the kinds of spelling variants and outright errors people make, or are likely to make, and to suggest those as the most likely corrections.

### 2.1.7 Fallible Resources

In addition to simple inconsistencies in morphology and spelling, various outright errors were also discovered in the electronic Urdu dictionary. In some cases, these involved major, high-frequency lexical items such as the verb *aanaa* (“to come”) which was misspelled as *aayaa* and incorrectly listed as a transitive verb. In general, we have chosen not to fix these errors at present, since we plan to test the morphological parser with existing, proprietary sources, and because there is probably a more general need for repairs to the dictionaries, one sub-task of which could be fixes to this kind of inconsistency.

Most of these problems would be solved by a consistent and accurate dictionary; in the absence of one, they have required time-consuming checking and human removal or adjustment of annotations of words which we know the parser cannot handle. Subsequent work with other languages should take the reliability of existing electronic resources into account, and allot time accordingly. In addition, checking the customer’s electronic dictionaries for a given language for problems which would affect the parser should be done early in the work on that language, rather than waiting until later.

### 2.1.8 svn Server

When the Bangla grammar was written, we had no version control mechanism except SharePoint—which is to say, no version control at all, since SharePoint did not work with the
XXE editor, nor did it supply a differencing mechanism compatible with non-Microsoft Office XML documents. Drafts were therefore stored on individual researchers' hard drives; only occasionally were they uploaded to our Linux machines, where they could be put into an older version control system (RCS). This caused some crossing of drafts and duplication of effort. Moreover, team members working offsite had no easy access to drafts, save by emailing copies to themselves, a process which was dangerous (if one forgot to send the email, one lacked an up-to-date version of the document). We mitigated this problem to some extent by breaking the grammar into chapters, so that failure to email a modified copy of chapter six from home to the office meant that that chapter could not be safely edited at the office until the next day, but the other chapters (assuming they had not been modified at home) could be edited.

This situation was obviously unacceptable, and we therefore requested CASL’s IT department to install an svn server. This is a server accessible over an http link (in our case, we used https for security). It provided both version locking (so that one person could not accidentally save a copy of a chapter over another’s edits and thus destroy them) and version control (allowing us to find the differences between two versions of a given document, which we used on at least one occasion when editing errors had been made). It would also allow us to revert to an earlier version of a document, should some edits be completely wrong (although we never used this capability). It took us some time to work out the kinks and learn how to use this system with our XXE editor (fortunately, the XXE editor has built-in support for use with an svn server).

In writing the Urdu grammar, we now store all our chapter drafts on the svn server, from which they can be checked by any authorized user—though only edited by one at a time. Authorized users can also check out drafts remotely, from anywhere, a functionality which greatly facilitates outsourcing, external review, and formatting for publication.

2.2 Formal Grammar

In the formal grammar, we were able to leverage earlier work done by Mike Maxwell on a formalism for modeling morphology and phonology, which provided a basis for describing the morphology of Bangla and Urdu. Slight modifications of the formalism were done to simplify
the writing of phonological rules and allomorphs—e.g., the use of strings rather than sequences of phonemes.

The result has been a stable way of representing the formal grammar of morphology and phonology. Apart from the minor changes described above, the only major change during the project has been to convert from a W3C schema (http://www.w3.org/XML/Schema) to a Relax NG schema (http://relaxng.org/). This was done primarily for reasons of compatibility with the Relax NG schema used for DocBook version 5, but it has proven to be much easier to maintain than the W3C schema was.5

2.2.1 Conversion to Parser

Using a pre-defined XML formalism has made it reasonably straightforward for us to create a modular program, written in the Python programming language, to convert the XML interchange format into the programming language of a morphological parsing engine. CASL expects this program to have a significantly long lifetime, if not as long as XML.

The converter program was written so as to divide the conversion into two stages; these two stages correspond to the notion of a front end and a back end for compilers (see e.g. http://en.wikipedia.org/wiki/Compiler). First, the XML representation of a grammar is converted into the program’s internal object-based representation, where each class of objects corresponds to the XML elements in the external formalism. In the second step, the internal representation is converted into the programming language of the chosen parsing engine, in this case that of the Stuttgart Finite State Tools (SFST). SFST was chosen by the customer on the basis of CASL’s M.1 deliverable under TTO 308, and has generally proven to be a good choice. The developer, Helmut Schmid, was cooperative in fixing bugs and providing guidance when problems arose, particularly problems with documentation.

One significant problem, discovered early on, is that while the memory usage of the optimized network is minimal (as tested by CASL and verified by LKS), SFST can use excessive amounts

5 The relative ease of use of Relax NG over that of the W3C schema has been confirmed in informal conversations with other developers of XML schemas; indeed, that may be why Relax NG was chosen for DocBook version 5.
of memory during the compilation phase. In fact, during compilation of the Bangla grammar this memory usage exceeded the amount of RAM in CASL’s development computers, resulting in thrashing. As CASL discovered (and as Helmut Schmid later verified), SFST performs memory optimization at the end of each “statement” in the programming language. The solution to the memory usage problem was therefore to make CASL’s converter break the grammar into smaller statements, allowing optimization after each one. This dropped the maximum memory usage from well over sixteen gigabytes for the Bangla grammar to under five gigabytes. (The need for optimizing the SFST code emitted by the converter is analogous to the optimization of machine code done by modern program compilers.)

Another issue with SFST has not yet been completely resolved: there appears to be a bug in the SFST compiler which surfaces only where there is a large memory usage, and then only on a Sun machine. This bug, verified through testing on a Sun machine at the University of Maryland's UMIACS facility, crashes the SFST compiler. This is a significant problem, for hardware reasons. When the grammar networks are compiled on an Intel machine, such as one of the Linux computers used by CASL, they are incompatible with the format used by LKS, which employs Sun machines: specifically, the byte order of the compiled networks differs across the two hardware types, and network compilation does not produce executable code (in the sense of compilation of typical programming languages.) Since these two formats are incompatible, networks intended to be run on Suns need to be compiled on Suns—but the bug in the SFST compiler prevents this. (At least in some cases. It appears when we compile the Bangla grammar, but not when we compile the Urdu grammar, due to the greater complexity of phonological rules in the former.)

Several fixes were attempted before CASL discovered a work-around. The compiler bug was known to occur in SFST version 1.1; since newer versions of SFST were available, we hoped the bug might be fixed in those versions. (There was considerable re-factoring between v1.1 and v1.2, making this at least plausible.) Unfortunately, the customer was only approved to use v1.1, and it is not known when v1.2 might be approved. Moreover, the UMIACS Sun machine had far too little memory installed for it to be used routinely for compiling grammar networks (compiling the Bangla grammar to the point of crashing took over ten hours, vs. about 45 minutes on the LKS machine with adequate memory—clearly a thrashing problem). Also, even
if version 1.2 or 1.3 turned out to be capable of compiling the grammars without crashing, the compiled network would not be acceptable to LKS, because they would have been compiled using an unapproved version of SFST.

CASL also looked into the possibility of converting the compiled network from the Intel format to the Sun format by changing the byte order. While this is simple in principle, it is more difficult in practice.

In the end, CASL devised a work-around based on the fact that the SFST compiler can produce three different formats, which can be “run” by three different SFST programs. The default format is not compatible between Intel and Sun machines, but another format—the “compact binary transducer format”—can be run on either machine, using a command line parameter at execute time to run a network compiled on one architecture on the other architecture. Using this format, CASL can compile grammar networks on an Intel machine using the approved version 1.1 of SFST, and LKS can run this compiled network on a Sun using the command line parameter. An added benefit of this format is that, as its name suggests, it uses significantly less memory at run time. Additionally, while the SFST documentation states that this format may be less efficient if grammar ambiguity is high, tests thus far have demonstrated that the compact format runs almost ten times faster and uses less memory. (Presumably this would not be true for highly ambiguous grammars.)

CASL and LKS would probably not have discovered either advantage, had it not been for the bug in the SFST compiler; so, in this case, a bug led to increased efficiency. Nevertheless, CASL believes it would be advantageous to acquire a Sun machine (with adequate memory) for testing this and other deliverables on the architecture used by the customer, particularly as LKS recently confirmed its decision to continue using Sun machines instead of Intel-based Linux machines.

2.2.2 General Processing Issues

During development, the conversion process from XML to PDF and from XML to the SFST programming language was controlled by multiple ‘makefiles,’ using the common Unix “make” utility. While this approach ensures that all processes needing to be run can be run quickly and in the correct order whenever a change to “source code” (such as the input XML files) makes it
necessary, it can be cumbersome to debug. Packaging this method for delivery to the customer is also difficult. Accordingly, David Zajic developed Perl scripts to run the entire XML-to-PDF process from beginning to end. This can be slower than using makefiles during development, since a change to any source file means that all dependent files need to be re-built (whereas with makefiles, only the necessary re-building is done); but this potential slowdown is not an issue for the customer, who will generally be making the product from scratch, and will not usually be modifying source files.

2.2.3 Coordination

Our overall goal, of course, has been to combine the descriptive and formal grammars, letting the human-readable descriptive grammar elucidate the formal grammar. To this end, we developed the two grammars in parallel.

Our working arrangement has been one of iterative development, with descriptive grammar writing leading formal grammar writing. Crucially, this iterative development allows frequent exchanges for clarification. A typical interchange might go as follows: The language expert writes a section of the descriptive grammar on noun qualifiers. The computational grammar writer reads the description and attempts to implement it, but a question arises: is the diminutive qualifier used in all the environments that the three allomorphs of the non-diminutive qualifier are used, or does it appear in only one of those environments? The language expert finds examples showing the diminutive in all environments, enabling the computational grammar writer to proceed. Crucially, the descriptive grammar is modified to clarify this issue, and to include the new examples.

Although we are writing our grammars a short hallway apart, this interchange was accomplished largely by email; the writers could as well have been a continent apart. With a secure server holding the drafts, and good version control, the greatest impediments to effective collaboration among the team were almost comically simple: stylistic inconsistencies, and skill bottlenecks.
2.2.4 Editorial Issues

One of the earliest steps in developing subsequent grammars needs to be for all team members to sign off on a style sheet. The issue is less consistency in serial commas and such basic mechanical issues—though that does greatly simplify the final editorial pass—as consistency in all the aspects of a document which require special encoding:

- **Hierarchical Structure.** At what level of detail is a new subheading justified?

- **Table Structure.** Both internal—how much information should be put into a single cell, and in what format; and external—how and when to mark long tables to 'float.'

- **References.** Including cross-references, internal and external citations, and footnotes.

- **Tags.** In both the formal grammar and the descriptive grammar, a list should be kept of all tags and of the circumstances of their use. This list may be generated while the grammar is written, but a list should be kept.

- **Abbreviations.** This list may also be generated during writing, and can be included as an appendix in the finished document.

- **Interlinear text.** Which examples do and do not require it, which information belongs on which line, and the degree of alignment between text rows; also, any techniques required to keep interlinear blocks together, such as encasing examples in text boxes.

- **Character sets.** Any non-Roman fonts to be used, their Roman transcriptions or transliterations, and any diacritical marks; these should be agreed upon at the beginning, and tested to ensure they display properly in tables, interlinear examples, etc.

2.2.5 Technical Issues

All team members should also be trained, before writing begins, in all the technical skills which they are likely to need. Chief among those, for this grammar, have been making PDFs from XML, and a variety of XML Mind skills: xrefs, attributes, tags, footnotes, tables, and interlinear text.
2.2.6 Implications

We draw out the implications of this issue in some detail here, because these were, in our opinion, one of the most important results of the project thus far, and at the same time a result that we had not anticipated.

As we mentioned above, while we the authors of the descriptive and formal grammars were working in the same building, this was not a pre-requisite to success. Had we been on opposite sides of the Earth, we could still have conducted the necessary back-and-forth interaction by email and phone. This means that both the descriptive and the formal grammars could have been contracted out independently, so long as the contractors could work together; or only one or the other grammar might have been contracted out, leaving the other grammar to be done in-house.

At the same time, it was important that we, the authors of the descriptive and formal grammars, worked at the same time. More precisely, while the formal grammar writing must necessarily lag the descriptive grammar slightly (since the formal grammar is based on the descriptive grammar), it is important that the author of the descriptive grammar have the opportunity to immediately incorporate feedback from the formal grammar. That is, the descriptive grammar could not have been written and delivered in its entirety before the formal grammar.
3 Software Tools

We have described in previous sections most of the software tools we have used. We list those and some additional tools here. It is noteworthy that many of the tools we have found to be necessary in our work are open source tools, and some of them are foreign software. One implication of this is that we have confined most of these tools to our Linux systems, and in one case a tool resides on CASL’s “virtual” PCs (computers running the Microsoft Windows operating system, which are isolated from most of the rest of the network).

<table>
<thead>
<tr>
<th>Tool</th>
<th>Purpose</th>
<th>Operating System</th>
<th>Tool Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblatex</td>
<td>Convert DocBook to XeTeX for formatting</td>
<td>Linux</td>
<td>Open source (<a href="http://dblatex.sourceforge.net/">http://dblatex.sourceforge.net/</a>) by Benoît Guillon, France</td>
</tr>
<tr>
<td>XeTeX</td>
<td>Format text for publication</td>
<td>Linux, Mac</td>
<td>Jonathan Kew, UK (former SIL member)</td>
</tr>
<tr>
<td>jEdit</td>
<td>Programmer’s Editor</td>
<td>Windows, Mac, Linux</td>
<td>Open Source (<a href="http://www.jedit.org">http://www.jedit.org</a>), multiple contributors</td>
</tr>
<tr>
<td>Covington macros</td>
<td>Format interlinear text for publication (Used in XeTeX)</td>
<td>(Used in XeTeX)</td>
<td>Dr. Michael Covington, University of Georgia</td>
</tr>
<tr>
<td>svn</td>
<td>Remote access to files, version control</td>
<td>Linux</td>
<td>Open source (<a href="http://subversion.tigris.org/">http://subversion.tigris.org/</a>), multiple contributors</td>
</tr>
<tr>
<td>Stuttgart Finite State Transducer</td>
<td>Parsing engine</td>
<td>Linux, Solaris</td>
<td>Dr. Helmut Schmid, University of Stuttgart, Germany</td>
</tr>
<tr>
<td>Tool</td>
<td>Purpose</td>
<td>Operating System</td>
<td>Tool Provider</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------------------</td>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>xmlint, xsltproc</td>
<td>Verifying and processing XML</td>
<td>Linux</td>
<td>Open source (<a href="http://www.xmlsoft.org/XSLT/xsltproc2.html">http://www.xmlsoft.org/XSLT/xsltproc2.html</a>), Daniel Veillard, France</td>
</tr>
<tr>
<td>xmlgrep</td>
<td>Verifying and testing XML dictionaries, preprocessing of formal grammar</td>
<td>Linux</td>
<td>Open source (<a href="http://search.cpan.org/dist/XML-Twig/tools/xml_grep/xml_grep">http://search.cpan.org/dist/XML-Twig/tools/xml_grep/xml_grep</a>), Michel Rodriguez, Italy</td>
</tr>
<tr>
<td>grepp, ped</td>
<td>Unicode utilities</td>
<td>Linux</td>
<td>Open source (<a href="http://www.perlmonks.org/?node_id=345275">http://www.perlmonks.org/?node_id=345275</a>), David Graff, LDC at the University of Pennsylvania</td>
</tr>
</tbody>
</table>
4 Costs

4.1 Salaries

A total of approximately 10,000 staff hours was required to produce the two grammars. Roughly 5,250 hours were spent on the planning of the initial project and the Bangla grammar production, and roughly 4,750 hours were spent on training new staff and production of the Urdu grammar.

The university is not equipped to track effort by task and subtask, so these hours are approximations to the best of our ability. The estimated costs for staff salary were:

$250,960 for Bangla and $227,059 for Urdu.

4.2 Direct payments to language experts

The team employed a series of consultants who were either native speakers, experts in the language, or both. These individuals helped to clarify contradictions between sources; identify errors, exceptions and omissions; provide example sentences in the language; and annotate corpora to be used as an input to parser testing. The rates of pay varied from $25 to $67.50 per hour, and roughly 400 hours were required for each language. Payments to these consultants were approximately $20,000 per language.

4.3 Relevance and Applicability of this Data

This data shows that now that a process has been established and one grammar produced, costs for related languages can be reduced through use of more junior staff (under supervision of senior researchers). It is important to note that the hours and costs related to these two languages are not indicative of the hours and costs required for other languages, as each language has its own challenges. Each language should be evaluated independently to determine the level of effort required to produce a dual-use grammar.
Specifically, the team is expected to begin work on Pashto, which has far less documentation concerning its grammatical features, as well as several regional dialects which will be hard to capture. This will require a different approach than that used for Bangla and Urdu, and will present new challenges for the team to overcome. As such, we anticipate the level of effort to be significantly different than what was required for Bangla and Urdu.