1.0 Introduction.

The American Indian Studies Research Institute (AISRI), in Bloomington, Indiana, and its collaborators are presently engaged in several long-term language documentation projects. These projects closely integrate lexicography, text corpora, archived cultural information and historical records. They focus primarily on Northern Caddoan languages (especially Arikara and Pawnee) and Siouan languages (Assiniboine and dialects of Sioux). These projects undertake linguistic research but also support language stabilization by partnering with school districts and communities in North and South Dakota, Oklahoma, and Saskatchewan. From the founding of our institute in 1985 we have made every effort to make effective use of technology including sound and video, and software tailored to the special needs of language documentation and analysis.

We will discuss the development of our linguistic text-processing application, the Annotated Text Processor, or ATP. This work was funded by the NSF as part of Douglas R. Parks’ “Northern Caddoan Dictionaries Project.” We will focus on the design of data structures, annotation, and metadata in our text corpora, dictionaries, and digital library projects. Then we want to turn to the problems of exposing metadata, data, and web-services for efforts like EMELD and OLAC, and how we think these new requirements will shape our future efforts.

2.0 The ATP Document Interface

We developed ATP in Borland C++ Builder and it operates on all Windows platforms from 95 to 2000 and XP. The components exist to make ATP available in Linux and Macintosh versions and we simply need the time and resources to do so.

The interlinear document being displayed in the Editor is the first paragraph of a Skiri Pawnee Coyote story from the Mad Bear collection analyzed at AISRI by Doug Parks and Bill Anderson. They established a phonemic rendering of the text, added a literal gloss to each form, parsed those forms into morphemic constituents, and then glossed each of the morphemes.

One of the basic goals of ATP was to help a linguist produce a camera-ready version of an interlinear text as quickly and painlessly as possible. As an interlinear-text formatter ATP calculates all the spacing and wrapping as the linguist works through the text—the linguist is largely free to concentrate on the content and analysis of the forms themselves. But ATP also
provides a large number of tools and functions to aid the linguist in the analysis of the text, including glossing and annotation interfaces. It provides import interfaces, support for user keyboards, sort orders and character weighting, and flexible font management, as well as direct editing access to underlying tables and files.

Figure 1. ATP Document Interface

[• Document Editor. Fig. 1.] The center control (marked by the red dot now on screen) is the document editor. It is an enhanced RichText control and allows ATP to interact with the user, keystroke by keystroke. [• Font and Display Options in the Editor. Fig. 1.] The user can specify fonts, styles and sizes, and can choose to display everything in an interlinear block or hide some part of it, depending on what is most convenient for the task at hand. For example, the morphemic levels can be hidden to concentrate on glossing.

[• ATP Interface’s Document Model. Fig. 1.] Two tree views stand to the left of the document editor. The upper tree view shows the underlying data model and currently active display options. The data model and the controls above it are used to insert new elements, and to reformat or remove existing elements. The document model and display options are defined in a separate interface invoked from the menus. The lower tree view is a document map that allows...
the user to jump to particular elements in the text.

[• Annotator Query Controls. Fig. 1.] The user can trigger ATP’s glossing and annotation tools by typing control-enter at the element in question or by using the controls to the right of the Editor. The results of queries and glossing choices are displayed in the list box indicated by the red dot.

[• ATP Media Player. Fig. 1.] At the bottom of the main interface is ATP’s media player. This control is capable of playing WAV and AIFF sound files, and MOV, AVI and MPG video files. The media player can track through a media file and set positions down to a millisecond. The linguist doing a transcription uses the convenient controls to determine the start and end points of successive media segments and then stores those positions with the text data. In later sessions ATP will read those positions and cue up and play the media segments without further effort. We were scheduled to demonstrate ATP’s media player and transcription capabilities with Brenda Farnell’s Plains Sign Language project Saturday afternoon.

3.0 ATP Data Structures

![Figure 2. ATP Data Modeling Interface](image-url)
ATP’s strategy is to support entirely flexible, user-defined data models that are tree-like and hierarchical. The next slide shows ATP’s Data Modeling Interface. The model governs the contents of the document and validates its structure as it is being created, and supports the various functions that ATP applies to contents. Much of ATP was designed before XML specifications were generally available so there are differences between “native” ATP and XML. But we did pay attention to SGML and the Text Encoding Initiative so ATP operations will feel familiar to most participants here. We are integrating XML components and libraries, especially expat, a widely distributed SAX processor, into a new version of ATP and when that work is finished ATP should be a fairly seamless XML-processor.

This detail of the Data Modeler shows how an ATP data structure is organized. This model was used for the Skiri Pawnee text corpus including the Mad Bear narrative we just saw. All of the texts in the Skiri collection follow this fairly simple form. Parks needed elements for the story number, title, and narrator. And we were able to model the main body of the documents as a series of paragraphs of the same structure.

Each paragraph in this model contains of a series of sentences. And, in turn, each sentence element contains three child elements: first a series of Skiri phonemic forms, then a single free-translation element, and a single comment element. Each individual Skiri phonemic form has its own individual gloss field and its own series of morpheme elements, and each morpheme has a single morphemic gloss.
The ellipses that appear after several elements indicates that those node types are processed as a series of successor elements. ATP trees supplement the idea of sibling elements with the idea of successor elements. Successor elements have the same label and data type, and ATP will process a series of successors as a kind of segmented object, especially in querying and formatting operations. From the point of view of XML, however, ATP successors will be treated as just another sibling type with certain attribute values in common—as an arc in fact.

The use of the paragraph element in this model accords with the organization of the historical texts themselves. We could have organized the data without defining the sentence element as a child of the paragraph element. Instead we could have defined the sentence element as a sibling to the paragraph element and interspersed empty paragraph elements among the sentences as flags to mark paragraph breaks in the documents. ATP supports either scheme without breaking stride.

Figure 4. Data Model for Nakoda (Assiniboine) Project

[ Model for the Nakoda (Assiniboine) Project. See Figure 4.] ATP can support very complicated data models. This model was developed for the George Sword Texts collection at AISRI but it has been used for other Lakota collections as well. The historical documents differ in structure,
and our four analysts had different purposes in mind, so five different paragraph types have been defined. Notice also that two elements, *beginsound*, and *endsound*, near the red dot, are included to store media file bookmarks with the text data. Users can insert such storage fields anywhere in their models so that it is possible to link sound and video with text at any level of granularity.

3.0 ATP’s Present Storage Scheme

[• ATP Storage title slide.] ATP presently uses relational database tables to provide long-term data storage in manipulable formats. We can use these same tables to serve data to web browsers if we choose. (In fact we also have the option of serving XML documents derived from these tables.)

[• Detail of the Document Table Viewer. Figure 5.] However, to support the complete arbitrariness of document models we *denormalize* our database structures. We put all the interesting data into one field that is labeled TEXT on this slide. We have two other fields to preserve information about element types and document position and use these to recover distinctions and structure.

![Figure 5. ATP Document Table Viewer](image)

The LABEL field preserves information about element types. To store information about
document structure we have created a scheme for enumerating elements based on the document model. That scalable enumeration is stored in the NODENAME field.

Three quick steps will demonstrate how the enumeration works. For this example we can use SQL filters to select all paragraphs. [A Listing of the Paragraph Elements with Their Enumeration. Fig. 6a.] Notice their NODENAMEs. Now we throw in the sentences. [Paragraph and Sentence Elements and Their Enumeration. Fig. 6b.] Finally we add the phonemic forms. [Paragraph, Sentence, and Phonemic Forms. Fig. 6c.] An element’s NODENAME expands with its depth. A child element’s NODENAME always include its parent’s and ancestors’ NODENAMEs. And NODENAME values work with the table’s index to keep all the records in document order.

Figures 6a and 6b. Enumeration of Paragraphs and Sentences
This scheme allows us to exploit SQL to manage or query our denormalized data tables. The user can set a filter to select all the nodes whose label is *phonemic_form*. Or, the user can set a filter that selects all the nodes belonging to one sentence. This approach allows us to work with any document structure and any set of labels, and to pull back any desired subset in almost no time at all using SQL and regular expressions. We can use these capabilities to rapidly support new web-services initiatives while we continue to bring XML online at AISRI.
ATP’s database storage scheme has some advantages, including the ability to store and manipulate very large documents even when they are heavily analyzed and annotated. They can be served to the web rather directly through web-based applications. However, these database files also tend to be very large themselves and they don’t offer the simple, accessible convenience of lightly marked text files like Shoebox uses. We do continue to admire the design of Shoebox. Of course neither is XML as friendly as Shoebox text but we expect that XML storage will significantly reduce our current storage needs.

ATP does export RTF and plain text outputs that can be read by major word processors on different platforms, and the RTF can be used for camera-ready publication, but these exports cannot retrieved by ATP for further processing if changes are made to the RTF or plain text.

4.0 Annotation in ATP

ATP is designed to aid user annotation of corpus texts, especially repetitive annotation. We want to give you a whirlwind tour of ATP’s annotation tools before discussing the impact of EMELD’s initiatives on our own projects.

ATP records glosses and annotations from each analysis and then retrieves those stored data in subsequent queries, so ATP can be used to speed up the process of translation or analysis. The slide shows a case of a glossing query in one of the Nakoda texts. The user’s caret or point of insertion in the Editor is placed on the gloss line next to the gold dot, and is querying for a gloss for mitúgaši. The query results are returned to the list box marked by the red dot in the query control.
Figure 8. Details of Manual Query and Regular Expression Query

[ Detail of a Manual Annotator Query. Figure 8.] This closeup of the query control shows that two possible results have been found in the annotator table. The difference between them is that the second instance is ablauted and that is marked by the raised / at the end of the word. The user can make a choice and insert the selected item into the text using the controls near the gold dot at the bottom.

[ Detail of a Regular-Expressions Query. Fig. 8] ATP also queries by regular-expressions. The user can enter the expression in the Search form box next to the red dot and tell ATP to Find again. The results are listed as before.

Figure 9. Sort and Search Weight Interface
[• Defining Sort Orders and Search Weights. Figure 9.] The query was able to return two instances of *mitúgaši* in the first example because the *i* and raised-*i* characters have been given the same search weights. This Sort Order interface allows a user to define those equivalences. In the Order list box at the left, all the *i* characters have been grouped and ATP will assign them equal weights. Notice that the *i*-nasal characters have been distinguished and weighted separately. ATP also allows users to group strings with characters in case user fonts must use combinations of characters to produce a single phoneme. This also affords a technique for dealing with allophones. Again, ATP supports Truetype fonts but we do have routines that map those fonts onto Unicode equivalents when data is moved to XML or HTML.

[• Annotator Table for the Nakoda Project. Figure 10.] This is ATP’s Annotator interface, showing an annotator table for the Nakoda project. The form we were querying is visible near the bottom of the grid.

![Figure 10. Nakoda Project Annotator Table](image)

[• Annotator Table Configuration. Figure 11.] The Annotator Configuration interface lets a user define querying relations based on document models.
AISRI's multimedia dictionary database processor, the IDD. We have discussed IDD at several earlier gatherings and some of you may be familiar with it. It is a relational database application that supports the development of lexical research databases with full querying and regular-expressions capabilities. IDD supports the development of rich bilingual dictionaries and produces reports and camera-ready copy according to flexible, user-defined report forms. AISRI has six IDD databases in progress and IDD is now being used by several other projects. In the
last year we have worked out the means to deliver IDD data over the web and now we have five password-protected databases online. The IDD Reader in this image allows the user to query an IDD dictionary directly. Tools are also available for moving examples from ATP documents into IDD entries increasing the degree of integration between the two applications. In fact it should be possible to import an entire IDD dictionary into ATP in the near future for further export to XML.

[• A Model for Future Annotation Projects in Skiri Pawnee. Figure 13.] Up to this time our annotation projects have focused on literal glossing and morphosyntactic analysis but we are thinking ahead to further possibilities. Given ATP’s resources for data modeling, storage, and annotation, it should be clear that we can already support the most common kinds of annotation in large corpora: part-of-speech annotation, lemmatization, and syntactic parsing. Our approach will tend to take the form of stand-off annotation rather than the appended tagging schemes used by major corpus programs like the Spoken English Corpus or the CRATER corpus for Spanish. However, to truly be effective in those areas we need to provide generalized, automated annotating procedures. This requires the addition of an open-ended programming interface that lets users write scripts which can read from records conditionally, process their contents, and then write the processed output back to the same records or other designated records. We do plan to add such a scripting interface given time and resources. In the meantime it is also possible to write routines external to ATP that can operate on ATP tables in the same way.

Figure 13. Models for Future Annotation Work and for Metadata Headers

[• A Model for Metadata Information in Skiri Pawnee. Fig. 13] We think that we can derive solutions to problems of exposing metadata and web-services for inclusion in web-based data-banks like EMELD and OLAC directly from existing ATP resources. We would have to add headers to our project documents, as shown here, to satisfy the various harvesting organizations
that may interested in our corpora. We are already well-positioned to expose our grammatical class information from both our lexical projects and corpus projects to Linguist List’s Electronic Metastructure for Endangered Languages. We found the workshop readings by Lewis, Farrar, and Langendoen to be especially stimulating.

AISRI has become involved in several digital library initiatives at Indiana University over the last eighteen months, and have become familiar with notion of web-services. We have begun to develop web-based applications in PHP and we use MySQL and XQueryLite to serve up our data. We can provide access through exposed services that can be called from friendly applications elsewhere. We are intrigued by the ideas of WSDL (Web-Services Description Language) and UDDI (Universal Description, Discovery and Integration) and the semantic web, and we welcome the opportunity to participate in initiatives that are moving in those directions.

Thank you.