Recent Improvements to the ATLAS Architecture

Christophe Laprun, Jonathan Fiscus,

John Garofolo, Sylvain Pajot

National Institute of Standards and Technology
100 Bureau Drive Mail Stop 8940
Gaithersburg, MD 20899-8940
(+1) 301 975 3191

{claprun, jfiscus, jgarofolo, pajot}@nist.gov

ABSTRACT

We examine the recent improvements that were made to the ATLAS (Architecture and Tools for Linguistic Analysis Systems) architecture. We first introduce the architecture and the historical context for this work. Next, we describe NIST's initial implementation of the framework before analyzing it. We then focus on three important improvements (relating to multi-dimensional signals, hierarchical structures and validation) we have made to the architecture to make it more usable. We conclude by summarizing the major points covered and discuss plans for future work.

Keywords

ATLAS, MAIA, Linguistic infrastructure

1. INTRODUCTION

Annotated corpora are a central component of research in human language technology. As corpora have proliferated across languages, disciplines, and technologies, the lack of common exchange and storage formats has become a critical problem. This profusion of formats has made reusing annotated data or adapting existing tools for new annotation tasks significantly more difficult.

The standardization of tag sets (an approach we tried with our Universal Transcription Format [5]) is of moderate usefulness since language research is by necessity an open-ended task, subject to constant revision as the research domains change and the theories evolve.

A solution to this "bazaar of tools and formats" [2] is to interpose a generic annotation model via which annotation data is manipulated. ATLAS (Architecture and Tools for Linguistic Analysis Systems) makes use of such a generic data model. We first examine the historical context that led to the creation of the project. We then briefly describe the first implementation of the architecture, singling out three aspects of the architecture that needed to be improved: handling of complex signals and hierarchical structures and validation. Each of these aspects is then discussed in detail in subsequent sections. We conclude by summing up the major points we covered and suggest future work.

2. HISTORICAL CONTEXT

The ATLAS project started as a collaboration between the LDC, MITRE and NIST in 1999 following Bird and Liberman's seminal work on Annotation Graphs (AGs)[1] that demonstrated commonality across a diverse range of annotation practices and defined a formalism based on labeled, directed acyclic graphs.

The three parties recognized the urgent need for a consistent way to represent and process annotation data. NIST needed such a framework to accommodate constantly evolving needs in linguistic evaluation. The LDC was developing the AG formalism in order to develop an infrastructure that would help reduce the cost of linguistic annotation while MITRE was interested in extending their Alembic Workbench annotation tool to support new domains.

NIST recognized the importance of the LDC's work on AGs and decided to form a working group to explore the creation of a generic annotation framework and toolset that would address three important issues for the linguistic research community. First, ATLAS would promote language corpora reuse and exchange. By providing a generic annotation framework, ATLAS would make it easier to share data since data annotated with a generic representation could be reused in new contexts. Second,

reusable tools could be written in terms of the generic data model thus easing tool development. Finally, given its genericity, ATLAS would be able to gracefully accommodate changing domains and annotation schemes.

As the result of this collaboration, ATLAS Level 0 (based on AGs) and the basis for a generalized version (ATLAS Level 1) were formally introduced at the second international conference on Language Resources and Evaluation (LREC 2000) in Athens, Greece and were the subject of [2].

After LREC, the LDC moved on to implement an architecture based on Annotation Graphs (also called *AGs* or *ATLAS level 0*, [4]). NIST decided to pursue the development of the generalized version of ATLAS encompassing signals of arbitrary dimensions. A first implementation of the generalized framework (subsequently referred to as 'ATLAS') was made available in April 2001. Based on feedback gathered on this first implementation, a redesigned implementation was released, in Beta form, at the end of January 2002 followed by regular updates in the following months.

3. BASIC CONCEPTS

At the time of its introduction, only the premises for the generic framework were defined. The LDC had developed a prototype C++ implementation for ATLAS level 0. But, the implementation of the generic model had just begun. The remainder of this paper will address only NIST's implementation of the generalized version of ATLAS. We will discuss later how some of these components have evolved.

ATLAS provides an architecture targeted at facilitating the development of linguistic annotation applications[1]. When it was first introduced, the architecture was comprised of three main components that are still present in the newer implementation:

1. a generic linguistic annotation ontology,
2. an Application Programming Interface (API),
3. the ATLAS Interchange Format (AIF, [7]).

The ontology at its core provides the constructs on which the rest of the framework is built. The paradigm is simple but is surprisingly powerful when it comes to expressing complex annotation schemes. This provides a level of indirection separating physical storage and application logic that did not exist when most tools were written to directly read and write data using a specific format. Tools developed in terms of the ATLAS ontology can therefore work with any data conforming to ATLAS' paradigm, which is summarized as follows:

> An annotation is the fundamental act of associating some content to a region in a signal.

Paralleling this ontology, an ATLAS *Annotation* references a *Region* in a *Signal* and associates it with a *Content* element. The preceding notation defines a convention that we follow in this paper: ATLAS concepts are formatted using *this font and style*. *Annotation*s are organized in *Analysis* elements, which are themselves contained in *Corpus* elements. *Corpus* also manages *Region*s and *Anchor*s. *Children* elements will be discussed later. Figure 1, below, presents a pictorial view of the conceptual model.

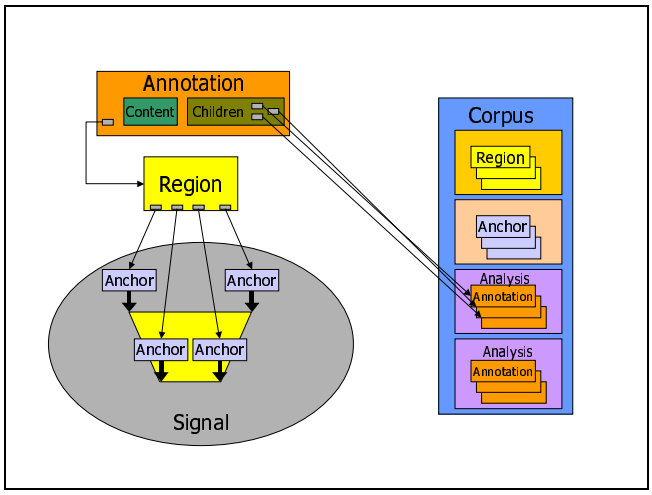

Figure 1: ATLAS Conceptual Model

These constructs can be implemented using a variety of programming languages. The ATLAS API defines ways to manipulate these constructs while the ATLAS Interchange Format provides a way to serialize ATLAS structures so that they can be reused and exchanged.

4. INITIAL IMPLEMENTATION

NIST's first implementation of the ATLAS architecture was made available in April 2001. This first implementation provided us with a starting point allowing us to gather feedback and from which we could mature the framework.

NIST's implementation (called jATLAS) provides a Java instantiation of these constructs in the form of objects that can be used to quickly develop linguistic applications.

Each of these objects publishes operations via which its data can be manipulated and behavior controlled. The ensemble of these operations defines the ATLAS API. Developers can thus access the components of the framework via a well-established interface, permitting alternate implementations to be developed without impacting client applications.

[1] Atlas was (in the Greek mythology) the Titan condemned to bear heavens upon his shoulders. ATLAS is supposed to bear the complexity of annotation management for the benefit of linguistic applications!

jATLAS can serialize the object structures it manipulates so that these structures can be easily shared and reused. The first implementation provided a single modality for serialization: the ATLAS Interchange Format. AIF is an XML-based file format optimized for the serialization of ATLAS structures. XML was chosen as the (then emergent but now well established) standard data interchange format.

jATLAS' first implementation provided NIST with a proof of concept as well as a full-scale model for experimentation. This implementation permitted us to create some initial tools that gave us a deeper understanding of the issues involved in the creation of a generic annotation infrastructure. The development of the framework was not, by its very nature, driven by a particular application, making it rather complex to make the right design choices and understanding their consequences before actually putting the architecture to use.

Both implementation and usage made us realize that the framework exhibited limitations of two kinds: *extrinsic* and *intrinsic*. *Extrinsic* limitations are problems that appear as part of the development process. *Intrinsic* limitations are problems that derive from the data model itself.

4.1 Extrinsic limitations

Extrinsic limitations revealed themselves as our development work progressed. Even though we realized at the time that some problems existed, we wanted to push ahead to explore the issues more deeply and create an architecture with which we could experiment.

The first limitation that we identified was related to compatibility with Annotation Graphs. ATLAS started as a conceptual framework based on the Annotation Graph formalism. Consequently, ATLAS' first instantiation was still very much influenced by the Annotation Graph formalism. Lots of effort had been made, during our collaboration with the LDC and MITRE, to ensure that the data model would be as isomorphic to AGs as possible. In particular, the expressiveness of the data model was restricted in order to respect the constraints imposed by the AG formalism. But, over time, it became clear to NIST that in order to fully generalize the data model, some of these constraints needed to be relaxed. In particular, we decided that strict conformance to the AG model was not required provided that ATLAS could express everything that could be expressed with AGs without loss of information.

The second extrinsic limit that we identified was that the data model was too strongly tied to the serialization format. The reason for this was that our early design work focused on the exchange format rather than the data model. The work on AIF was, in effect, driving the development of the data model, because, even though this was not a conscious effort, the AIF DTD was more or less used as the ATLAS data model. This restricted the expressive power of the framework to XML's.

These issues led us to re-examine the data model from a fresh perspective. We re-designed it starting with the core ontology and only adding new constructs as deemed necessary. The serialization format became then an outgrowth of the stabilized data model.

4.2 Intrinsic limitations

Extrinsic limitations aside, several limitations also existed with the data model itself. These were, however, more difficult to identify and appeared only after we put the architecture to work in building applications.

The first limitation that we will examine was the need to develop the support of complex signal types.

The next limitation, we will then turn our attention to, was the creation of hierarchical annotation structures. Although ATLAS' first data model allowed the creation of hierarchical annotations, it was not as intuitive as it could have been.

The last limitation was the management of semantic information. We initially considered deferring semantic validation to the XML layer. However, this proved to be insufficient and we decided to introduce a meta-annotation concept, implemented in ATLAS by the Meta-Annotation Infrastructure for ATLAS (MAIA), to efficiently address semantic issues.

We discuss each of these points in turn in the next sections and present the solutions that were implemented in the redesign version of ATLAS.

5. MULTI-DIMENSIONAL SIGNALS

Traditionally, linguistic resources have focused on a linear class of signals (e.g., text or audio) that can be indexed via a simple offset into a file. However, as technical capabilities and processing power have increased, so has interest in multi-modality and signals that cannot be reduced to a single dimension.

From the beginning, NIST and the ATLAS Working Group recognized the need for a framework that would be able to evolve gracefully as research interests moved toward more complex, multi-modal signal types. Initially, we believed that the AG formalism could be employed for all signal types. However, as we began working with AGs, it became apparent that although they were well-suited to linear signals, they did not scale well for more complex signal types. The basic premise of multi-dimensional signal handling was included in the first ATLAS implementation. However, many details were yet to be worked out

5.1 Signals and SignalGroups

Annotation sources are represented in ATLAS by the *Signal* construct. An ATLAS *Signal* is an immutable, N-dimensional space containing phenomena that are the target of *Annotation*s. Even though typical *Signal*s can be equated to physical signal files (speech waveforms, newswire text, video or other more complex data with

higher dimensionality), it is not a necessity. In ATLAS, a *Signal* is an entity that identifies a logical (as opposed to a physical file) target for *Annotation*s and can thus refer, for example, only to the left track of a stereo recording. ATLAS also does not prescribe to any single format or dimensionality for physical signals, but there must be a way to define an unambiguous coordinate system for the *Signal*.

Furthermore, groups of related *Signal*s can be formed to create targets for *Annotation*s spanning several logical signals. The *SignalGroup* construct is used to model this grouping. *SignalGroup*s constitute new logical signals themselves and are treated as such by the framework. However, some constraints need to be imposed to make them manageable. In particular, since ATLAS *Signal*s need to be immutable, once a *SignalGroup* is created and referenced, it cannot be changed. Moreover, in order to ensure consistency, only *Signal*s that share common dimensions can be grouped together. An obvious use of *SignalGroup* is the alignment of different signals along a given time line. An example of an index into a stereo audio signal is a simple example.

5.2 Regions and Anchors

Once *Signal*s have been identified, linguistic phenomena of interest are identified via *Region* constructs. A *Region* is an abstraction for identifying an area of the *Signal* space. *Region*s are delimited by a set of coordinates that mark specific areas of interest. These markers are modeled by the *Anchor* construct, thus named because they are used to "anchor" annotations to *Signal*s.

*Anchor*s are the only ties that annotations have to the physical structure of the signal and the only ATLAS concept that is signal-specific. *Region*s use as many *Anchor*s as needed to index into *Signal*s. Thus, the *Region* construct encapsulates the specificity of the underlying signal. This is a particularly important aspect of ATLAS since it allows the framework to evolve and scale gracefully, when confronted with new classes of signals, without requiring change to the basic ontology. It is also worth noting that *Region*s can reference other *Region*s as well.

Suppose, for example, that a given television signal has been modeled in ATLAS using three *Signal*s: one for each of the two audio tracks and one for the video signal. One might want to annotate a phenomenon occurring only on the audio part of the television signal. If the phenomenon of interest spans both audio channels, one could create a *SignalGroup* grouping them together. Since the video frames can be projected along the time dimension, a grouping of the three *Signal*s can be made to annotate time-based phenomena for the entire television signal. However, if the annotation task is to track hand movements, the *Region*s of interest could be defined using both a temporal *Anchor* to mark the frames of interest and another *Region* to define the area of interest within each video frame. The complete *Region* of interest would then be the composition of the temporal *Anchor*s and 2D *Region*s. Figure 2, below, presents a graphical view of this example.

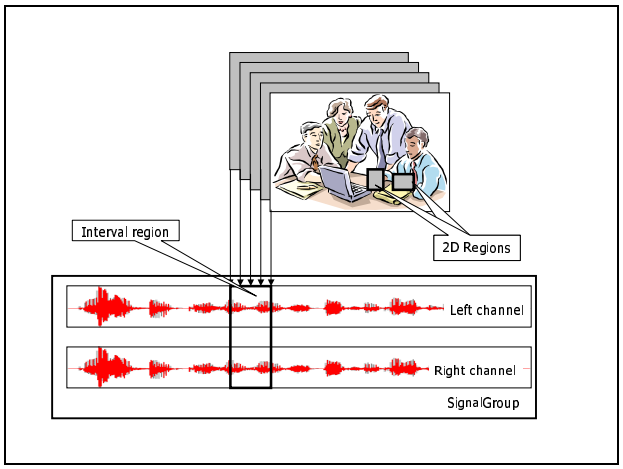

Figure 2: Video Annotation Example

6. HIERARCHICAL STRUCTURES

Support for the representation of inter-annotation relations (in particular, annotation hierarchies) was somewhat limited in the first jATLAS' implementation that constrained such relationships to be represented as part of an *Annotation*'s *Content* in a way that was neither intuitive nor explicit.

As a result of the experience we gained with this implementation, we introduced the *Children* construct to model relations between *Annotation*s. *Children* constructs maintain a list of references to *Annotation*s that are descendants of a parent *Annotation*. Since references (rather than actual annotations) are used, it is possible to build overlapping hierarchies. Reuse is therefore improved since already created *Annotation*s can be reused in new contexts.

Parent *Annotation*s can also use their *Children* elements to derive their *Content* and/or *Region*. It is therefore possible to dynamically update *Content* and/or *Region* information of a parent based on changes to its set of *Children*, thus avoiding redundancy and simplifying annotation management.

The TIMIT Acoustic Phonetic Corpus ([3]) provides a simple example of parent/child annotation relationships. In TIMIT, words are composed of a set of phones. This relationship is modeled quite simply in ATLAS by creating a *Children* element of a word *Annotation* that contains a list of references to the phone *Annotation*s for that word. Further, this makes it possible for TIMIT word *Annotation*s to be reused in the context of morphological analysis – thus creating overlapping yet separate hierarchies. Moreover, since a sentence in TIMIT could be defined as the set of its subordinate words, a sentence *Annotation* could be created by deriving its *Region* (and

Content) from the union of the *Region*s referenced by its subordinate word *Annotation*s.

7. META-ANNOTATION INFRASTRUCTURE FOR ATLAS

7.1 Motivation

ATLAS defines a very generic data model that is designed to be able to model a wide range of annotation tasks. Because of this genericity, ATLAS' constructs are minimally constrained. ATLAS provides enough expressive power to represent extremely complex annotations. However, this capability did not come, in the first implementation, without a significant development cost since the responsibility for such issues as structural integrity and consistency checking were pushed up to the application level. To unburden application developers, a means of constraining ATLAS constructs was therefore needed so that ATLAS applications could interpret a particular *Annotation* as a word, for example. The Meta-Annotation concept was introduced to address these needs.

A Meta-Annotation is a piece of meta-information about a kind of linguistic annotation. In the context of ATLAS, meta-annotations are concretized in the form of the *ATLASType* construct. The Meta-Annotation Infrastructure for ATLAS (MAIA[2], [9]) implements the Meta-Annotation concept for ATLAS. It provides a scheme language that allows type definitions to be declared using a simple, XML-based syntax. The ATLAS framework can then dynamically interpret these type definitions. MAIA also provides services (such as the loading and saving of types) that can be utilized by ATLAS implementations.

7.2 ATLASTypes

A rudimentary placeholder existed in the first jATLAS implementation for types. However, the implementation of types and type-checking was left completely up to application developers. We realized, however, when we started working with the framework that this required a considerable amount of work and resulted in a lot of code duplication.

We also explored the idea of deferring semantic validation to the XML layer. However, this provided only limited validation. Worse, it necessitated that validation could only occur on XML file read/writes and could not be used for internal operations.

We realized that to be maximally useful, ATLAS applications need to be able to automatically interpret type information without requiring user intervention or developer effort. Further, we realized that such information more properly belonged at the corpus definition layer than in the application layer. We decided, therefore, to re-design the framework to incorporate support for data typing. This was accomplished via the addition of MAIA and *ATLASType*s to ATLAS.

An *ATLASType* is a piece of metadata associated with an ATLAS construct to describe attributes of, and permitted operations for a specific annotation element. This provides the kind of data typing supported in most object-oriented languages. *ATLASType*s are thus very similar to classes in object-oriented parlance.

By definition, elements with the same *ATLASType* share the same structure. *ATLASType*s also enforce constraints on relationships between ATLAS constructs. For example, from our TIMIT example above, we want to require that sentence annotations can contain only word annotations and, word annotations, in turn, can contain only phone annotations.

7.3 MAIA

MAIA provides a mechanism for the creation and management of *ATLASType*s. Its type definition language provides a formalism for the specification of an annotation corpus which can be used to validate operations during the creation and modification of the corpus. It permits the ATLAS framework (and ATLAS-based applications) to perform validation to ensure that elements that are supposed to be of a given type have the correct structure and behave as expected. The MAIA type definition language provides a mechanism to create a self-documenting, concise definition of a corpus usable by both human designers/users and ATLAS-based tools.

Essentially, MAIA adds a semantic layer on top of ATLAS' generic structures. This enables developers to focus on higher-level issues, such as user-interaction, without the burden of having to attend to low-level data management.

We see MAIA as a major step towards the development of more generic tools that can be tailored to specific needs at the data level, rather than at the application level. For example, a generic annotation editor could be created which dynamically builds specific interfaces based on the MAIA definition of the data it is to work with.

MAIA is clearly a work in progress and we already have several ideas about how it can be made more powerful and expressive. Currently, it supports only basic data type and position constraints. However, it would be more useful if it supported relationship- and content-dependent constraints. For example, in TIMIT, the time for the first word of a sentence should align with the sentence boundary and we should be able to define what constitutes a legal word. In particular, MAIA will eventually support more elaborate typing for value and range constraints and more accurate description of inter-annotation dependencies. Our ultimate goal is to make MAIA so comprehensive that application developers will have to write no corpus-specific code. MAIA will be detailed in a forthcoming paper.

[2] Maia was one of Atlas' daughter in the Greek mythology.

8. CONCLUSION

Since it was first introduced at LREC 2000, the ATLAS framework has evolved to incorporate numerous enhancements including support for hierarchical relationships, multi-dimensional signal types, and data typing via MAIA.

The Java instantiation of ATLAS (jATLAS [8]) has been updated with these enhancements and is now currently in Beta version, available for download, along with more information on the architecture, on the ATLAS web site [6].

Although, ATLAS has matured into a powerful and very usable annotation framework, we are still working to improve it. In particular, we are investigating extending the current framework into a full-fledged annotation server that would allow multiple users to concurrently work on the same annotations in a fully distributed annotation environment. We are also working to make MAIA more expressive by tying it to a query language we intend to develop for ATLAS – thus providing a complete annotation data development/research environment. Work is also in progress on defining a widget library for visualization and editing of ATLAS elements to further improve the ease of application development.

At this time, we would like to invite people interested in using ATLAS to send us descriptions of their annotation corpora. We will work with them to develop MAIA definitions for their data and build a sample ATLAS version of their corpus. This will also provide us with a diversity of data so that we can further improve the framework.

9. REFERENCES

[1] Bird, S. and Liberman, M., 1999. A formal framework for linguistic annotation. Technical report MS-CIS-99-01, Department of Computer and Information Science, University of Pennsylvania. Revised version appeared in Speech Communications 33 (1,2), pp 23-60.

[2] Bird, S., Day, D., Garofolo, J., Henderson, J., Laprun C. and Liberman, M., 2000. ATLAS: A flexible and extensible architecture for linguistic annotation in Proceedings of LREC 2000 (Athens, Greece, May 2000), pp 1699-1706.

[3] Garofolo, J., Lamel, L., Fisher, W., Fiscus, J., Pallett, D., Dahlgren, N., NIST, 1986. The DARPA TIMIT acoustic-phonetic continuous speech corpus CDROM. [http://www.ldc.upenn.edu/lol/docs/TIMIT.html]

[4] LDC, 1999. Annotation Graphs. [http://www.ldc.upenn.edu/AG/]

[5] NIST, 1998. A Universal Transcription Format (UTF) annotation specification for evaluation of spoken language technology corpora. [http://www.nist.gov/speech/tests/bnr/hub4_98/utf-1.0-v2.ps]

[6] NIST, 1999. Architecture and Tools for Linguistic Analysis Systems. [http://www.nist.gov/speech/atlas/]

[7] NIST, 1999. ATLAS Interchange Format. [http://www.nist.gov/speech/atlas/develop/aif.html]

[8] NIST, 2000. jATLAS, a Java implementation of the ATLAS framework. [http://www.nist.gov/speech/atlas/jatlas/]

[9] NIST, 2002. Meta-Annotation Infrastructure for ATLAS. [http://www.nist.gov/speech/atlas/develop/maia.html]