BUILDING AN ARCHITECTURE: A CAWG SAGA

Ralph Grishman
Computer Science Department
New York University
715 Broadway, 7th Floor
New York, NY 10003
grishman@cs.nyu.edu

Abstract

The Tipster Architecture — a standardized interface for providing document management, document retrieval, and information extraction services — is one of the major products of Phase II of the Tipster program. The architecture was developed by the Contractors' Architecture Working Group (CAWG) over the past two years. It has been refined through feedback from the demos developed by the CAWG for the 6-month and 12-month Tipster meetings, and from the Tipster-compliant systems now being implemented.

MOTIVATION

Phase II of the Tipster program had a twofold mission: to advance the technology for document detection (information retrieval and routing) and information extraction from free text, and also to facilitate the delivery of this technology to Government customers.

How could delivery of this technology be improved? The Government proposed an approach of identifying a core set of services needed by a broad range of text analysis applications, and defining a standard set of functions and interfaces for these services. These functions and interfaces would constitute an architecture — the Tipster Architecture.

Creation of a standardized architecture could help in many ways. It could speed initial application development by providing standardized, pre-existing modules. It could ease system maintenance through well-defined, well-document internal interfaces between modules. It could support system upgrading to take advantage of improved technology, by making it much easier to replace an older module with a newer (and hopefully better-performing) module. And by making it easier to combine language analysis modules in novel ways (for example, combining extraction and detection) it could enhance system performance.

In addition, the architecture would make it easier to conduct research on individual components of a text analysis system, by creating an environment of standardized modules. The lone experimenter would not need to create an entire application system from scratch to conduct experiments.

Central to all these benefits would be the notion of *plug and play*: defining a set of modules and interfaces with sufficient precision that we could unplug one vendor's module and replace it with another vendor's without affecting system functionality.

THE SAGA

The Launch

The objective was certainly quite ambitious: to create an architecture which could satisfy the needs of a wide range of text analysis applications and could be implemented efficiently enough to support operational systems. To meet this challenge, the Government assembled a group of representatives of the contractors who would be conducting research and development under Tipster Phase II, and told us we had six months to create an architecture. By including representatives of the contractors who would have to *use* the architecture (during the rest of Phase II), the Government sought to insure that all the essential needs of various detection and extraction applications would be addressed.[1]

The prospect was challenging and indeed a bit daunting for a group none of whom had written an

[1]The members of the committee have included, over the course of Phase II, Bill Caid, Jamie Callan, Jim Conley, Hal Corbin, Jim Cowie, Kathy DiBella, Ted Dunning, Joe Dzikiewicz, Louise Guthrie, Jerry Hobbs, Clint Hyde, Marc Ilgen, Paul Jacobs, Matt Mettler, Bill Ogden, Peggy Otsubo, Bev Schwartz, Ira Sider, Ralph Weischedel, and Remi Zajac; Ralph Grishman chaired the committee.

architecture before. No one was sure quite what the final document would look like. Fortunately, a series of planning workshops for Phase II, held in the spring of 1993, had identified some basic concepts for an architecture, and provided a strong base of ideas from which to start.

Central among these ideas was the notion of an *annotated document.* All of the information which was learned about a document in the course of its analysis — header zones, paragraph and sentence boundaries, person and organization names in the text, relational information about selected types of events, comments by human analysts, etc. — would be stored as *annotations* which would be kept with the document. The annotations would point to segments of the original text; the original text would be maintained unchanged.

Starting from these ideas, the CAWG set out in April, 1994 to knit together an architecture. Over the spring and summer of 1994 the outlines of an architecture began to come together in the form of an Architecture Design Document. The CAWG initially met once a month in Washington; in the intervening four weeks, the committee chair prepared a revised Design Document to reflect the changes and additions proposed at the previous meeting.

The architecture was specified in terms of a hierarchy of object classes, with operations associated with each class (and inheriting operations from classes above it in the hierarchy). The initial specification was programming language independent, but included some basic guidelines for implementations in C and Lisp.

The principal object classes included the *document* and the *annotation*, which were mentioned before. Documents were organized into *collections.* In addition to annotations, which provided information about segments of a document, a document could have *attributes* which specified information about the entire document (e.g., its source or creation date). Information about an entire collection could be recorded as attributes on the collection.

Modules within the Architecture Communicate primarily by passing documents and collections, and by adding annotations and attributes. Thus a "name annotator" would take a collection and add to each document in the collection some annotations indicating the proper names found in that document. An information extraction module would add annotations corresponding to instances of a type of event. A retrieval engine would take a collection and a query and return a (smaller) collection of relevant documents, with an attribute on each document indicating its degree of relevance to the query.

A number of specialized object types had to be added for retrieval and routing, including different types of queries and indexes for document and query collections.

The Demos

To show the benefits of the architecture, and to push its further development, the Government asked the CAWG to create an integrated demo system for the 6-month Tipster meeting in November 1994. New Mexico State Univ. created a "Document Manager" — an implementation of the core document management functions in accordance with the architecture. Several other contractors[2] provided detection and extraction components which conformed to the architecture and interfaced to this document manager. This demo provided the first, albeit limited, demonstration of "plug and play", and the first demonstration of detection and extraction systems interacting through annotated documents.

This demo led to several revisions to the architecture.[3] In addition, its success led to the design of an even more ambitious demo for the 12-month Tipster meeting in May 1995. This second demo integrated several extraction systems, several detection systems and a richer set of interfaces[4]. The 12-month demo, in turn, propelled further developments in the architecture, including methods for declaring annotations and for representing information extraction templates as annotations.

A few more changes were made to the architecture document as a result of problems which came up during the 12-month demo. In particular, an explicit specification of the C-language interface was added to the document.

On the Open Sea

By late 1995, several Government systems were being implemented in conformance with the Tipster Architecture. Contractors found that they had to extend or modify the architecture to meet the needs of specific applications. These modifications and extensions are now beginning to be presented to the CAWG and to the Government's Architecture Committee for possible revision of the standard architecture.

[2]BBN, HNC, Martin-Marietta, and the Univ. of Massachusetts

[3]Most contentiously, the removal of "document lists", which were lists of pointers to documents in collections; all sets of documents were now stored uniformly as collections.

[4]including contributions from BBN, HNC, Lockheed-Martin, New Mexico State Univ., SRI, TRW, and the Univ. of Massachusetts

A LIVING ARCHITECTURE

The Government originally hoped that an architecture could be completed in six months, and that we could then all go back to doing research and writing systems in conformance with the architecture. It hasn't quite worked out that way. We did have *an* architecture after six months, so we might have stopped there and let the Government mandate that people use that architecture as best they could. However, the Government had an ambitious set of goals, and we were still a long way from meeting all of them. In addition, it was recognized that real success for the architecture lay in making it attractive enough, in both design and availability of components, to be widely and voluntarily adopted. So we have embarked on a program of gradual improvement of the architecture. This is comparable to many efforts at programming language standardization, where an initial specification is gradually revised over several years in response to user and developer feedback.

After two years, we are closer to meeting the goals set for the architecture, but we are still not done. We expect that there will be continuing incremental revisions to the architecture driven by the need for efficiency, completeness, precision, and simplicity:

- **efficiency**: we have tried to minimize the loss of efficiency due to conformance to the architecture. We recognize that there will be some cost in conformance, since we are using general mechanisms in place of ones specially developed for a single application, but we need to insure that these costs are not so great that they make the architecture unattractive. In particular, we have been looking recently at the problems of retrieval, to insure that the operations associated with creating a new collection of documents for the results of a retrieval operation can be performed quickly.

- **completeness**: in some sense the architecture will never be "complete": there will always be requests to standardize additional services or resources associated with text analysis. However, there are some areas which are clearly lacking and are needed by many applications. The architecture as yet makes no special provision for operation in a multi-process environment; we need to include such mechanisms as read and write locks and transaction control which are typical of data base systems. We need to specify the error conditions for the operations in the architecture and the mechanisms for error signaling. The details of the interface have as yet been fully spelled out only for C; based on the implementations which are planned, we need to add explicit specifications for C++, CORBA, and Common Lisp.

- **precision**: We recognized all along that the relatively brief English prose describing the operations in the architecture left some things underspecified. Nonetheless, it was a sobering experience in the fall of 1995 when for the first time a contractor who had not been part of the CAWG (and hence was not privy to the "oral tradition" of the CAWG) prepared a design for a Tipster-compliant application, and interpreted the Design Document in some unexpected ways. This experience has led to an extended discussion of the semantics of some operations, and plans to include both more details and some examples in the Design Document.

  It has also prompted the development of an initial version of a *validation suite* by New Mexico State Univ. Until now, conformance to the Tipster Architecture has been gauged by a manual comparison of an implementation with the Design Document. The validation suite implements a series of tests, each of which tests some aspect of architecture compliance. It thus will allow the verification of architecture compliance to be partially automated. It will also provide a definition of some aspects of the architecture which can complement the prose descriptions in the Design Document.

- **simplicity**: We have made a strong effort from the beginning to assemble an architecture from a small number of simple yet powerful objects, and where necessary to rethink parts of the design rather than simply adding stuff on. As we try to make the architecture more comprehensive, there will be a natural tendency to just add features and operations. Unfortunately this makes the architecture harder to understand and harder to implement. We therefore will need to maintain our vigilance in protecting and enhancing the clarity of the design.

Our hope is that, by continuing to be responsive to the needs of the users — the designers and implementers of text analysis systems — in developing the architecture, we can encourage the creation of a wide variety of Tipster-compliant modules, available as COTS (commercial, off-the-shelf) products. Through this "marketplace of Tipster modules" we will be able to meet our goal of facilitating the transition of advanced text analysis techniques to operational systems.