A flexible distributed architecture for NLP system development and use

Freddy Y. Y. Choi
Artificial Intelligence Group
University of Manchester
Manchester, U.K.
choif@cs.man.ac.uk

Abstract

We describe a distributed, modular architecture for platform independent natural language systems. It features automatic interface generation and self-organization. Adaptive (and non-adaptive) voting mechanisms are used for integrating discrete modules. The architecture is suitable for rapid prototyping and product delivery.

1 Introduction

This article describes TEA[1], a flexible architecture for developing and delivering platform independent text engineering (TE) systems. TEA provides a generalized framework for organizing and applying reusable TE components (e.g. tokenizer, stemmer). Thus, developers are able to focus on problem solving rather than implementation. For product delivery, the end user receives an exact copy of the developer's edition. The visibility of configurable options (different levels of detail) is adjustable along a simple gradient via the automatically generated user interface (Edwards, Forthcoming).

Our target application is telegraphic text compression (Choi (1999b); of Roelofs (Forthcoming); Grefenstette (1998)). We aim to improve the efficiency of screen readers for the visually disabled by removing uninformative words (e.g. determiners) in text documents. This produces a stream of topic cues for rapid skimming. The information value of each word is to be estimated based on an unusually wide range of linguistic information.

TEA was designed to be a development environment for this work. However, the target application has led us to produce an interesting architecture and techniques that are more generally applicable, and it is these which we will focus on in this paper.

[1]TEA is an acronym for Text Engineering Architecture.

2 Architecture

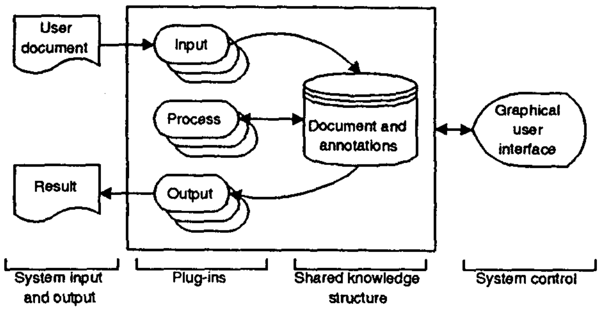

Figure 1: An overview of the TEA system framework.

The central component of TEA is a frame-based data model (F) (see Fig.2). In this model, a document is a list of frames (Rich and Knight, 1991) for recording the properties about each token in the text (example in Fig.2). A typical TE system converts a document into F with an input plug-in. The information required at the output determines the set of process plug-ins to activate. These use the information in F to add annotations to F. Their dependencies are automatically resolved by TEA. System behavior is controlled by adjusting the configurable parameters.

Frame 1: (:token An :pos art :begin_s 1)
Frame 2: (:token example :pos n)
Frame 3: (:token sentence :pos n)
Frame 4: (:token . :pos punc :end_s 1)

Figure 2: "An example sentence." in a frame-based data model

This type of architecture has been implemented, classically, as a 'blackboard' system such as Hearsay-II (Erman, 1980), where inter-module communication takes place through a shared knowledge structure; or as a 'message-passing' system where the modules communicate directly. Our architecture is similar to blackboard systems. However, the purpose of F (the shared knowledge structure in TEA) is to provide a single extendable data structure for annotating text. It also defines a standard interface for inter-module communication, thus, improves system integration and ease of software reuse.

2.1 Voting mechanism

A feature that distinguishes TEA from similar systems is its use of voting mechanisms for system integration. Our approach has two distinct but uniformly treated applications. First, for any type of language analysis, different techniques t_i will return successful results $P(r)$ on different subsets of the problem space. Thus combining the outputs $P(r|t_i)$ from several t_i should give a result more accurate than any one in isolation. This has been demonstrated in several systems (e.g. Choi (1999a); van Halteren et al. (1998); Brill and Wu (1998); Veronis and Ide (1991)). Our architecture currently offers two types of voting mechanisms: weighted average (Eq.1) and weighted maximum (Eq.2). A Bayesian classifier (Weiss and Kulikowski, 1991) based weight estimation algorithm (Eq.3) is included for constructing adaptive voting mechanisms.

$$P(r) = \sum_{i=1}^{n} w_i P(r|t_i) \tag{1}$$

$$P(r) = \max\{w_1 P(r|t_1), \ldots, w_n P(r|t_n)\} \tag{2}$$

$$w_i = P(t_i | P(r|t_1), \ldots, P(r|t_n)) \tag{3}$$

Second, different types of analysis a_i will provide different information about a problem, hence, a solution is improved by combining several a_i. For telegraphic text compression, we estimate $E(w)$, the information value of a word, based on a wide range of different information sources (Fig.2.1 shows a subset of our working system). The output of each a_i are combined by a voting mechanism to form a single measure.

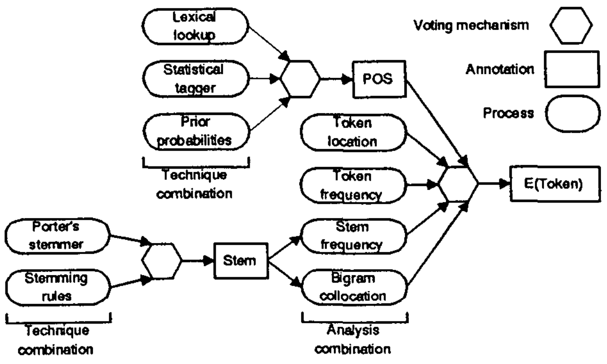

Figure 3: An example configuration of TEA for telegraphic text compression.

Thus, for example, if our system encounters the phrase 'President Clinton', both lexical lookup and automatic tagging will agree that 'President' is a noun. Nouns are generally informative, so should be retained in the compressed output text. However, grammar-based syntactic analysis gives a lower weighting to the first noun of a noun-noun construction, and bigram analysis tells us that 'President Clinton' is a common word pair. These two modules overrule the simple POS value, and 'President Clinton' is reduced to 'Clinton'.

3 Related work

Current trends in the development of reusable TE tools are best represented by the Edinburgh tools (LTGT)[2] (LTG, 1999) and GATE[3] (Cunningham et al., 1995). Like TEA, both LTGT and GATE are frameworks for TE.

LTGT adopts the pipeline architecture for module integration. For processing, a text document is converted into SGML format. Processing modules are then applied to the SGML file sequentially. Annotations are accumulated as mark-up tags in the text. The architecture is simple to understand, robust and future proof. The SGML/XML standard is well developed and supported by the community. This improves the reusability of the tools. However,

[2]LTGT is an acronym for the Edinburgh Language Technology Group Tools

[3]GATE is an acronym for General Architecture for Text Engineering.

the architecture encourages tool development rather than reuse of existing TE components.

GATE is based on an object-oriented data model (similar to the TIPSTER architecture (Grishman, 1997)). Modules communicate by reading and writing information to and from a central database. Unlike LTGT, both GATE and TEA are designed to encourage software reuse. Existing TE tools are easily incorporated with Tcl wrapper scripts and Java interfaces, respectively.

Features that distinguish LTGT, GATE and TEA are the configuration methods, portability and motivation. Users of LTGT write shell scripts to define a system (as a chain of LTGT components). With GATE, a system is constructed manually by wiring TE components together using the graphical interface. TEA assumes the user knows nothing but the available input and required output. The appropriate set of plug-ins are automatically activated. Module selection can be manually configured by adjusting the parameters of the voting mechanisms. This ensures a TE system is accessible to complete novices and yet has sufficient control for developers.

LTGT and GATE are both open-source C applications. They can be recompiled for many platforms. TEA is a Java application. It can run directly (without compilation) on any Java supported systems. However, applications constructed with the current release of GATE and TEA are less portable than those produced with LTGT. GATE and TEA encourage reuse of existing components, not all of which are platform independent[4]. We believe this is a worth while trade off since it allows developers to construct prototypes with components that are only available as separate applications. Native tools can be developed incrementally.

4 An example

Our application is telegraphic text compression. The examples were generated with a subset of our working system using a section of the book *HAL's legacy* (Stork, 1997) as test data. First, we use different compression techniques to generate the examples in Fig.4. This was done by simply adjusting a parameter of an output plug-in. It is clear that the output is inadequate for rapid text skimming. To improve the system, the three measures were combine with an unweighted voting mechanism. Fig.4 presents two levels of compression using the new measure.

[4]This is not a problem for LTGT since the architecture does not encourage component reuse.

1. With science fiction films the more science you understand the less you admire the film or respect its makers
2. fiction films understand less admire respect makers
3. fiction understand less admire respect makers
4. science fiction films science film makers

Figure 4: Three measures of information value: (1) Original sentence, (2) Token frequency, (3) Stem frequency and (4) POS.

1. science fiction films understand less admire film respect makers
2. fiction makers

Figure 5: Improving telegraphic text compression by analysis combination.

5 Conclusions and future directions

We have described an interesting architecture (TEA) for developing platform independent text engineering applications. Product delivery, configuration and development are made simple by the self-organizing architecture and variable interface. The use of voting mechanisms for integrating discrete modules is original. Its motivation is well supported.

The current implementation of TEA is geared towards token analysis. We plan to extend the data model to cater for structural annotations. The tool set for TEA is constantly being extended, recent additions include a prototype symbolic classifier, shallow parser (Choi, Forthcoming), sentence segmentation algorithm (Reynar and Ratnaparkhi, 1997) and a POS tagger (Ratnaparkhi, 1996). Other adaptive voting mechanisms are to be investigated. Future release of TEA will support concurrent execution (distributed processing) over a network. Finally, we plan to investigate means of improving system integration and module organization, e.g. annotation, module and tag set compatibility.

References

E. Brill and J. Wu. 1998. Classifier combination for improved lexical disambiguation. In *Proceedings of COLING-ACL'98*, pages 191–195, Montreal, Canada, August.

F. Choi. 1999a. An adaptive voting mechanism for improving the reliability of natural language processing systems. Paper submitted to EACL'99, January.

F. Choi. 1999b. Speed reading for the visually disabled. Paper submitted to SIGART/AAAI'99 Doctoral Consortium, February.

F. Choi. Forthcoming. A probabilistic approach to learning shallow linguistic patterns. In *Proceedings of ECAI'99 (Student Session)*, Greece.

H. Cunningham, R.G. Gaizauskas, and Y. Wilks. 1995. A general architecture for text engineering (gate) – a new approach to language engineering research and development. Technical Report CD-95-21, Department of Computer Science, University of Sheffield. http://xxx.lanl.gov/ps/cmp-lg/9601009.

M. Edwards. Forthcoming. An approach to automatic interface generation. Final year project report, Department of Computer Science, University of Manchester, Manchester, England.

L. Erman. 1980. The hearsay-ii speech understanding system: Integrating knowledge to resolve uncertainty. In *ACM Computer Surveys*, volume 12.

G. Grefenstette. 1998. Producing intelligent telegraphic text reduction to provide an audio scanning service for the blind. In *AAAI'98 Workshop on Intelligent Text Summarization*, San Francisco, March.

R. Grishman. 1997. Tipster architecture design document version 2.3. Technical report, DARPA. http://www.tipster.org.

LTG. 1999. Edinburgh university, hcrc, ltg software. WWW. http://www.ltg.ed.ac.uk/software/index.html.

H. Rollfs of Roelofs. Forthcoming. Telegraphese: Converting text into telegram style. Master's thesis, Department of Computer Science, University of Manchester, Manchester, England.

G. M. P. O'Hare and N. R. Jennings, editors. 1996. *Foundations of Distributed Artificial Intelligence*. Sixth generation computer series. Wiley Interscience Publishers, New York. ISBN 0-471-00675.

A. Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of the empirical methods in NLP conference*, University of Pennsylvania.

J. Reynar and A. Ratnaparkhi. 1997. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the fifth conference on Applied NLP*, Washington D.C.

E. Rich and K. Knight. 1991. *Artificial Intelligence*. McGraw-Hill, Inc., second edition. ISBN 0-07-100894-2.

D. Stork, editor. 1997. *Hal's Legacy: 2001's Computer in Dream and Reality*. MIT Press. http://mitpress.mit.edu/e-books/Hal/.

H. van Halteren, J. Zavrel, and W. Daelemans. 1998. Improving data driven wordclass tagging by system combination. In *Proceedings of COLING-ACL'98*, volume 1.

J. Veronis and N. Ide. 1991. An accessment of semantic information automatically extracted from machine readable dictionaries. In *Proceedings of EACL'91*, pages 227–232, Berlin.

S. Weiss and C. Kulikowski. 1991. *Computer Systems That Learn*. Morgan Kaufmann.