Pacific Symposium on Biocomputing 13:616-627(2008)

FILLING THE GAPS BETWEEN TOOLS AND USERS: A TOOL COMPARATOR, USING PROTEIN-PROTEIN INTERACTION AS AN EXAMPLE

YOSHINOBU KANO[1], NGAN NGUYEN[1], RUNE SÆTRE[1], KAZUHIRO YOSHIDA[1], YUSUKE MIYAO[1], YOSHIMASA TSURUOKA[3], YUICHIRO MATSUBAYASHI[1], SOPHIA ANANIADOU[2,3], JUN'ICHI TSUJII[1,2,3]

[1]*Department of Computer Science, University of Tokyo*
Hongo 7-3-1, Bunkyo-ku, Tokyo 113-0033 Tokyo

[2]*School of Computer Science, University of Manchester*
PO Box 88, Sackville St, MANCHESTER M60 1QD, UK

[3]*NaCTeM (National Centre for Text Mining), Manchester Interdisciplinary Biocentre, University of Manchester, 131 Princess St, MANCHESTER M1 7DN, UK*

Recently, several text mining programs have reached a near-practical level of performance. Some systems are already being used by biologists and database curators. However, it has also been recognized that current Natural Language Processing (NLP) and Text Mining (TM) technology is not easy to deploy, since research groups tend to develop systems that cater specifically to their own requirements. One of the major reasons for the difficulty of deployment of NLP/TM technology is that re-usability and interoperability of software tools are typically not considered during development. While some effort has been invested in making interoperable NLP/TM toolkits, the developers of end-to-end systems still often struggle to reuse NLP/TM tools, and often opt to develop similar programs from scratch instead. This is particularly the case in BioNLP, since the requirements of biologists are so diverse that NLP tools have to be adapted and re-organized in a much more extensive manner than was originally expected. Although generic frameworks like UIMA (Unstructured Information Management Architecture) provide promising ways to solve this problem, the solution that they provide is only partial. In order for truly interoperable toolkits to become a reality, we also need sharable type systems and a developer-friendly environment for software integration that includes functionality for systematic comparisons of available tools, a simple I/O interface, and visualization tools. In this paper, we describe such an environment that was developed based on UIMA, and we show its feasibility through our experience in developing a protein-protein interaction (PPI) extraction system.

1. Introduction

In the biomedical domain, an increasing number of Text Mining (TM) and Natural Language Processing (NLP) tools, including part-of-speech (POS) taggers [1], named entity recognizers (NERs) [10], protein name normalizers [2], syntactic parsers [3,4], and relation or event extractors (ERs) have been developed, and some of them are now ready for biologists and database curators

to use for their own purposes [5]. However, it is still very difficult to integrate independently developed tools into an aggregated application that achieves a specific task. The difficulties are caused not only by differences in programming platforms and different input/output data formats, but also by the lack of higher level interoperability among modules developed by different groups.

UIMA, the Unstructured Information Management Architecture [11], was originally developed by IBM. It recently became an open project in OASIS and Apache. It provides a promising framework for tool integration. UIMA has a set of useful functionalities, such as type definitions shared by modules, management of complex objects, linkages between multiple annotations, and the original text, and a GUI for module integration. However, since UIMA only provides a generic framework, it requires a user community to develop their own end-to-end analysis pipelines with a set of actual software modules. A few attempts have already been made to establish platforms for the biomedical domain, including toolkits by the Mayo Clinic [25], the Biomedical Text Mining Group at the University of Colorado School of Medicine [6][26], and Jena University [22], as well as for the general domain, including toolkits by OpenNLP [8], the CMU UIMA component repository [20], and GATE [21] with its UIMA interoperability layer.

However, simply wrapping existing modules for UIMA does not offer a complete solution for flexible tool integration, necessary for practical applications in the biomedical domain. Users, including both the developers and the end-users of TM systems, tend to be confused when choosing appropriate modules for their own tasks from a large collection of tools.

Individual user groups in the biomedical domain have diverse interests. Requirements for NLP/TM modules vary significantly depending on their interests [18]. For example, an NER module developed for a specific user group usually cannot satisfy the needs of another group. Different groups may need different types of entities to be recognized. They may also need to process different types of texts, such as scientific papers, reports, or medical records. Due to this range of needs, significant effort is often required to combine modules that were developed independently for different user groups, even after they are wrapped for UIMA. (Wrapping a tool for UIMA is a process of adding a conversion layer, which wraps the original I/O of the tool in order to communicate with the UIMA framework).

Furthermore, a task in the biomedical domain is composite in nature, from the TM/NLP point of view, and can only be solved by combining several modules. Although the selection of modules affects the performance of the aggregated system, it is difficult to estimate how this selection affects the

ultimate performance of the system. Users need careful guidance in the selection of modules to be combined.

In this paper, we discuss our strategy of using comparators and automatic generators of processing streams to facilitate module integration and to guide the selection of modules. Taking the extraction of protein-protein interaction (PPI) as a typical example of a composite task, we illustrate how our platform helps users construct a system for their own needs.

There are several other technical issues that we encountered as UIMA users. For example, the issue of efficiency cannot be ignored, since we want to process a large collection of documents including all of Medline and full papers in a collection of open access journals in BMC (BioMed Central). From the viewpoint of a tool provider, the burden of making an existing module compatible with a specific platform should be minimized. Some of these issues are discussed in this paper.

2. Motivation and Background

2.1. *Goal Oriented Evaluation, Module Selection and Inter-operability*

There are standard evaluation metrics for NLP/TM modules, including precision, recall, and F-measure. For basic tasks such as sentence splitting, POS tagging, and named-entity recognition, these metrics can be estimated using existing gold-standard test sets. However, accuracy measurements based on standard test sets are sometimes deceptive because the accuracy may change significantly in practice, depending on the types of texts and the actual tasks at hand.

For example, in the bioinformatics task of recognizing occurrences of entities of specific types (e.g. cell-lines, cell locations) in text when comprehensive lexicons for those entities are available, an NER system for an open set of entities (e.g. proteins or metabolites) trained using a gold-standard data set may not be the best choice, even if it yields the best performance on a standard test set. Moreover, systems which have similar levels of performance according to standard metrics often behave differently in specific cases. Because these accuracy metrics do not take into account the importance of different types of errors to any particular application, the practical utility of two systems with seemingly similar levels of accuracy may in fact differ significantly. To users and developers alike, a detailed examination of how systems perform (on the text they would like to process) is often more important than standard metrics and test sets. Naturally, far greater importance is placed in measuring the end-to-end performance of a composite system than in measuring the performance of individual components.

In reality, because selection of modules usually affects the performance of the entire system, careful selection of modules that are appropriate for a given task is crucial. This is the main reason for having a collection of interoperable modules. What we need to be able to test is how the ultimate performance will be affected by selection of different modules and what would be the best combination of modules in terms of the performance of the whole aggregated system for the task at hand.

Since the number of possible combinations of component modules is typically large, the evaluation system has to be able to enumerate and execute them semi-automatically. This requires a higher level of interoperability for individual modules than just wrapping them for UIMA.

2.2. UIMA

2.2.1. *CAS and Type System*

The UIMA framework uses the "stand-off annotation" style [16]. The raw text in a document is kept unchanged during the analysis process. When processing is performed on the text, the result is added as new stand-off annotations with references to their positions in the raw text. A *Common Analysis Structure* (CAS) maintains a set of these annotations, which in turn are objects themselves. The annotation objects in a CAS belong to *types* that are defined separately in a hierarchical *Type System*. The features of an *annotation*[*] object have values which are typed as well.

2.2.2. Components and Component Descriptors

The analysis process, which includes any sort of processing of the text, is performed by one or more *Annotators*, the smallest processing components in UIMA. Components in UIMA are divided into three types: *Collection Reader*, *Analysis Engine* and *CAS Consumer*. An *Analysis Engine* analyzes a document and creates annotation objects. For example, a named entity recognizer receives a CAS, detects named entities in the text, and adds *annotation* objects of a corresponding *type(s)* (`NamedEntity` in our case) to the received CAS. There are two types of *Analysis Engines.* An *Analysis Engine* with a single *Annotator* is called a *Primitive Analysis Engine*, and an *Analysis Engine* with more than two *Annotators* inside is called an *Aggregate Analysis Engine*. A *Collection*

[*] In the UIMA framework, `Annotation` is a base type which has *begin* and *end* offset values, as a subtype of the root type `TOP`. In this paper we call any objects (any subtype of `TOP`) *annotations*.

Reader reads documents from outside of a UIMA framework and generates CASs, while a *CAS Consumer* does not output CASs.

Every UIMA component (i.e. *Collection Reader*, *Analysis Engine* and *CAS Consumer*) has a *descriptor* XML file, which provides its behavioral information. For example, the *Capability* property in a descriptor file describes what *types* of objects the component may take as input and what *types* of objects it produces as output. The compatibility of their *capabilities* is the pre-requisite for two components to be combined.

It is possible to deploy any UIMA component as a SOAP web service. Therefore, we can combine a remote component on a web service with local component freely inside a UIMA-based system.

3. Integration Platform and Comparators

3.1. *Shared Type System*

Although UIMA provides a useful set of functionalities for an integration platform of NLP/TM tools, users still have to develop the actual platform to use these functionalities effectively. The designer of an integration platform must make several decisions.

Firstly, as a crucial decision, the designer must decide how to use *types* in UIMA. At one extreme, the designer may wrap existing programs without using explicit *types*, putting information into a single String field of a common generic *type*. Since compatibility among modules is already automatically guaranteed, such a design decision would be easy to follow; however, it would not be appropriate if we aim to attain the higher level of inter-operability required for goal-oriented module selection and evaluation.

At the other extreme, the designer may force all modules developed by different groups to accept a unique *type system* which the platform defines. While this would make inter-operability readily attainable, it puts too much of a burden on the individual modules. In the worst case, we may have to re-program all of the tools developed by other groups. Thus, this design is impractical.

Our decision lies in the middle between these two extremes. That is, if necessary, we keep different *type systems* by individual groups as they are. We require, however, that individual *type systems* have to be related through a common, shared *type system* which our platform defines. Such a shared *type system* can bridge modules with different *type systems*, though bridging module may lose some information during the translation process.

Whether such a shared *type system* can be defined or not is dependent on the nature of each problem. For example, a shared *type system* for POS tags in

English can be defined rather easily, since most of POS-related modules, such as POS taggers (their output is a sequence of POSs), shallow parsers (their input is a sequence of words with their POS assignments), etc., more or less follow the well-established types defined by the Penn Treebank [24] tag set for POS types.

Figure 1 shows a part of our shared *type system*. We deliberately define a highly organized type hierarchy, since the structure of a shared common *type system* directly influences the loss of information during the translation process. For instance, it is better to express each POS as a distinct *type*, not as a String feature value, in order to identify each POS uniquely. It is also better to make abstract *types* in hierarchies as much as possible, in order not to lose information during the translation between *type systems*. For example, if a local *type system* has a *type* of general verb but has no *type* of past tense verb, then the shared *type system* should have an abstract *type* (like `Verb`) in order to capture the local *type* information.

Secondly we should consider that the *type system* could be used to compare and/or mix similar tools. *Types* should be defined in a distinct and hierarchical manner; both tokenizers and POS taggers generate a variety of tokens, but their roles are different when we assume a cascaded pipeline. We defined `Token` as a supertype (tokenizer) and `POSToken` (POS tagger) as a subtype of `Token`. Each tool should have an individual *type* to make clear which tool generated which instance; this is necessary because each tool may have a slightly different definition of output *types* even if they are the same sort of tools.

3.2. *General Combinatorial Comparison Generator*

Even if the *type system* is defined in the way previously described, there are still some issues to consider when comparing tools. We illustrate these issues using

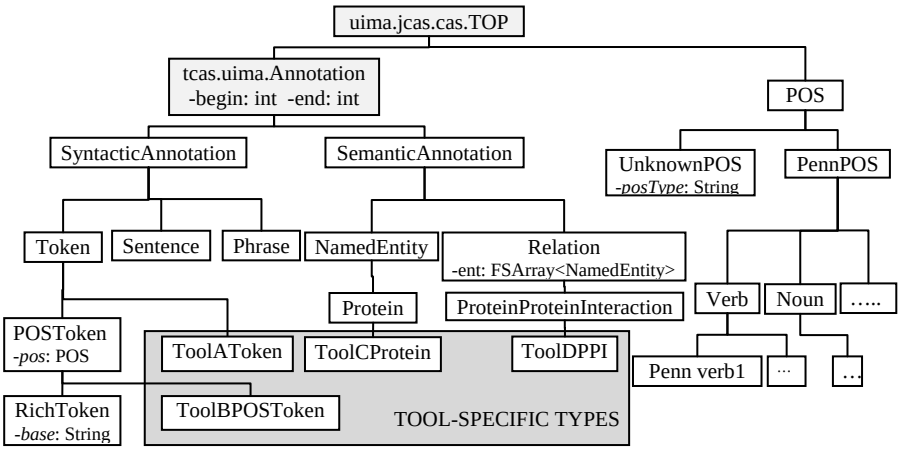

Figure 1 Part of our type system

the PPI *workflow* that we utilized in our experiments.

Figure 2 shows the *workflow* of our whole PPI system conceptually. If we can prepare two or more *Annotators* for some type of the components in the *workflow* (e.g. two sentence detectors and three POS taggers), then we could make combinations of these tools to form a multiplied number of *workflow* patterns (2x3 = 6 patterns). See Table 1 for the details of UIMA components used in our experiments.

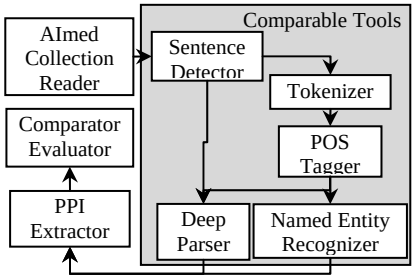

Figure 2. PPI system workflow (conceptual)

We made a pattern expansion mechanism which generates possible *workflow* patterns automatically from a user-defined *comparable workflow*. A *comparable workflow* is a special *workflow* which explicitly specifies which set of *Annotators* should be compared. Then, users just need to group comparable components (e.g. ABNER[†] and MedT-NER as a comparable NER group) without making any modifications to the original UIMA components. This aggregation of comparable *Annotators* is controlled by our *custom workflow controller*.

In some cases, a single tool can play two or more roles (e.g. the GENIA Tagger performs tokenization, POS tagging, and NER; see Figure 4). It may be possible to decompose the original tool into single roles, but in most cases it is difficult and unnatural to decompose such a complex tool. We designed our comparator to detect possible input combinations automatically by the *types* of previously generated *annotations*, and the input *capability* of each posterior *Annotator*. As described in the previous section, *Annotator* should have appropriate *capabilities* with proper *types* in order to permit this detection.

When an *Annotator* requires two or more input *types* (e.g. our PPI extractor

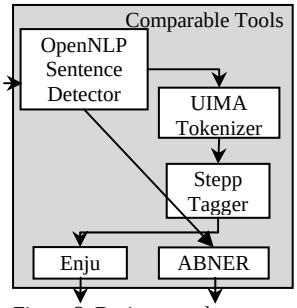

Figure 3. Basic example pattern

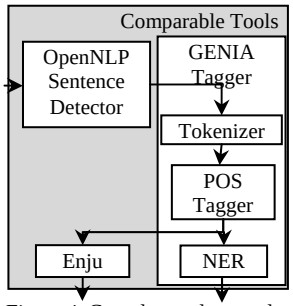

Figure 4. Complex tool example

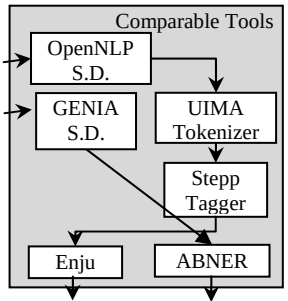

Figure 5. Branch flow pattern

[†] In the example figures, ABNER requires `Sentence` to make the explanation clearer, though ABNER does not require it in actual usage.

requires outputs of a deep parser and a protein NER system), there could be different *Annotators* used in the prior flow (e.g. OpenNLP and GENIA sentence detectors in Figure 5). Thus, our comparator calculates such cases automatically.

Because of limitations of the current Apache UIMA implementation, we originally defined `AnnotationGroup`, each of which holds *annotations* generated by a single *Annotator* in a specific *workflow* pattern. An `AnnotationGroup` has dependency links to the prior `AnnotationGroups`. Because an expanded combinatorial *workflow* is cascaded, `AnnotationGroups` are shared within posterior *Annotators* in order to increase performance.

Although it is efficient to share `AnnotationGroups`, whole combinatorial results are put into a single CAS in this design and a CAS may contain a large number of *annotations*. When web services or network communications are used, a large CAS could be costly with respect to transmission time, and may therefore decrease the performance of the system. In addition it is impossible for normal UIMA components to process such a mixture of combinatorial *annotations*. We made a special adapter component which generates a temporary CAS by the *CAS Multiplier* functions. This temporary CAS contains only a set of required *annotations* for each component in order to avoid these problems.

Table 1.List of UIMA-compliant tools that we used in the experiment.
Legend: Input type(s) required for that tool Input required optionally Output type(s)

`Sentence`	`Token`	`POSToken`	`RichToken`	`Protein`	`Phrase`	`PPI`
GENIA Tagger: Trained on the WSJ, GENIA, and PennBioIE corpora (POS). Uses Maximum Entropy [17], trained on the NLPBA data set [9], derived from the GENIA corpus (NER) [15].						
Enju: HPSG parser with predicate-argument structures (PAS) as well as phrase structures. Although trained with newswire articles (Penn Treebank [24]), it can compute accurate analyses of biomedical texts due to our method for domain adaptation. The evaluated bio-performance is 86.9 F-score [4].						
STePP Tagger: Based on probabilistic models, tuned to biomedical text trained by WSJ, GENIA, and PennBioIE corpora, with state-of-the-art performance (97.3% on the standard WSJ test set).						
MedT-NER: Statistical recognizer trained on the JNLPBA [9] data. NEs are normalized to Uniprot entries to provide a high-quality link into existing databases. When a name is ambiguous between several Uniprot entries, a MaxEnt classifier is used to rank the candidate IDs.						
ABNER: From the University of Wisconsin [10], wrapped by the Center for Computational Pharmacology at the University of Colorado.						
Akane++: A new version of the AKANE system [12], trained by SVM^{light}-TK [14,19] and the AImed Corpus, achieving state-of-the-art F-measures for PPI using 10-fold cross validation (F=52- 69).						
Annotation Comparator and Evaluator: Compares annotations using the type system hierarchy to decide which annotations can be compared; generates statistical results and visualization.						
UIMA Examples: Provided in the Apache UIMA example. Sentence Splitter and Tokenizer.						
OpenNLP Tools: Part of the OpenNLP project [8], included in the Apache UIMA example.						
AImed Corpus: 225 Medline abstracts with proteins and protein-protein interactions annotated [13].						

3.3. *User- and Developer-Friendly Utilities*

For the end-user utilities, our comparator provides a filtering function and visualization of the results, in addition to providing statistical results.

Web services are a better option when a specific runtime environment or rich computational resources are required, when a tool cannot be distributed due to licensing issues, or when it is necessary to save the time needed for module initialization. We deployed most of our components as SOAP web services so that users can launch our entire *workflow* from any environment.

We also made a single-click-to-launch system based on the Java Web Start technology. Users need not follow any explicit installation process or settings, if their machines already have Java installed.

Although Apache UIMA provides its Java APIs and C++ enhancement kit with rich functionality, it is cumbersome for developers to make their existing tools UIMA-compliant. For developers, we provide a simpler I/O interface that does not depend on any specific programming languages, so that the developers do not need to learn anything about Java or UIMA when they need to wrap existing tools into UIMA. Wrapper developers should only have to make stand-off annotations, using specified *type* and feature names, via the standard I/O streams. Our Java adapter then automatically performs all tasks to wrap the tools.

4. Experiments and Results

We have performed experiments using our PPI extraction system as an example. The PPI system (Figure 2) is similar to our BioCreative PPI system [7]. It differs in that we have decomposed the original system into seven different components.

4.1. *Combinatorial Comparison*

As summarized in Table 1, we have several comparable components and AImed as gold standard data. In this case, possible combination *workflow* patterns are 36 for `PosToken`, 589 for `ProteinProteinInteraction`, etc.

Table 2. Screenshot of a POS combinatorial comparison. Values are precision/recall in "labeled (unlabeled)" pairs, and total numbers of instances are shown.

.POSToken .POS	.Sentence	.SimpleToken	gold	OpenNLP Genia OpenNLP 3105	Stepp Genia OpenNLP 1693
Stepp	Genia	UIMA	1661	40/75 (40/75)	62/63 (62/6
Stepp	AIMED	UIMA	1661	40/75 (40/75)	62/63 (62/6
GENIA	Genia	UIMA	1661	39/73 (39/73)	53/54 (53/5
GENIA	AIMED	UIMA	1661	39/73 (39/73)	53/54 (53/5
Stepp	AIMED	GENIA	1690	31/57 (31/57)	63/63 (63/6
Stepp	AIMED	OpenNLP	1692	31/57 (31/57)	63/63 (63/6

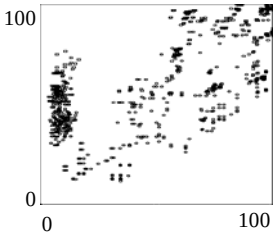

Figure 6. NER comparison distribution of precisions (x-axis) and recalls (y-axis).

Table 2 and Figure 6 show a part of the comparison result screenshots between these patterns on 20 articles from the AImed corpus. In Table 2, *labeled* scores represent complete matches of every feature of *annotations*, while unlabeled scores ignore primitive fields excluding offsets (e.g. compare offsets but ignore protein IDs). Table 3 shows a part of PPI extraction results from which we discern which combination of tools generate the best result.

When neither of compared results include the gold standard data (AImed in this case), the comparison results show a *similarity* of the tools for this specific task and data, rather than an evaluation. Even if we lack an annotated corpus, it is possible to run tools and compare results in order to understand the characteristics of tools depending on the corpus and the tool combinations.

4.2. *Performance with Multi-threading*

Apache UIMA provides an option to enable multi-threading of a *workflow* or multi-deployment of components without modifying UIMA components. We have tested multi-threading performance and the result suggests that we can increase the overall performance easily by using a parallel architecture. Because CPU architectures are evolving rapidly towards multi-cores in order to increase global performance, the capability of UIMA to support multi-threading promises considerable advantages, despite the wrapper overheads or web service communication overheads.

Table 3. PPI Evaluated on AImed, with 5631 protein pairs. (1068 true interactions). DEP means our dependency parser. Values are percentages from 10-fold cross-validation on abstracts. "pairwise" is the widely used 10-fold cross-validation on protein pairs. Refer to [23] for details.

Features	Prec	Recall	F1
DEP	67.6	26.3	37.1
WORDS	55.7	29.2	37.8
PAS (Enju)	72.0	28.7	41.0
DEP+WORDS	59.9	39.3	46.9
PAS+DEP	68.9	37.8	48.6
PAS+WORDS	61.3	40.7	48.6
ALL	64.3	44.1	52.0
ALL (pairwise)	78.1	62.7	69.5

5. Conclusion and Future Work

Although UIMA provides a general framework with much functionality, we still need to fill the gaps between what is already provided and what the users need for their specific tasks. Biomedical tasks typically consist of many components, and it is necessary to show which sets of tools are most suitable for each specific task and data. In this paper, we provided an answer to this problem using extraction of protein-protein interaction as an example task.

With any set of UIMA components that have *types* designed in the way described in this paper, our general combinatorial comparator generates possible combinations of tools for a specific *workflow* and compares/evaluates the results.

We are preparing to make a portion of the components and services described in this paper available publicly (http://www-tsujii.is.s.u-tokyo.ac.jp/uima/).

The system shows which combination of components yields the best score, and also succeeds in generating comparative results. This helps users to grasp the characteristics of and differences between the tools, which cannot be easily observed just by the widely used F-score metric.

Future directions for this work include combining the output of several modules of the same kind (such as NER systems) to obtain better results, collecting other tools developed by other groups using bridging type systems, making machine learning tools UIMA-compliant, and making grid computing available with UIMA *workflows* to increase overall performance.

Acknowledgments

We wish to thank Dr. Lawrence Hunter's text mining group at the Center for Computational Pharmacology for discussing with us and making their tools available for this research. This work was partially supported by NaCTeM (the UK National Centre for Text Mining), Grant-in-Aid for Specially Promoted Research (MEXT, Japan) and Genome Network Project (MEXT, Japan). NaCTeM is jointly funded by JISC/BBSRC/EPSRC.

References

1. Y. Tsuruoka, Y. Tateishi, J. D. Kim, T. Ohta, J. Tsujii and S. Ananiadou, *Developing a Robust Part-of-Speech Tagger for Biomedical Text.* Volos: In the Advances in Informatics. LNCS 3746: pp. 382-392 (2005).
2. N. Okazaki and S. Ananiadou, *Building an abbreviation dictionary using a term recognition approach.* Bioinformatics, pp. 22(24):3089-3095 (2006).
3. S. Pyysalo, T. Salakoski, S. Aubin and A. Nazarenko, *Lexical adaptation of Link grammar to the biomedical sublanguage: a comparative evaluation of three approaches.* BMC Bioinformatics. Suppl 3:S2 (2006).
4. T. Hara, Y. Miyao and J. Tsujii, *Evaluating Impact of Re-training a Lexical Disambiguation Model on Domain Adaptation of an HPSG Parser*. In the Proceedings of IWPT 2007. Prague, Czech Republic, June 2007.
5. L. Hirschman, M. Krallinger and A. Valencia, *Proc. of Second BioCreative Challenge Evaluation Workshop*. Madrid: Centro Nacional de Investigaciones Oncologicas (2007).
6. H. L. Johnson, W. A. Baumgartner, M. Krallinger, K. B. Cohen and L. Hunter. *Corpus refactoring: a feasibility study*. J Biomed Discov Collab pp. 2:4 (2007).
7. R. Sætre, K. Yoshida, A. Yakushiji, Y. Miyao, Y. Matsubayashi and T. Ohta, *AKANE System: Protein-Protein Interaction Pairs in the BioCreAtIvE2 Challenge*, PPI-IPS subtask (2006).

8. J. Baldrige and T. Morton, *OpenNLP*. http://opennlp.sourceforge.net/
9. J. D. Kim, T. Ohta, Y. Tsuruoka, Y. Tateishi and N. Collier, *Introduction to the bio-entity recognition task at JNLPBA.* Geneva, Switzerland. JNLPBA-04. pp. 70–75 (2004).
10. B. Settles, *ABNER: an open source tool for automatically tagging genes, proteins, and other entity names in text.* Wisconsin: Bioinformatics. pp. 21(14):3191-2 (2005).
11. A. Lally and D. Ferrucci, *Building an Example Application with the Unstructured Information Management Architecture*, IBM Systems Journal 43, No. 3, pp. 455–475 (2004).
12. A. Yakushiji, *Relation Information Extraction Using Deep Syntactic Analysis*, PhD. thesis. University of Tokyo (2006).
13. R. C. Bunescu and R. J. Mooney, *Subsequence kernels for relation extraction.* NIPS (2005).
14. T. Joachims, *Making large-Scale SVM Learning Practical.* In B. Schölkopf and C. Burges and A. Smola (ed.), Advances in Kernel Methods - Support Vector Learning, MIT-Press (1999).
15. J. D. Kim, T. Ohta, Y. Tateishi, and J. Tsujii, *GENIA corpus - a semantically annotated corpus for bio-textmining.* Bioinformatics. pp. 19(suppl. 1):i180–i182 (2003).
16. D. Ferrucci et al. *Towards an Interoperability Standard for Text and Multi-Modal Analytics.* IBM Research Report, RC24122 (2006).
17. A. L. Berger, S. D. Pietra, and V. J. D. Pietra, *A maximum entropy approach to natural language.* Comp. Ling., pp. 22(1):39–71 (1996).
18. S. Ananiadou, D. B. Kell and J. Tsujii, *Text mining and its potential applications in systems biology.* Trends Biotechnol, Vol. 24 (2006).
19. A. Moschitti, *Making tree kernels practical for natural language learning.* Trento, Italy. In Proc. EACL-2006.
20. The Carnegie Mellon University, *UIMA component repository*. http://uima.lti.cs.cmu.edu/
21. H. Cunningham, D. Maynard, K. Bontcheva and V. Tablan. *GATE: an Architecture for Development of Robust HLT.* In Proc. ACL-2002.
22. The JULIE Lab (the Jena University Language & Information Engineering Lab). http://www.julielab.de/
23. R. Sætre et al. *Syntactic features for protein-protein interaction extraction*. LBM2007, to be submitted.
24. M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. *Building a large annotated corpus of english: The penn treebank*. Comp. Ling. 19:313--330 (1993).
25. S. Pakhomov, J. Buntrock, P. Duffy. *High Throughput Modularized NLP System for Clinical Text (Interactive Poster)*. ACL 2005; Ann Arbor, MI.
26. W. A. Baumgartner, Z Lu, H. L. Johnson, J. G. Caporaso, J. Paquette, A. Lindemann, E. K. White, O. Medvedeva, L. M. Fox, K. B. Cohen, and L. Hunter. *An integrated approach to concept recognition in biomedical text.* Proc. of the Second BioCreative Challenge Evaluation Workshop (2006).