Using UIMA to Structure an Open Platform for Textual Entailment

Tae-Gil Noh and Sebastian Padó

Department of Computational Linguistics
Heidelberg University
69120 Heidelberg, Germany
{noh, pado}@cl.uni-heidelberg.de

Abstract. EXCITEMENT is a novel, open software platform for Textual Entailment (TE) which uses the UIMA framework. This paper discusses the design considerations regarding the roles of UIMA within EXCITEMENT Open Platform (EOP). We focus on two points: a) how to best design the representation of entailment problems within UIMA CAS and its type system. b) the integration and usage of UIMA components among non-UIMA components.

Keywords: Textual Entailment, UIMA type system, UIMA application

1 Introduction

Textual Entailment (TE) captures a common sense notion of inference and expresses it as a relation between two natural language texts. It is defined as follows: *A Text (T) entails a Hypothesis (H), if a typical human reading of T would infer that H is most likely true* [4]. Consider the following example:

T: *That was the 1908 Tunguska event in Siberia, known as the Tunguska meteorite fall.*
H_1: *A shooting star fell in Russia in 1908.*
H_2: *Tunguska fell to Siberia in 1908.*

The text (T) entails the first hypothesis (H_1), since a typical human reader of T would (arguably) believe that H_1 is true. In contrast, T does not entail H_2. Nor does H_1 entail T, that is, entailment is a directed relation.

The promise of TE lies in its potential to subsume the semantic processing needs of many NLP applications, offering a uniform, theory-independent semantic processing paradigm. Software for the Recognition of Textual Entailment (RTE) have been used to build proof-of-concept versions of various tasks, including Question Answering, Machine Translation Evaluation, Information Visualization, etc. [1, 7].

As a consequence of the theory-independence of TE, there are many different strategies to build RTE systems [1]. This has led to a practical problem of *fragmentation*: Various systems exist, and some have been made available as open-source systems, but there is little to no interoperability between them, since the systems are, as a rule, designed to implement one specific algorithm to solve RTE. The problems is complicated by the fact that RTE systems generally rely on tightly integrated components such as linguistic analysis tools and knowledge resources. Thus, when a researcher wants to develop a new RTE algorithm, they often need to invest major effort to build a novel system from scratch: Many of the components already exist – but just not in a usable form.

EXCITEMENT open platform (EOP) has been developed to address those problems. It is a *suite* of textual inference components which can be combined into complete textual inference systems. The platform aims to become a common development platform for RTE researchers, and we hope that it can establish itself in the RTE community in a similar way to MOSES [6] in Machine Translation.

Compared to Machine Translation, however, a major challenge is that semantic processing typically depends on linguistic analysis as well as large knowledge sources, which is a direct source of the reusability problems mentioned above. In this paper, we focus on the architectural side of the platform which was designed with the explicit goal of improving component re-usability. We have adopted UIMA (Unstructured Information Management applications) and UIMA CAS (Common Analysis Structure) as the central building blocks for data representation and preprocessing within EOP.

One interesting aspect is that our adoption of UIMA has been *partial* and *parallel*. By *partial*, we mean that there are two groups of sharable components within EOP: the "core" components and the "LAP" components (see Section 2). We have adopted UIMA only for LAPs; however, we use UIMA CAS as one of the standard data containers, even in non-UIMA components. *Parallel* refers to the fact that we allow non-UIMA components to be integrated into our LAPs transparently.

2 EXCITEMENT: An Open Platform for Textual Entailment Systems

RTE systems traditionally rely on self-defined input types, pre-processing (linguistic annotation) representations, and resources, tailored to a specific approach to RTE. EXCITEMENT open platform (EOP) tries to alleviate this situation by providing a generic platform for sharable RTE components. The platform has the following requirements.

Reusing of existing software : The platform must permit easy integration and re-using of existing softwares, including language processing tools, RTE components, and knowledge resources.

Multilinguality : The platform is not tied to a specific language. Adding suites for a new language in the future should not be restricted by the platform design.

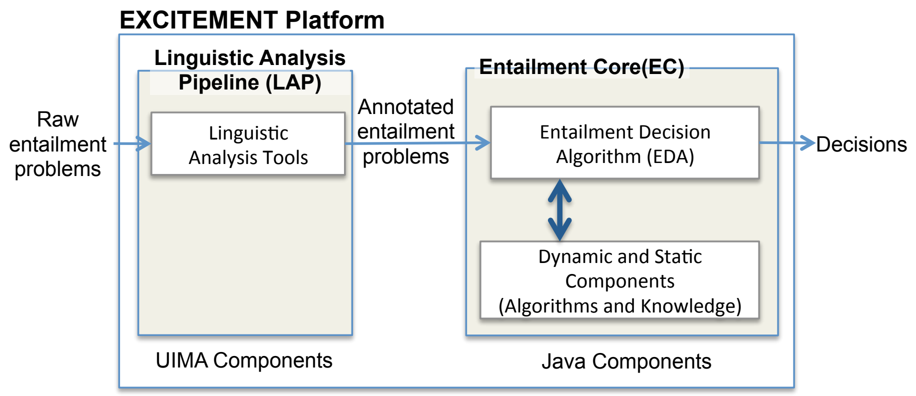
EXCITEMENT Platform
Raw entailment problems
Linguistic Analysis Pipeline (LAP)
Linguistic Analysis Tools
Annotated entailment problems
Entailment Core(EC)
Entailment Decision Algorithm (EDA)
Dynamic and Static Components (Algorithms and Knowledge)
Decisions
UIMA Components
Java Components

Fig. 1: EXCITEMENT Architecture Overview

Component Independence : Components of EOP should be independent and complete as they are. So they can be used by different RTE approaches. This is also true for linguistic annotation pipelines and their components: An annotation pipeline as a whole, or an individual component of the pipeline, can be replaced with equivalent components.

Figure 1 visualizes the top level of the platform. At this level, the platform can be grouped into two boxes: one is the Linguistic Analysis pipeline (LAP), and the other is the Entailment Core (EC). Entailment problems are first analyzed in the LAP, since almost all RTE algorithms require some level of linguistic annotation (e.g., POS tagging, parsing, NER, or lemmatization). The annotated TE problems are then passed to the EC box. In this box, the problems are analyzed by Entailment Decision Algorithms (EDAs), which are the “core” algorithms that make the entailment call and may in turn call other core components to provide either algorithmic additions or knowledge. Finally, the EDA returns an entailment decision.

It is relatively natural to think of the LAP in terms of UIMA, since the typical computational linguistic analysis workflow corresponds well to UIMA’s annotation pipeline concept. Each annotator in LAP adds some annotations, and downstream annotators can use existing annotations and add richer annotations. UIMA CAS and its type system are strong enough to represent any data. UIMA AEs (Analysis Engines) are a good solution for encapsulating and using annotator components. In Section 3, we describe the UIMA adoption in the LAP in more detail.

For Entailment Core (EC) components, however, the situation is different. In contrast to LAP, the functionalities of EC components are often not naturally mapped as “annotation behavior”. To visualize this, let’s check the example in Figure 2. The figure shows a conceptual search process of a RTE system that is based on textual rewriting. In this example, the text is “Google bought Motorola.”, and the system tries to determine hypothesis “Motorola is acquired by

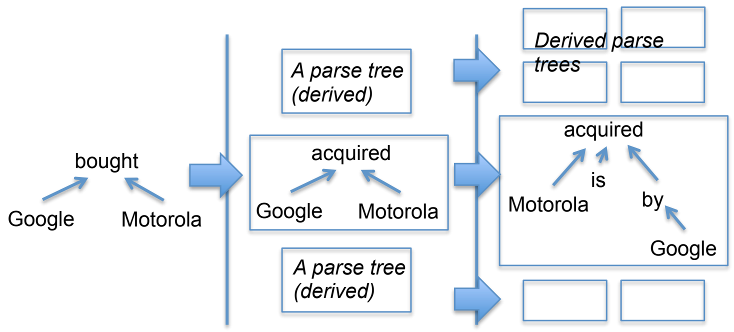

Fig. 2: Entailment as a search on possible rewritings

Google." as an entailment. The example system gets a dependency parse tree of the text, and starts the rewriting process. On each iteration, it generates possible entailed sentences by querying knowledge bases. In the example, lexical knowledge is used on the first rewriting (*buy* entails *acquire*), and syntactic knowledge (change to passive voice) is used on the second derivation. The process will generate many derived candidates per iteration. The algorithm must employ a good search strategy to find the best rewriting path from text (T) to hypothesis (H).

On this example, there are three major component types. One is the knowledge component type that supports knowledge look-up, another is generation of derived parse trees, and finally the decision algorithm itself drives the search process and makes the entailment decision. Expressing behaviors of such components in terms of annotations on the artifact, might be possible, but is very hard and counter-intuitive.

Following this line of reasoning, we decided that the EC components are better thought of as Java modules whose common behavior is defined by a set of Java interfaces, data types, and contracts, and have defined them accordingly in the EXCITEMENT open platform specification.[1] More specifically, we have defined a typology of components. They include a type for the top-level EDA as well as (currently) five major component types: (1) a feature extractor (get a T-H pair CAS, return a set of features for the T-H pair); (2) a semantic distance calculator (get a T-H pair CAS, return semantic similarity); (3) a lexical resource type (lexical relation database); (4) a syntactic resource type (phrasal relation database); (5) an annotation component (dynamic enrichment of entailment problems).

Although UIMA components are not suitable for conceptualizing inference components, we decided to keep CAS as the data container even in the EC components as far as possible to take advantage of the CAS objects created in the LAP. Thus, various components (including EDAs) gets CAS (as JCas) as an argument on their methods. Also note that LAP and EC boxes are independent:

[1] *Specification and architecture for EXCITEMENT open platform*, http://excitement-project.eu/index.php/results

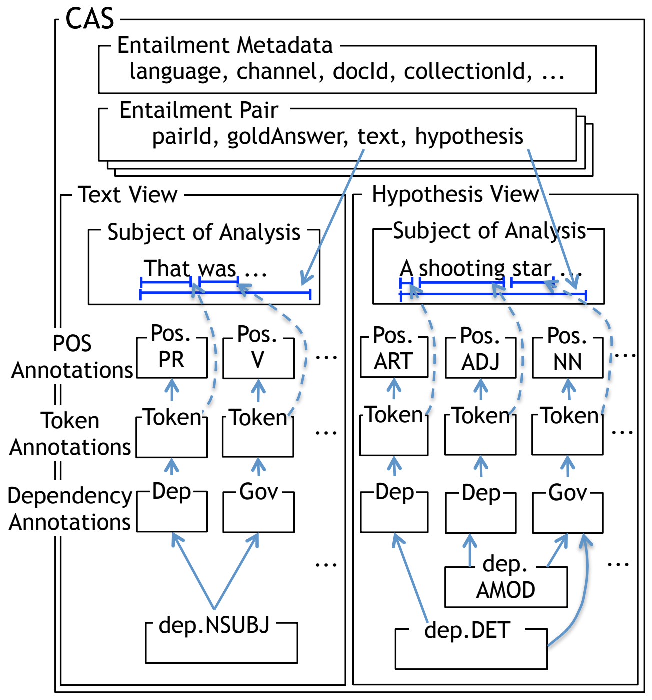

Fig. 3: CAS representation of a Text-Hypothesis pair

as long as the CAS holds correct data, the EC components does not care which pipeline has generated the data.

3 Details on the UIMA usage in EXCITEMENT

3.1 CAS for Entailment Problems

The input to any RTE system is a set of *entailment problems*, typically Text-Hypothesis pairs, each of which is represented in one CAS. Figure 3 shows a pictorial example of the CAS data structure for the example pair (T, H_1) from Section 1. It contains the two text fragments (in two views) and their annotations (here, POS tags and dependencies), as well as global data such as generic metadata (e.g., language) and entailment-specific metadata (e.g., the gold-standard answer).

On the level of the CAS representation, we had to address two points: one is the representation of entailment problems in terms of CASes, the other one is the type definitions.

Regarding the first point, general practice in text analysis use cases is to have one UIMA CAS corresponding to one document. This suggests representing both text and hypothesis (including, if available, their document context) as separate CASes. However, we decided to store complete entailment problems as individual CASes, where each CAS has two named views (one view for text, the other for hypothesis). This approach has two major advantages: first of all, this enables us to represent cross-annotations between texts and hypotheses, notably alignments, which can be added by annotators. Second, this enables us to define a straightforward extension from "simple" entailment problems (one text and one hypothesis) to "complex" entailment problems (one text and multiple hypotheses or vice versa, as in the "RTE search" task [2]).

Regarding the second point, we adopted the DKPro type system [5], which was designed with language independence in mind. It provides types for morphological information, POS tags, dependency structure, named entities, and co-reference, etc. We extended the DKPro type system with the types necessary to define textual entailment-specific annotation. This involved types for marking stretches of text as texts and hypotheses, respectively, as well as storing correspondence information between texts and hypotheses, pair IDs, gold labels, and some meta data. We also added types for linguistic annotation that are not exclusively entailment-specific, but were not covered yet by DKPro. This included annotation for polarity, reference of temporal expressions, word and phrase alignments, and semantic role labels.

Details about the newly defined types can be found in the platform specification, and the type definition files are part of the platform code distribution.

3.2 Wrapping the Linguistic Annotation Pipeline

One decision that may be surprising at the first glance is that we defined our own top-level Java interface for users of the LAP that hides UIMA's own runtime access methods. This interface dictates the common capabilities that all pipelines of LAP should provide.

The reason for this decision is twofold and pragmatic in nature, making transitioning to and using the EOP as easy as possible for developers.

The first aspect is the learning curve. We would like to avoid the need for Entailment Core developers to deeply understand UIMA AEs and Aggregated Analysis Engines (AAEs). We feel that a deep understanding of these points requires substantial effort but is not really to the point, since many EC developers will only want to use pre-existing LAPs. By making the UIMA aspect of the LAP transparent to the Entailment Core, EC developers do not need to know how the LAP works internally beyond knowledge of the (fairly minimal) LAP interface. Of course, the EC developers still need to understand UIMA CAS very well.

The second aspect is migration cost. If the LAP pipelines were nothing but UIMA AEs, all analysis pipelines of existing RTE systems would have to be deeply refactored, which comes at a considerable cost. Our approach allows such analysis pipelines to be kept largely intact and merely surrounded by a wrapper that provides the requires functionality and converts their output into valid UIMA CASes according to the EOP's specification.

Nevertheless, there are good reasons to encourage the use of AE-based LAPs: AE-based components are generally much more flexible, and they are very easy to assemble into AAE pipelines. Therefore, we encourage AE-based LAP development by providing ready-to-use code that implements our LAP interface, taking a list of AEs as input. Thus, if the individual components are already present as AEs, the implementation effort to assemble them into a LAP is near zero. In this sense, we see our LAP interface as a thin wrapper above UIMA with the purpose of enabling peaceful co-existence between UIMA and non-UIMA pipelines. In the long run, we also hope to provide some new AEs back to the UIMA community.

4 Some Open Issues

In this section, we discuss two open questions that we are facing in future work.

CAS in non-UIMA environments. There is considerable number of best-practice strategies for handling CAS objects (reset the data structure instead of creating a new one; use a CAS pool instead of generating multiple CASes, etc). When a CAS is used in an UIMA context (i.e., in the LAP), it is not hard to guide the developers to follow these rules. However, with CAS being used as a general data container throughout the EOP, developers also often encounter CAS (JCas) objects outside specific UIMA contexts, and we have found it harder to guide the developers towards "proper usage".

For example, one part of the EXCITEMENT project is concerned with the construction of Entailment Graphs [3], structured knowledge repositories whose vertices are statements and whose edges indicate entailment relations. Since the standard data structure for annotations is JCas, the graph developers tend to add one JCas for each node. This is not problematic for small graphs, but once the graph gets bigger, this can be problematic; CAS is a very large data structure, and its creation and deletion take some time. We are still trying to establish best practices for using CASes in non-UIMA EOP environment.

Annotation Styles: Hidden dependencies. One of the EOP design requirement was the clear separation of LAP and EC. This has been fairly well achieved, at least on a technical level.

However, it is clear that there are still *implicit* dependencies between linguistic analysis tools and entailment core components. Consider the case of syntactic knowledge components such as DIRT-style paraphrase rules in the Entailment Core. Such components store entailment rules as pairs of partial dependency trees which have typically been extracted from large corpora. If the corpus used for rule induction was parsed with a different parser than the current entailment problem, then matching the sentence against the rule base will result in missing rules, due to differences in the analysis style. Note that this implicit dependency does not break the UIMA pipeline, since it does not involve the use of a novel type system, but rather differences in the interpretation of shared types. We are currently investigating what type of "style differences" can be observed from actual annotators.

5 Conclusion

In this paper, we have provided an overview of the EXCITEMENT open platform architecture and its adoption of UIMA. We have adopted and adapted UIMA CAS and the DKPro type system as a flexible, language-independent data container for Textual Entailment problems. UIMA also provides the backbone for platform's LAP components. There are several open issues that is to be resolved in the future, but the EXCITEMENT project has already profited substantially from the use of the abstractions that UIMA offers as well as the integration of existing components from UIMA communities.

The first version of EXCITEMENT open platform has been finished[2] with three fully running RTE systems integrated with all core components and annotation pipelines. The platform currently supports three languages (German, Italian and English), and is also shipped with various tools and resources for TE researchers. We believe that the platform will become a valuable tool for researchers and users of Textual Entailment.

Acknowledgment. This work was supported by the EC-funded project EXCITEMENT (FP7 ICT-287923).

References

1. Androutsopoulos, I., Malakasiotis, P.: A Survey of Paraphrasing and Textual Entailment Methods. Journal of Artificial Intelligence Research **38** (2010) 135–187
2. Bentivogli, L., Magnini, B., Dagan, I., Trang Dang, H., Giampiccolo, D.: The fifth PASCAL recognising textual entailment challenge. In: Proceedings of the TAC 2009 Workshop on Textual Entailment, Gaithersburg, MD (2009)
3. Berant, J., Dagan, I., Goldberger, J.: Learning entailment relations by global graph structure optimization. Computational Linguistics **38**(1) (2012) 73–111
4. Dagan, I., Glickman, O., Magnini, B.: The PASCAL Recognising Textual Entailment Challenge. In: Proceedings of the First PASCAL Challenges Workshop on Recognising Textual Entailment, Southampton, UK (2005)
5. Gurevych, I., Mühlhäuser, M., Müller, C., Steimle, J., Weimer, M., Zesch, T.: Darmstadt knowledge processing repository based on UIMA. In: Proceedings of the First Workshop on Unstructured Information Management Architecture at the Conference of the Society for Computational Linguistics and Language Technology, Tübingen, Germany (2007)
6. Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., Herbst, E.: Moses: Open source toolkit for statistical machine translation. In: Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, Prague, Czech Republic (2007) 177–180
7. Sammons, M., Vydiswaran, V., Roth, D.: Recognizing textual entailment. In Bikel, D.M., Zitouni, I., eds.: Multilingual Natural Language Applications: From Theory to Practice. Prentice Hall (2012)

[2] The platform has been released under an open source license, and all codes and resources can be freely accessed via the project repository. http://hltfbk.github.io/Excitement-Open-Platform/project-info.html