Predicate-Argument EXtractor (PAX)

Ralf Krestel,[1] René Witte,[2] and Sabine Bergler[2]

[1]L3S Research Center
Leibniz Universität Hannover, Germany

[2]Department of Computer Science and Software Engineering
Concordia University, Montréal, Canada

Abstract

In this paper, we describe the open source GATE component PAX for extracting predicate-argument structures (PASs). PASs are used in various contexts to represent relations within a sentence structure. Different "semantic" parsers extract relational information from sentences but there exists no common format to store this information. Our predicate-argument extractor component (PAX) takes the annotations generated by selected parsers and transforms the parsers' results to predicate-argument structures represented as triples (subject-verb-object). This allows downstream components in an analysis pipeline to process PAS triples independent of the deployed parser, as well as combine the results from several parsers within a single pipeline.

1. Introduction

Recent NLP applications have increasingly tackled semantic notions, such as textual entailment determination, incremental summary generation, or event extraction in BioNLP. The basic first issue in all these tasks is the scoping of predicative constructs as it is expressed in a predicate-argument structure (PAS). PASs can be extracted from the output of parsers; in particular dependency parsers output the component semantic relations of *subject* and *object* directly and assembling PAS structure amounts mainly to combining the related semantic relations for each verb.

Great progress has been achieved over the past 10 years with the increasing availability of different systems that tackle the same, matured tasks (POS tagging, parsing, IR) and with integration platforms that facilitate mixing and matching different application modules in a pipeline of greater and greater sophistication, reducing development time and increasing reuse of tested systems. One influential integration platform is GATE (Cunningham et al., 2002), which provides components for most steps in common NLP tasks, including several parsers. To manage these parsers' output within a pipeline in need of PAS annotations, a "normalized" output format is needed.

We present here a system that has been conceived as a GATE component that extracts PA structures from the output of several different parsers. It shields downstream components from the different output formats of the different parsers by providing a common result structure, thereby facilitating experiments that mix and match different parsers in an analysis pipeline. Our PAX component is available under an open source license.[1]

2. Predicate-Argument Structures (PASs)

Most verbs in English require a subject and an object to be specified in a grammatical sentence. For simple sentences, this subject-verb-object structure constitutes a complete analysis; for more complex sentences the task is to identify the PA structure, to assign the correct arguments to all verbs, and to identify adjuncts, i.e., PPs or NPs that are not in argument position (Merlo and Ferrer, 2006).

Dependency parsers have been addressing this as the major issue for some time and some prioritize correct dependencies over achieving a complete parse for a sentence. Even full-fledged constituent parsers have lately offered a conversion module that transforms a parse tree into dependency notation, because these notations have been most useful for different applications. Dependency relations are like severed components of predicate-argument structure or adjunct specifications, but they do not make the complete event structure explicit and it is surprisingly complex to extract the underlying PAS from dependency parser output.

As an example, consider the sentence:

> President Barack Obama will not meet the Dalai Lama during his five-day trip to the U.S. capital.

The outputs of SUPPLE, MiniPar, RASP, Stanford Parser, and the MuNPEx noun phrase chunker can be seen in Table 1.

SUPPLE
qlf=[meet(e26), adv(e26, not), time(e26, present), aspect(e26, simple), voice(e26, active), lobj(e26, e27), ne_tag(e27, offsets(165, 169)), name(e27, 'Lama'), ne_tag(e28, offsets(159, 164)), name(e28, 'Dalai'), realisation(e28, offsets(159, 164)), qual(e27, e28), det(e27, the), realisation(e27, offsets(155, 169)), realisation(e26, offsets(146, 169)), realisation(e26, offsets(146, 169))]
MiniPar
nn c_id=43, c_word=U.S., h_id=44, h_word=capital obj c_id=34, c_word=Lama, h_id=31, h_word=meet s c_id=1026, c_word=President, h_id=1031, h_word=meet
RASP
ncsubj meet:6 VV0 Obama:3 NP1 iobj meet:6 VV0 during:10 II dobj meet:6 VV0 Lama:9 NP1
Stanford Parser
args=[57, 63], kind=dobj, args=[57, 73], kind=prep args=[63, 59], kind=det, args=[63, 61], kind=nn
MuNPEx Noun Phrase Chunker
DET=his, HEAD=trip, HEAD_END=194, HEAD_START=190, MOD=five-day

Table 1: Excerpts from the output of the different parsers for the example sentence

Our PAX component normalises the different outputs into

[1]PAX, see http://www.semanticsoftware.info/pax

PAS as shown in Table 2.

RASP PAS	Obama – meet – trip
SUPPLE PAS	– meet – Lama
MiniPar PAS	Obama – meet – Lama
Stanford PAS	Obama – meet – Lama
Noun Phrase PAS	trip – be – five-day Dalai – be – Lama

Table 2: Output of three different parsers with extracted predicated-argument structures for the example sentence

As can be seen, the parsers have quite different opinions about the input sentence. This is not an exceptional, or special case, but typical for this task. Notice that we chose a rather simple sentence to demonstrate the different outputs. For more complex sentence structures, the difference in output is even greater and the extracted predicate-argument structures look quite different. Figure 1 gives an impression of the output of the MiniPar parser for a complete newspaper article.

3. Resource Description

Our PAX component is intended to be used as part of a larger processing pipeline, running after the individual parsers but before higher-level components that make use of PASs. It first collects the output of various parsers from the annotations added by them to a document. It then computes predicate-argument structures for each sentence as explained in Section 4. The predicate-argument structures for each sentence as extracted by PAX are then added as new annotations for this sentence and can be processed by other components in subsequent steps.

Supported Parsers. A variety of different parsers offer support for syntactic analysis of sentences. With this resource we try to extract predicate-argument structures using the output of different such parsers. Currently, we support MiniPar (Lin, 1998), RASP (Briscoe et al., 2006), SUPPLE (Gaizauskas et al., 2005), and the Stanford Parser (Klein and Manning, 2003a). In addition, we can extract PASs out of noun phrases, by making use of the output of a noun phrase chunker like MuNPEx.[2]

4. Design

We now describe in more detail how to extract predicate-argument structures from the output of different parsers as shown in Table 1.

Our PAS extractor is based on a set of rules for each of the three parsers. These rules determine which part of the parser output is considered the subject, verb, and object. Because of the different nomenclature and relations scheme of the parsers, this has to be done individually for each parser.

4.1. SUPPLE

For SUPPLE (Gaizauskas et al., 2005), the extraction process is quite straightforward. The parser outputs *semantic* relations, which comprise a logical subject and verb, and sometimes also a logical object. The PAS extractor therefore only has to filter out these elements from the output of SUPPLE. The coverage of SUPPLE is lower in comparison with other parsers. This is due to the philosophy of the parser (Gaizauskas et al., 2005): *"Rather than producing all possible analyses or using probabilities to generate the most likely analysis, the preference is not to offer a single analysis that spans the input sentence unless it can be relied on to be correct. This means that in many cases only partial analyses are produced, but the philosophy is that it is more useful to produce partial analyses that are correct than full analyses which may well be wrong or highly disjunctive."*

4.2. MiniPar

To obtain predicate-argument structures that represent the underlying sentence as closely as possible, we often have to choose between multiple candidates for the *object*. We employ a decision tree to select the grammatical structure to fill the *object* slot from the parser's output. If it exists and relates to the subject-verb pair we choose in this order: "obj," "obj1," "pred," and "pcomp-n."

Sometimes the *object* does not have a direct relation to the *verb* but an indirect link through another element in common like a "mod" construct. In this case we have to track down and identify this relation to find a representative object. A complex sentence can contain more than one subject and our extractor has to be able to handle them reasonably. Besides dealing with more than one "s" (subject) in one sentence, it can also handle conjunctions. Table 3 gives a general overview of the different conjunction types and how the extractor deals with them.

Sentence	**X and Y** buy a car.
Target PAS	X – buy – car Y – buy – car
Sentence	X **buys a car and sells his house**.
Target PAS	X – buy – car X – sell – house
Sentence	X buys **a car and a house**.
Target PAS	X – buy – car X – buy – house

Table 3: PAS extractor strategy for conjunctions

4.3. RASP

For RASP's version 3 (Briscoe et al., 2006) we developed a wrapper to be able to use it from within GATE. It calls the appropriate script and delivers the parser's output for further processing.

The strategy to find subject, verb, and object relations is to look for "ncsubj" occurrences in the parser output. They describe a subject together with the corresponding verb. To find a suitable object we often have to choose between different elements like "dobj," "iobj," "obj," or "xcomp." To obtain predicate-argument structures that accurately represent the underlying sentence, we use the following decision tree on what grammatical structure to use as *object*. If it exists and is related to the verb of the subject, we choose in this order: "obj," "dobj" if dependent of an "iobj," which itself relates to the relevant *verb*, "iobj," "dobj," and last "xcomp."

Besides dealing with more than one "ncsubj" in one sentence, we can also handle conjunctions. This has already

[2] Multi-lingual Noun Phrase Extractor (MuNPEx), http://www.semanticsoftware.info/munpex

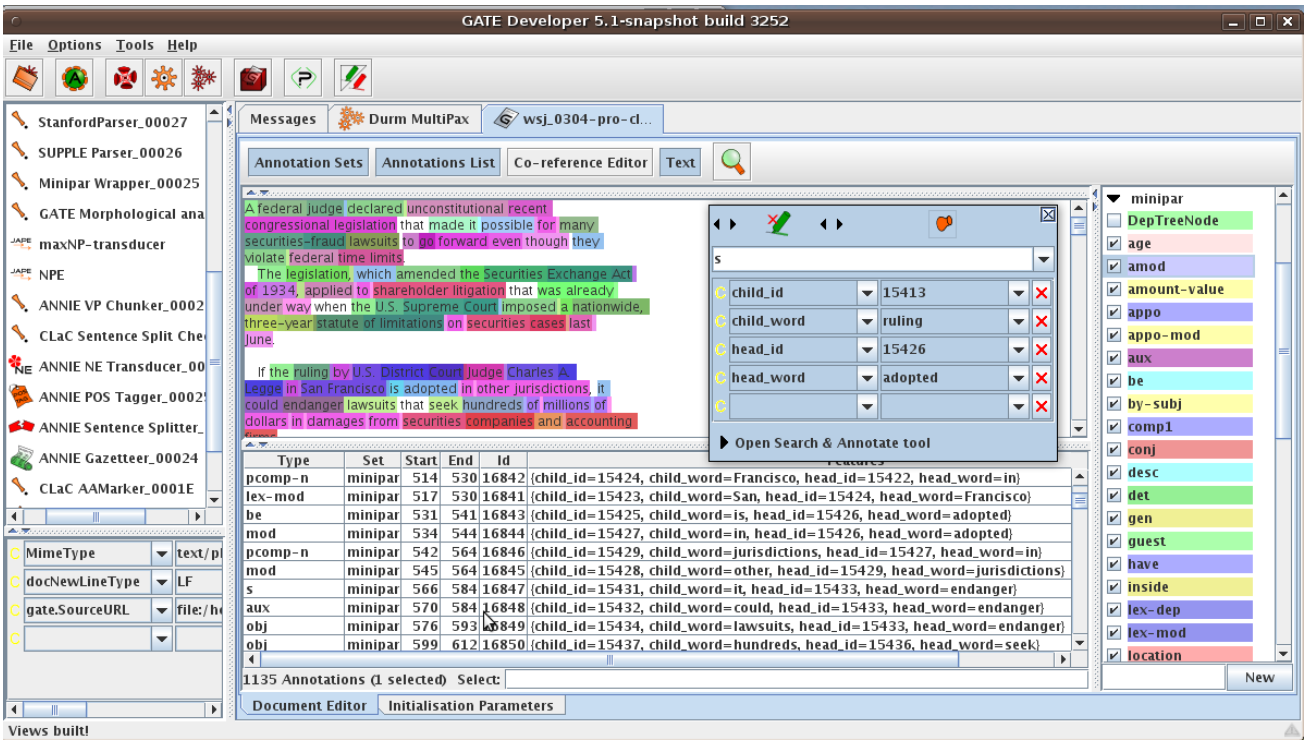

Figure 1: GATE screenshot of MiniPar results

been demonstrated for MiniPar in Table 3 and applies to RASP as well.

4.4. Stanford Parser

The Stanford Parser (Klein and Manning, 2003a), (Klein and Manning, 2003b) extracts dependency relations. We take all "nsubj" and "nsubjpass" elements for subjects and the associated predicates as verbs. For the object we take in this order: "dobj," "prepPobj," and "dep."
Conjunctions are already considered by the parser and there is no further processing needed from our side.

4.5. MuNPEx Noun Phrases

Each noun phrase that contains a modifier is a candidate for a predicate-argument structure. For example, the noun phrase "the rich king" contains the same information as the PAS "king – be – rich". Adding the noun phrase predications generates additional PASs that can be especially useful for certain task like comparing documents' content based on predicate-argument structures or for recognizing textual entailment (e.g., the RTE[3] tasks) between statements.

5. Implementation

Our resource is implemented as a component for the *General Architecture for Text Engineering* (GATE) (Cunningham et al., 2002). Figure 2 shows example output of the PAX component with the detected PASs for all supported parsers. For each parser x, a new annotation set of type xParserPaX is added to the document. If the component detects a predicate-argument structure in the output of the selected parser, the sentence containing the PAS is annotated and "sub", "obj", and "verb" properties are added to the annotation. In addition, we try to detect simple negations within the sentences like "not" or "never" (Figure 2).

6. Evaluation

To evaluate our PAX component, we selected an article from the *Wall Street Journal* and annotated it manually with predicate-argument structures. The structure of the sentences was particularly complex with most of the time three or more PASs per sentence. For simple sentences of the shape "subject, verb, object" all parsers perform well and we can extract the predicate-argument structures reliably from the parsers' output. Therefore we are interested in the most difficult cases only. We excluded the noun phrase PAS extraction from this evaluation since it is a special case also yielding different types of errors. Table 4 gives an overview of the performance of the different parsers with correctly extracted PAS, wrong PAS, and partially correct PAS, where partially means for example that the object was not found or an indirect object instead of a direct one was found.
Some errors like unresolved pronouns, e.g. "that," "he," "who" or "myself" were not considered errors of the PAS extraction but need to be dealt with in the future, although for some parsers we are already able to resolve these constructs. Another possible source of errors are verb phrases like "declare unconstitutional," or "prevent s.o. from doing s.th." If we insist on having only one term as a predicate we need to decide which verb reflects the intended meaning of the PAS best.
Noun Phrases with modifiers can not always be converted to predicate-argument structures. For example, it works fine with "the elected President" $\rightarrow$ "President – be – elected"; but not for "last year's President" $\not\rightarrow$ "President – be – year".

[3]RTE, see http://www.nist.gov/tac/2009/RTE/

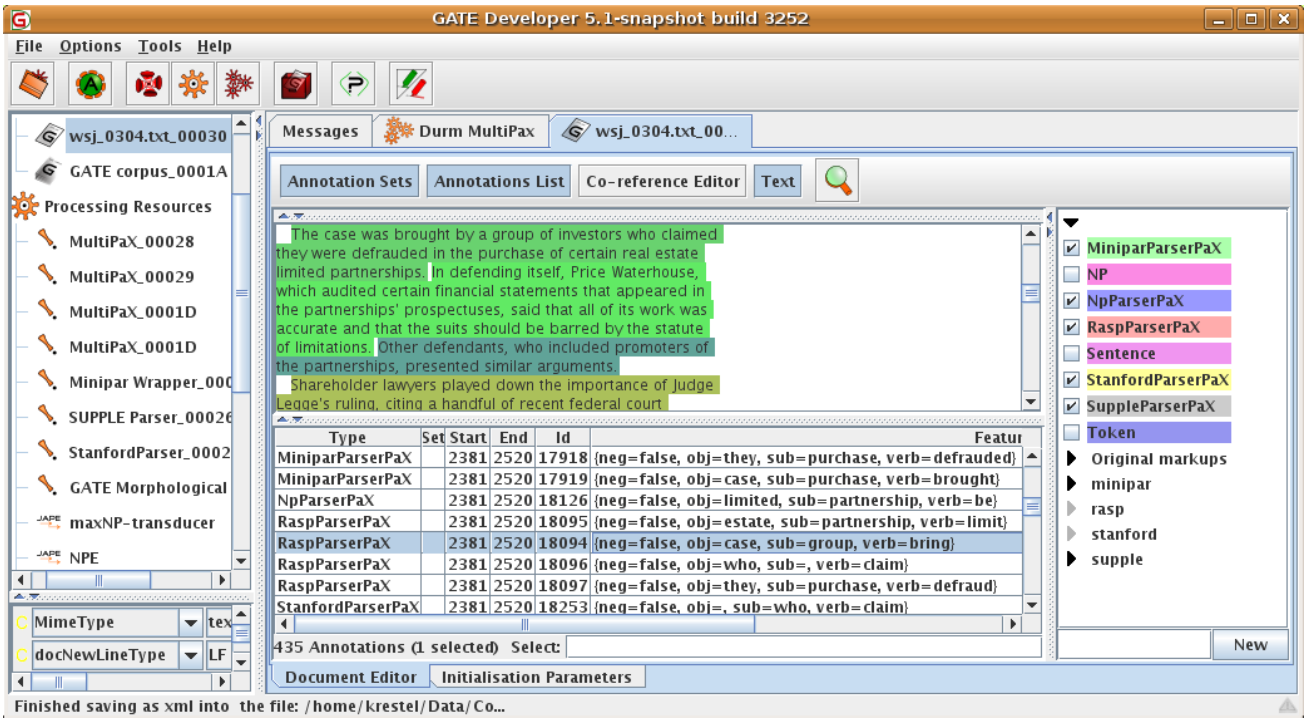

Figure 2: PAX results in GATE showing subject-verb-object triples extracted from MiniPar, RASP, Stanford, and MuNPEx

Sent	No of PAS	SUPPLE			MiniPar			RASP			Stanford		
		C	P	F	C	P	F	C	P	F	C	P	F
1	4	-	2	-	-	2	-	1	2	1	1	2	1
2	4	1	-	-	-	2	-	2	2	-	2	2	-
3	3	-	-	-	-	2	-	2	1	-	1	1	-
4	3	-	-	-	-	-	1	2	-	-	-	-	-
5	4	-	1	-	-	2	-	-	2	1	-	2	-
6	1	-	1	-	-	1	-	-	1	1	-	1	-
7	4	1	-	-	-	2	-	-	3	-	-	4	-
8	5	2	1	-	3	1	-	-	4	1	3	1	-
9	3	-	-	-	-	1	2	1	2	1	1	1	-
10	3	-	1	-	-	1	-	-	-	-	-	1	1
11	6	1	2	-	3	1	-	2	4	-	1	1	2
12	3	1	-	-	-	2	-	-	3	1	-	-	-
13	5	1	1	-	2	1	-	1	2	-	1	1	-
14	2	-	-	-	-	1	-	-	2	-	-	1	-
15	3	-	1	-	-	1	-	-	2	-	-	2	-
16	2	-	1	-	-	-	-	-	1	2	-	1	1
17	4	1	1	-	-	3	-	-	4	-	1	1	1
18	3	-	-	-	-	2	-	-	3	-	-	3	-
19	3	1	-	-	-	1	-	2	-	-	1	1	-
20	3	-	1	-	-	2	-	1	-	2	1	2	-
21	2	-	1	-	-	2	-	-	2	-	-	2	-
22	1	-	1	-	-	1	-	-	1	-	-	1	-
23	4	-	-	-	2	-	1	2	1	1	-	1	1
24	0	-	-	-	-	-	-	-	-	-	-	-	1
∑	75	9	15	-	8	31	3	16	42	11	13	32	8
Recall		0.32			0.52			0.77			0.60		
Precision		1.0			0.93			0.84			0.85		

Table 4: Results for the four parsers: C=correct, P=partially correct, F=false

7. Conclusion & Future Work

We described a strategy to extract predicate-argument structures from the output of different parsers and its implementation in the PAX component for GATE. PASs can be used to represent content and make it comparable. Finding similar content (e.g., for text summarization, paraphrase detection) or entailment (RTE, inferences) are some of the application areas of predicate-argument structures. Our component creates a common result structure for the different parsers and thereby allows downstream analysis components to work independently of a concrete parser's output. This simplifies the setup significantly of experiments where the impact of different parsers on the overall application performance needs to be measured.

In the future we want to introduce a voting algorithm to identify the best predicate-argument structures for each sentence based on the output of multiple parsers.

8. References

E. Briscoe, J. Carroll, and R. Watson. 2006. The Second Release of the RASP System. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*.

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 2002. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proc. of the 40th Anniversary Meeting of the ACL*.

R. Gaizauskas, M. Hepple, H. Saggion, M. A. Greenwood, and K. Humphreys. 2005. SUPPLE: A practical parser for natural language engineering applications. In *Proc. of the 9th Intl. Workshop on Parsing Technologies (IWPT2005)*, Vancouver.

Dan Klein and Christopher D. Manning. 2003a. Accurate unlexicalized parsing. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 423–430, Morristown, NJ, USA. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2003b. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems*, volume 15. MIT Press.

Dekang Lin. 1998. Dependency Based Evaluation of MINIPAR. In *Proceedings of the Workshop on the Evaluation of Parsing Systems, First International Conference on Language Resources and Evaluation*.

Paola Merlo and Eva Esteve Ferrer. 2006. The notion of argument in prepositional phrase attachment. *Computational Linguistics*, 32(3):341–378.