Interoperability and Customisation of Annotation Schemata in Argo

Rafal Rak, Jacob Carter, Andrew Rowley, Riza Theresa Batista-Navarro, Sophia Ananiadou
National Centre for Text Mining and School of Computer Science, University of Manchester
131 Princess Street, Manchester, M1 7DN, UK
{rafal.rak,jacob.carter,andrew.rowley,riza.batista,sophia.ananiadou}@manchester.ac.uk

Abstract

The process of annotating text corpora involves establishing annotation schemata which define the scope and depth of an annotation task at hand. We demonstrate this activity in Argo, a Web-based workbench for the analysis of textual resources, which facilitates both automatic and manual annotation. Annotation tasks in the workbench are defined by building workflows consisting of a selection of available elementary analytics developed in compliance with the Unstructured Information Management Architecture specification. The architecture accommodates complex annotation types that may define primitive as well as referential attributes. Argo aids the development of custom annotation schemata and supports their interoperability by featuring a schema editor and specialised analytics for schemata alignment. The schema editor is a self-contained graphical user interface for defining annotation types. Multiple heterogeneous schemata can be aligned by including one of two type mapping analytics currently offered in Argo. One is based on a simple mapping syntax and, although limited in functionality, covers most common use cases. The other utilises a well established graph query language, SPARQL, and is superior to other state-of-the-art solutions in terms of expressiveness. We argue that the customisation of annotation schemata does not need to compromise their interoperability.

Keywords: annotation schemata, tool interoperability, annotation platforms

1. Introduction

Annotation schemata define the scope and technical details of an annotation task and enable the creation of derived work in the form of other language resources such as lexica and ontologies. The Unstructured Information Management Architecture (UIMA) (Ferrucci and Lally, 2004) provides a framework for defining expressive annotation schemata, known as type systems, and facilitates the interoperability of analytics that abide by such type systems. UIMA is an Apache Software Foundation open-source project and is commonly utilised for the processing of language resources. Notable examples of UIMA component repositories include U-Compare (Kano et al., 2010), DKPro (Gurevych et al., 2007), cTAKES (Savova et al., 2010), JCoRe (Hahn et al., 2008), and BioNLP-UIMA (Baumgartner et al., 2008).

UIMA defines common interfaces and data structures that are exchanged between analytics (or processing components). Data structures strictly conform to one or more type systems that are used in a processing pipeline. UIMA provides a built-in, top-level type system that consists of primitive types (e.g., integer, string, boolean) as well as complex types, i.e., types that contain features. Features may be any of the primitive types as well as references to other complex types. Developers are free to extend any of the complex types, which makes UIMA flexible enough to encode schemata for a variety of syntactic and semantic annotation tasks including syntax parse trees, dependency graphs, named entities, entity relationships, coreferences and discourse analyses. U-Compare is an example of a library of components that conform to a single, all-in-one type system (consisting of nearly 300 types) that supports all of the aforementioned tasks.

On the other hand, the freedom of defining own type systems contributes to the creation of closed component repositories where components are interoperable (support same type systems) within a repository, but not outside of it, despite sharing conceptual similarities. For instance, the coreference phenomenon may be structurally represented as a linked list or an array of coreferring mentions. A component producing coreferences as linked lists cannot be then directly connected to a component that consumes coreferences as arrays.

In this paper we demonstrate Argo, a UIMA-based tool that supports multiple, heterogenous type systems and addresses the problem of type system alignment. Argo (Rak et al., 2012) is a Web-based workbench for the analysis of textual resources, which facilitates both automatic and manual annotation. The workbench is equipped with an ever-increasing library of elementary processing components that can be arranged by users to form meaningful processing units (or workflows). The processing components range from data serialisers and deserialisers to syntactic and semantic annotators. Automatic processing may be manually validated owing to the Manual Annotation Editor component that, if present in a workflow, pauses the execution of the workflow and expects manual intervention from a user. The user may then create new or modify existing annotations through a flexible graphical user interface.

The type system alignment is accomplished in Argo by including specialised analytics in workflows. The workbench currently features two such components, namely, Type Mapper and SPARQL Annotation Editor. The choice between the two boils down to trading simplicity for functionality.

We also introduce a graphical type system editor, a new feature of Argo that allows users to create ad hoc annotation schemata. The editor features the semi-automatic recognition of types from user-supplied data in several widely-used formats.

The remainder of this paper is organised as follows. The next section provides the overview of Argo. Section 3 in-

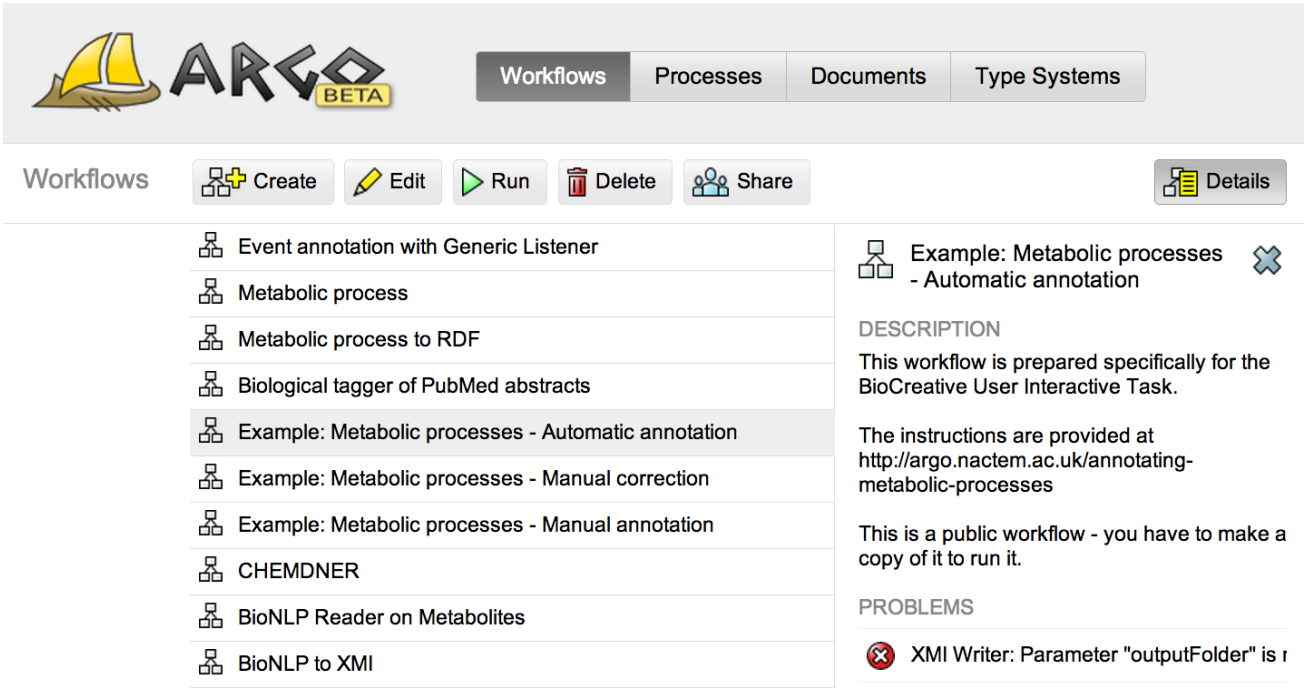

Figure 1: Main interface of Argo. The selected view shows a list of user's workflows.

troduces the two type mapping analytics and Section 4 describes the type system editor, which is followed by a use case in Section 5. Section 6 briefly presents related work, whereas Section 7 concludes the paper.

2. System Overview

Argo is a multi-user and collaborative system. The Web-based graphical interface provides users with access to creating workflows, executing them and tracking their progress, a document storage (a user's space for uploading files for processing and downloading results), and the newly introduced type system editor. A screenshot of the main interface is shown in Figure 1.

Users create workflows using a diagramming editor shown in Figure 2. Each block represents a processing component and each connector represents the flow of data between components. Argo enables multiple branching and merging points in a workflow, thus logically parallelising flows. The components are categorised into collection readers, analytics, and writers (a special subgroup of analytics). Readers and writers deserialise and serialise, respectively, data in various formats. They include generic formats such as plain text, XMI (XML Metadata Interchange) and RDF, as well as domain- and task-specific formats. For instance, Kleio Search fetches data from a remote Web service, CRF Writer produces a statistical model file, and Reference Evaluator outputs a tab-separated values file containing effectiveness evaluation metrics. Analytics are processing components whose role is to modify the input data structures and pass them onto the succeeding components in a workflow. They include syntactic, semantic and utility components that range from tokenisers and sentence detectors to constituent tree and dependency parsers to various named entity and relationship recognisers to external resource linkers to customisable machine learning-based taggers for advanced users. Each workflow begins with a collection reader that is followed by one or more analytics and may terminate with one or more writers. An example of such a workflow is shown in Figure 2.

3. Type System Interoperability

Argo currently features nearly a dozen publicly available type systems. They include the comprehensive U-Compare type system, several generic systems that encode structures such as events (complex relationships) and machine learning-related types, as well as domain-specific type systems such as biological and chemical typologies.

The available components in Argo support one or more of the type systems. A problem arises when a workflow involves components which do not share the same type systems and yet are meant to communicate, i.e., pass data structures between each other. For instance, the BioC Writer component serialises biologically relevant entity and relationship annotations in a specific, XML-based format and supports only the purpose-built BioC type system. In order to save data from, e.g., the Anatomical Entity Tagger component that produces named entities defined in the U-Compare type system, the annotations need to be transcribed from one format to the other. In platforms such as U-Compare, this has been resolved programmatically by building components that transcribe annotations from one type system to another. This solution, however, requires software development skills and is not scalable with a growing number of type systems.

In order to alleviate this problem, solutions based on (cascaded) finite state transducers, or transcription rules, have been introduced. For each transcription rule, a pattern is matched against a stream of existing feature structures, and

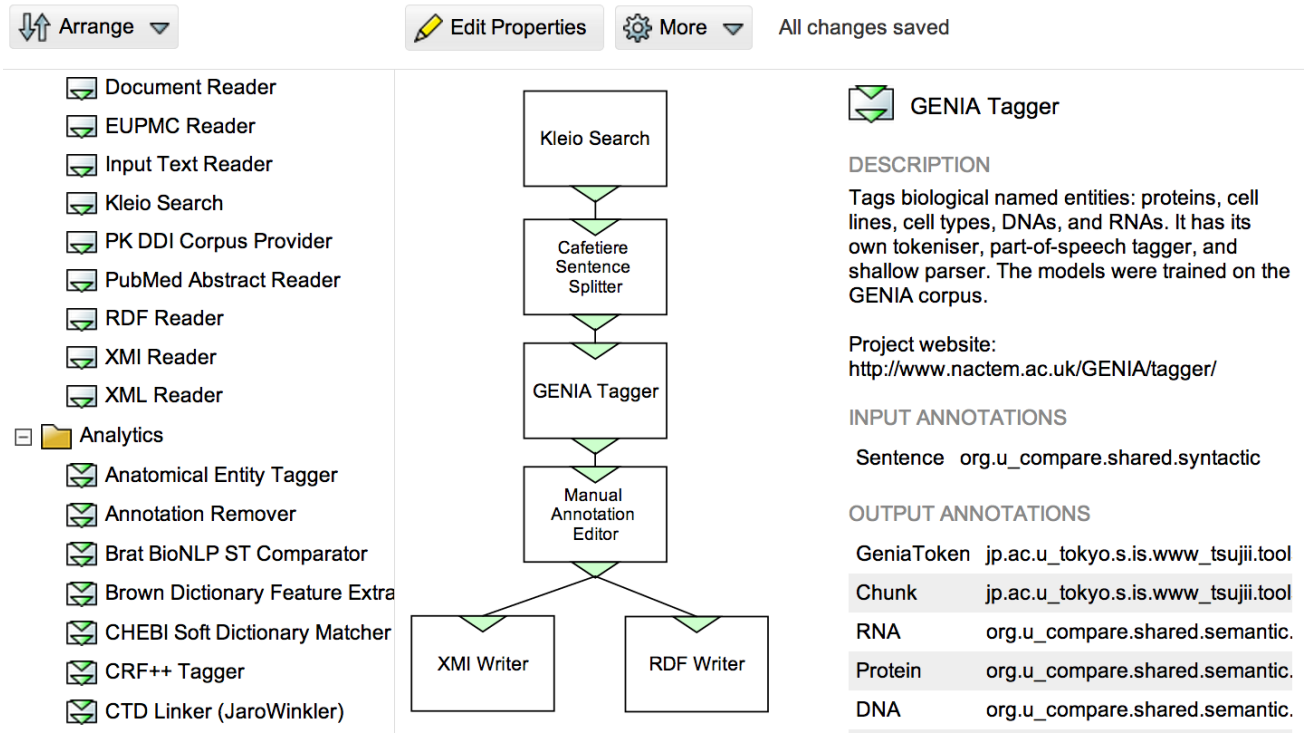

Figure 2: Workflow diagramming window.

if a match is found, new structures are created.

Ruta (formerly known as TextMarker) (Kluegl et al., 2009) is currently the most comprehensive UIMA analytic that includes its own language for creating rules and offers Eclipse-based tooling. Similar rule-based analytics include uima-map (Hernandez, 2012) that uses a tailored syntax encoded in XML, and PEARL (Pazienza et al., 2012) that utilises SPARQL-like constructs in otherwise proprietary syntax.

In Argo, the type system alignment is supported by two components, Type Mapper and SPARQL Annotation Editor. The former is a dedicated rule-based analytic for transcribing feature structures between types, whereas the latter is a general-purpose annotation editor that utilises SPARQL for manipulating feature structures (Rak and Ananiadou, 2013).

Type Mapper, although similar in concept to the other rule-based solutions, provides a very simplistic syntax for rudimentary type conversions. Despite its limitations, its functionality covers most common uses cases. The most basic transcription solely requires specifying the names of two types. The analytic will create a new feature structure of a destination type for each feature structure of a source type and automatically copy common features between the structures. Additional functionality includes conditional transcriptions, custom transcriptions of features or feature paths, and executing predefined functions. Examples of the Type Mapper's syntax and capabilities are given in Figure 3.

The SPARQL Annotation Editor is a processing component that enables the transcription of annotations using a well-defined and widely-used language, SPARQL 1.1, that

```
# Example 1: Basic transcription
com.example.LabelledEntity -> org.example.NamedEntity;

# Example 2: Conditional transcription with feature mapping
com.example.Person where confidence > 0.8
   -> org.example.NamedEntity,
   "Person" -> category,
   confidence -> metaData/confidence
```

Figure 3: Examples of type transcriptions with Type Mapper.

allows for both querying and manipulating RDF graphs. The component facilitates the transcription by 1) representing data structures as an RDF graph, 2) performing a user-defined SPARQL query that modifies the graph, and 3) converting the modified graph back to the UIMA representation. A user-defined query involves statements that rewrite annotations in the source type system to the destination type system. Figure 4 shows examples of SPARQL queries that perform the same transcriptions that are shown in Figure 3. In comparison, SPARQL syntax is more verbose than the Type Mapper syntax as the SPARQL Annotation Editor does not enjoy the implicit automatic functionality the Type Mapper has. On the other hand, the SPARQL Annotation Editor does not suffer the limitations of the other component or any other rule-based transcriptor. SPARQL's random data access, as well as data modification, manipulation and filtering capabilities make the language far more expressive. The examples and analysis of advanced transcriptions using the SPARQL Annotation Editor, such as one-to-many and chain-array conversions, are given in (Rak and Ananiadou, 2013).

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX tcas: <uima:ts:uima.tcas.>
PREFIX org: <uima:ts:org.example.>
PREFIX com: <uima:ts:com.example.>

# Example 1: Basic transcription
INSERT {
  _:namedEntity a org:NamedEntity ;
    tcas:Annotation:begin ?begin ;
    tcas:Annotation:end ?end .
  ?view rdfs:member _:namedEntity .
}
WHERE {
  ?view a <uima:aux:View> ;
    rdfs:member ?labelledEntity .
  ?labelledEntity a com:Person ;
    tcas:Annotation:begin ?begin ;
    tcas:Annotation:end ?end .
}

# Example 2: Conditional transcription with additional
#            feature mapping
INSERT {
  _:metaData a org:MetaData ;
    org:MetaData:confidence ?confidence .
  _:namedEntity a org:NamedEntity ;
    tcas:Annotation:begin ?begin ;
    tcas:Annotation:end ?end ;
    org:NamedEntity:metaData _:metaData .
  ?view rdfs:member _:namedEntity .
}
WHERE {
  ?view a <uima:aux:View> ;
    rdfs:member ?labelledEntity .
  ?labelledEntity a com:Person ;
    tcas:Annotation:begin ?begin ;
    tcas:Annotation:end ?end .
    com:Person:confidence ?confidence
  FILTER(?confidence > 0.8)
}
```

Figure 4: Examples of type transcription with SPARQL Annotation Editor.

If the limited functionality is of no concern, users would choose the Type Mapper over the SPARQL Annotation Editor due to its syntax's brevity as well as speed, since the latter incurs costly conversions between data representations.

4. Type System Editor

To further facilitate the ad hoc creation of annotation schemata, we introduced a type system editor that features a graphical user interface for easy definition of new types. The editor is available directly in the Argo application and its primary purpose is to allow human annotators to define their custom, task-specific type systems that they can use for manual annotation.

The editor fully supports the UIMA specification of defining type systems. This includes the extension of existing types, the definition of primitive and referential features, and the description of types and their features. As an example, Figure 5 demonstrates the definition of type `LinkedEntity` and three of its subtypes, `Process`, `Chemical`, and `GeneOrGeneProduct`. The `LinkedEntity` type itself extends the built-in `Annotation` type (available in the `uima.tcas` namespace). The selected type defines four features, three of which (`sofa`, `begin` and `end`) come from the ancestors' types. The features `begin`, `end` and `id` are primitive features (respectively, two integers and a string), whereas the `sofa` feature has the complex value type, `uima.cas.Sofa`, that represents a subject of analysis by including the original text as well as its meta data.

To accelerate the definition of custom type systems, we have been developing automatic type recognition plugins whose purpose is to create an initial version of a type system directly from data supplied by a user. Currently, we support IOB/IOBES CoNLL column format, BioNLP Shared Task event definition format, and Document Type Definition (DTD).

5. Use Case

We demonstrate the interoperability and customisation of annotation schemata by invoking the use of Argo in the recently concluded BioCreative IV User Interactive Track. The defined, biology-related task involved the annotation of concepts relevant to metabolic processes, namely, chemical compounds (CCs), genes or gene products (GGPs) and action words. In addition to the tagging of text spans corresponding to these concept types, the human annotators were asked to link the text spans to identifiers in external resources where possible (Rak et al., 2013).

The main requirement of the track was the capability to assist a domain expert in the annotation task by the provision of annotations automatically generated by text mining tools. The workflow in Argo involved two named entity recognisers (NERs), namely, GENIA Tagger (Tsuruoka et al., 2005) for annotating GGPs and OscarMER (Kolluru et al., 2011) for annotating CCs and action words. The two NERs comply with two different type systems, respectively, the U-Compare and Oscar type systems.

To facilitate the harmonisation of the annotations coming from the different components in the workflow, we defined the Metabolic Process type system (presented in Figure 5) specifically for this annotation task[1]. This simple type system has the `LinkedEntity` type for any span-of-text annotation, whose `id` feature can be assigned an identifier from the relevant external resource. It is extended by the more specific types `Process`, `Chemical` and `GeneOrGeneProduct`, corresponding to our concept types of interest. With the inclusion of a SPARQL Annotation Editor component in the workflow, the complex annotations from the OscarMER and GENIA Tagger components (involving a large number of features which were of no interest to human annotators) were aligned with the simplified, human annotator-friendly Metabolic Process type system.

By designing a simple, intuitive, custom type system we eliminated the need to inelegantly merge two completely disparate type systems and allowed Argo to present to the annotators only those annotation types and features which were relevant to the task.

[1]The Metabolic Process type system had been developed outside Argo since the editor had not been available at that time.

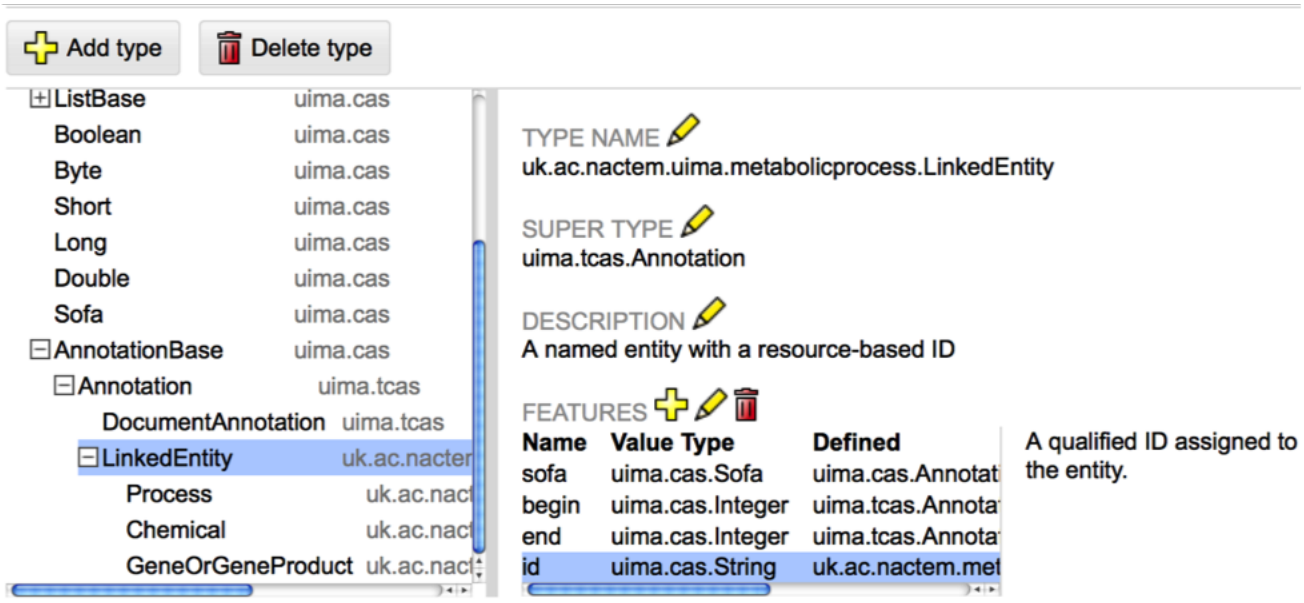

Figure 5: Type system editor.

6. Related Work

Other notable natural language processing platforms that feature rich graphical user interfaces include U-Compare (Kano et al., 2010) and GATE (Cunningham et al., 2002), both available as standalone Java applications. The former, similarly to Argo, supports UIMA, but involves only a handful of programmatic annotation transcriptions between type systems and does not include a type system editor. Argo additionally features user collaboration and interactive components. GATE includes a suite of text processing and annotation tools and allows users to define their own annotation schemata using only direct XML Schema syntax. The expressiveness of types in GATE is also inferior to those in UIMA, e.g., GATE types do not support type inheritance nor referential features. A comprehensive comparison of UIMA, GATE and several other frameworks is given by Bank and Schierle (Bank and Schierle, 2012).

Like Argo, Egas (Campos et al., 2013) is a Web-based, collaborative annotation tool that allows a superuser to define annotation types; however, the types are limited to named span-of-text annotations and named binary relationships between them. In contrast, Argo fully supports UIMA type system specification which accommodates types consisting of both primitive and referential features and therefore facilitates the encoding of schemata for a broader variety of annotation tasks.

7. Conclusions and Futre Work

Fixed and well-defined annotation schemata facilitate the interoperability of processing resources. However, unified, universal and all-embroiling schemata are unlikely to ever materialise due to variations in requirements, scope, conceptualisation and applicability. Additionally, manual annotation requires less complex and more human annotator-friendly schemata. Argo addresses these issues by introducing annotation transcription components that ultimately facilitate the alignment of schemata, and a graphical type system editor that allows a human annotator to define annotation task-tailored types whilst preserving structural integrity and reusability of annotations guaranteed by the UIMA framework.

As future work, in order to enhance users' experience, we are planning to provide a graphical user interface for the Type Mapper, which will release users from having to learn the type mapping syntax, as well as introduce more formats that the type system editor is able to automatically create schemata from.

8. Acknowledgements

This work was partially supported by Europe PubMed Central funders (led by Wellcome Trust).

9. References

Bank, M. and Schierle, M. (2012). A Survey of Text Mining Architectures and the UIMA Standard. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, May. European Language Resources Association (ELRA).

Baumgartner, W. A., Cohen, K. B., and Hunter, L. (2008). An open-source framework for large-scale, flexible evaluation of biomedical text mining systems. *Journal of biomedical discovery and collaboration*, 3:1+, January.

Campos, D., Lourencço, J., Nunes, T., Vitorino, R., Domingues, P., Matos, S., and Oliveira, J. L. (2013). Egas Collaborative Biomedical Annotation as a Service. In *Proceedings of the Fourth BioCreative Challenge Evaluation Workshop vol. 1*, pages 254–259, October.

Cunningham, H., Maynard, D., Bontcheva, K., and Tablan, V. (2002). GATE: A framework and graphical development environment for robust NLP tools and applications.

In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*.

Ferrucci, D. and Lally, A. (2004). UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering*, 10(3-4):327–348, September.

Gurevych, I., Mühlhäuser, M., Müller, C., Steimle, J., Weimer, M., and Zesch, T. (2007). Darmstadt Knowledge Processing Repository Based on UIMA. In *Proceedings of the First Workshop on Unstructured Information Management Architecture at Biannual Conference of the Society for Computational Linguistics and Language Technology*, Tübingen, Germany, April.

Hahn, U., Buyko, E., Landefeld, R., Mühlhausen, M., Poprat, M., Tomanek, K., and Wermter, J. (2008). An Overview of JCORE, the JULIE Lab UIMA Component Repository. In *Language Resources and Evaluation Workshop, Towards Enhanc. Interoperability Large HLT Syst.: UIMA NLP*, pages 1–8.

Hernandez, N. (2012). Tackling interoperability issues within UIMA workflows. In Chair), N. C. C., Choukri, K., Declerck, T., Doan, M. U., Maegaard, B., Mariani, J., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, May. European Language Resources Association (ELRA).

Kano, Y., Dorado, R., McCrochon, L., Ananiadou, S., and Tsujii, J. (2010). U-Compare: An integrated language resource evaluation platform including a comprehensive UIMA resource library. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC 2010)*, pages 428–434.

Kluegl, P., Atzmueller, M., and Puppe, F. (2009). TextMarker: A Tool for Rule-Based Information Extraction. In Chiarcos, C., de Castilho, R. E., and Stede, M., editors, *Proceedings of the Biennial GSCL Conference 2009, 2nd UIMA@GSCL Workshop*, pages 233–240. Gunter Narr Verlag.

Kolluru, B., Hawizy, L., Murray-Rust, P., Tsujii, J., and Ananiadou, S. (2011). Using workflows to explore and optimise named entity recognition for chemistry. *PLoS ONE*, 6(5):e20181.

Pazienza, M. T., Stellato, A., and Turbati, A. (2012). PEARL: ProjEction of Annotations Rule Language, a Language for Projecting (UIMA) Annotations over RDF Knowledge Bases. In Chair), N. C. C., Choukri, K., Declerck, T., Doan, M. U., Maegaard, B., Mariani, J., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, May. European Language Resources Association (ELRA).

Rak, R. and Ananiadou, S. (2013). Making UIMA Truly Interoperable with SPARQL. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 88–97, August.

Rak, R., Rowley, A., Black, W., and Ananiadou, S. (2012). Argo: an integrative, interactive, text mining-based workbench supporting curation. *Database : The Journal of Biological Databases and Curation*, page bas010.

Rak, R., Batista-Navarro, R., Rowley, A., Carter, J., and Ananiadou, S. (2013). Customisable Curation Workflows in Argo. In *Proceedings of the Fourth BioCreative Challenge Evaluation Workshop vol. 1.*, pages 270–278.

Savova, G. K., Masanz, J. J., Ogren, P. V., Zheng, J., Sohn, S., Kipper-Schuler, K. C., and Chute, C. G. (2010). Mayo clinical Text Analysis and Knowledge Extraction System (cTAKES): architecture, component evaluation and applications. *Journal of the American Medical Informatics Association : JAMIA*, 17(5):507–513, September.

Tsuruoka, Y., Tateisi, Y., Kim, J.-D., Ohta, T., McNaught, J., Ananiadou, S., and Tsujii, J. (2005). Developing a Robust Part-of-Speech Tagger for Biomedical Text. In *Advances in Informatics - 10th Panhellenic Conference on Informatics*, volume 3746 of *LNCS*, pages 382–392. Springer-Verlag, Volos, Greece, November.