Abstracting the types away from a UIMA type system

Karin Verspoor, William Baumgartner Jr., Christophe Roeder, and Lawrence Hunter

Center for Computational Pharmacology
University of Colorado Denver School of Medicine
MS 8303, PO Box 6511, Aurora, CO 80045
Karin.Verspoor@ucdenver.edu, William.Baumgartner@ucdenver.edu,
Chris.Roeder@ucdenver.edu, Larry.Hunter@ucdenver.edu

Abstract. This paper discusses the design of a "generic" UIMA type system, in which the type system itself is a lightweight meta-model and genericity is achieved by referencing an external object model containing the full semantic complexity desired for the application. This allows arbitrarily complex semantic types to be manipulated and created by UIMA components, without requiring representation of domain-specific types within the type system itself. A meta-model type system further allows for the definition of a single type system that can be used in a wide array of contexts, supporting an even wider array of semantic types.

1 Introduction

The Unstructured Information Management Architecture (UIMA)[6, 1] is a framework that supports the definition and integration of software modules that perform analysis on unstructured data such as text documents or videos. UIMA facilitates assignment of structure to these unstructured artifacts by providing a means to link the artifacts to meta-data that describes them[7]. Two key components of UIMA are the Common Analysis Structure, or CAS, and the UIMA type system. The CAS is the basic data structure in which both the unstructured information being analyzed and the meta-data inferred for that information are stored. The UIMA type system is a declarative definition of an object model, and serves two main purposes: (a) to define the kinds of meta-data that can be stored in the CAS; and (b) to support description of the behavior of a processing module, or *analytic*, through specification of the types it expects to be in an input CAS and the types it inserts as output.

There are many strategies for defining a type system in UIMA. UIMA itself does not include a particular set of types that developers must use, other than the high-level notion of a `uima.jcas.tcas.Annotation`, the basic type of all meta-data that refers to a region of an artifact (e.g. a span of text). Users must define their own domain and application specific type systems. This leads to the problem of type system divergence, which in turn stands in the way of seamless plug and play compatibility among UIMA components. The irony here

is that the ease with which UIMA can be customized actually works against the overall interoperability goals of the UIMA framework. Representing types in different forms, or even under different package names/namespaces, can easily break interoperability between UIMA components. This flexibility can result in large type systems that tie analysis engines down to specific domain models, are cumbersome to maintain in the face of shifting domain semantics, and prevent straightforward reuse in diverse applications.

In this paper, we will consider the semantics of the UIMA type system itself. We identify two extremes for the definition of a UIMA type system. Any given UIMA type system may lie at some point within these extremes.

1. **Fully specified semantic model:** A type system in which all of the relevant semantic types of the application domain for the meta-data of the system are specified within the UIMA type system.
2. **Generic meta-model:** A type system in which only very high-level semantic types are specified within the UIMA type system, while the meta-data semantics is defined externally to UIMA.

We argue in this paper for preferring type systems that are strongly generic, that abstract the domain semantics away from the UIMA framework. The types in our UIMA type system allow us to model knowledge captured by publicly-curated ontologies and other external resources by referencing the terms in those resources. A strength of this approach is the ability to take advantage of community-wide consensus already achieved in the external resources we use.

What we propose is essentially a type system without types, or more specifically with a minimal number of explicitly defined types. The approach purposefully creates a layer of abstraction between the type system itself and the semantics that it represents. We provide an example of such a type system, and show that this has the advantage of application flexibility while facilitating component reuse and sharing.

2 UIMA type system as a meta-model

The perspective we endorse is to consider the UIMA type system as a meta-model for the semantics of the structured content of an artifact. We adopt the term *meta-model* carefully, following its use in software engineering, where it indicates a model of a model. For instance, UML, the Unified Modeling Language, is a model defining how to express the structure of a software program. In the type system we describe here, the underlying model represents the domain semantics, and the meta-model is a description of how to characterize that domain semantics in UIMA. It follows from this perspective that the domain semantics itself should not be a part of the UIMA type system.

Most UIMA type systems have a generic "upper level" in the class hierarchy extending the built-in `Annotation` type with classes that capture the basic distinctions among kinds of meta-data to be inserted in the CAS. For

UIMA systems focused on representing the core information contained in textual documents for natural language processing, this normally consists of types such as `SyntacticAnnotation`, with subtypes for tokens, sentences, etc., and `ConceptAnnotation` as the umbrella class for specific kinds of entities and relations. These upper level types are seen as forming the application-independent core of the type system, easily shareable among analytics, with extensions used to capture more application-specific distinctions[10, 13]. Thus, an application in the biomedical domain, for instance, might add subtypes to `ConceptAnnotation` for particular entity types, such as proteins, organisms, and so forth.

We propose a type system that is restricted to this upper level, where all domain type distinctions can be made in resources external to UIMA, such as ontologies or databases. These specific distinctions are then referenced from a generic type within UIMA, where the relevant domain type is indicated as the value of a feature in the generic type, rather than as a class in the type system model. This approach gives the advantage of having a UIMA application that is robust to, and up to date with, changes in the domain semantics.

Others have also proposed UIMA type systems that include a connection to external resources[10, 3, 14]. However, these type systems are still presumed to make the core semantic distinctions of the application domain within the UIMA type system itself, so that for instance an entity that is linked to an identifier in the UniProtKB database (http://www.uniprot.org), a database which stores information about proteins, would be annotated with a UIMA type of `Protein`, extending `ConceptAnnotation`. In our view, such domain-specific types do not belong in the UIMA type system at all.

3 The CCP UIMA type system

In the Center for Computational Pharmacology (CCP), we have implemented a meta-model type system that is used in all of our UIMA applications, such as our tools for biomedical natural language processing[4, 2], and particularly the OpenDMAP system[12], a system for concept recognition in text. The CCP UIMA type system consists of a lightweight annotation hierarchy, where the domain semantics is captured through pointers into external resources.

In OpenDMAP, we take advantage of community-curated ontologies that have been developed for and by experts in our application domain, biomedicine. An ontology is an explicit specification of the concepts and relations employed in a given domain of interest[9]. In the biomedical domain, there has been a concerted effort to develop large-scale, shared, representations of domain concepts for the purpose of enabling linkages across databases and supporting consistent labeling of diverse data. Large amounts of time and money have been invested in achieving community consensus in resources such as the Gene Ontology[8]. Our meta-model approach is able to directly build on these efforts.

A prototype meta-model type system was used in [2], however our proposed CCP UIMA type system can be seen in Figure 1. This reflects the model we are working towards. It is very close to what we currently have implemented,

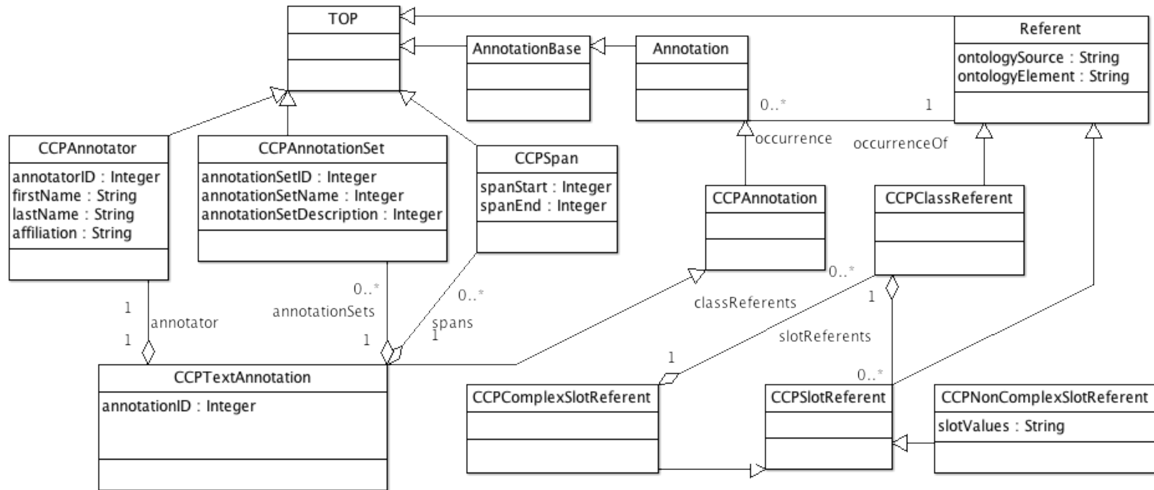

Fig. 1: UML diagram of the core CCP UIMA type system

with some modifications to support the new UIMA standard[7], in particular the inclusion of a `Referent`. The existing implementation is also not a completely generic meta-model as we have maintained some specific classes (mostly for syntactic annotations) for more efficient indexing (see Section 6 below).

It was previously noted by Heinze *et al*[11] that it can be important to separate mentions in a text from the entity those mentions refer to, for instance to link all mentions of a given entity together, and to normalize the representation of an entity. This observation led to the inclusion of the `Referent` type in the UIMA standard[7], encapsulating a reference into a data source containing the domain semantics. The specification notes that the domain ontology can be defined within the UIMA type system, as is done by [11], or externally to UIMA, as we propose here. In either case, this representation more cleanly separates semantic types from tokens, that is, the abstract representation of a concept from its physical manifestation in text (or some other data source). The `Referent` type fits in well with our approach. Each `Annotation` may be an occurrence of a `Referent`, while each `Referent` is manifested through an `Annotation`.

The primary UIMA annotation type in our type system is a `CCPTextAnnotation`, which augments the basic UIMA `Annotation` type with some additional features which we have found useful (e.g. an annotation ID and support for capturing non-contiguous spans with a single annotation). The `CCPClassReferent` type is a specialization of `Referent`, adding support for arbitrary slots, or attributes, to further characterize the reference. In our implementation, the CCPTextAnnotation `occurrenceOf` field is intended to be linked to a `CCPClassReferent` rather than a basic `Referent` object. A slot, implemented as `CCPSlotReferent`, also extends `Referent` and thus also refers to an element of an external ontology.

Figure 2 provides an example of the sort of complex representations that can be expressed in our type system. This representation results from the application of our OpenDMAP system to information extraction of biological events in

```
ClassReferent: [Elem: positive_regulation, Src: BioNLP09] "up-regulation"
   ComplexSlotReferent: [Elem: cause, Src: BioNLP09]
      ClassReferent: [Elem: protein, Src: BioNLP09] "gp41"
          SlotReferent: [Elem: ID, Src: BioNLP09] with SLOT VALUE: T3
   ComplexSlotReferent: [Elem: theme, Src: BioNLP09]
      ClassReferent: [Elem: protein, Src: BioNLP09] "IL-10"
          SlotReferent: [Elem: ID, Src: BioNLP09] with SLOT VALUE: T2
```

Fig. 2: An example of a complex class referent identified by the OpenDMAP system in text. The portion within brackets [] corresponds to the core features of a `Referent`, ontologyElement (Elem) and ontologySource (Src). We utilize an ontology specific to our submission to the BioNLP09 shared task[5], not shown.

the context of the BioNLP09 shared task[5]. The strings in quotes correspond to the text of the `CCPTextAnnotation` associated with a given class referent (`CCPClassReferent`), while the data in brackets reflects the core `Referent` features. What we see in the example is a single `positive_regulation` event with multiple slots (arguments), one labeled as a `cause` and the other as a `theme`, where the slot value for each slot is in turn a `CCPClassReferent` labeled as a `protein` and associated with a particular string in the text, and a simple slot value indicating the identifier of the protein. Essentially, the representation corresponds to the predicate `positive_regulation("gp41","IL-10")`, with the additional information that the two arguments are proteins, and their roles. All of the labels for class referent and slot elements correspond to concepts defined in the BioNLP09 ontology we developed for the task. Note that for a different use case in which the external ontology specifies the relevant protein identifiers, each protein ID, rather than with a simple slot referent, could be represented as `ClassReferent: [Elem: T2, Src: BioNLP09]`.

4 Discussion of the meta-model approach

Our experiences using a meta-model type system have enabled us to use semantic data models that would have been impossible to handle using a traditional fully specified type system. A clear advantage in using a meta-model type system is the fact that the full set of types, and the relationships between them, do not need to be generated *a priori*. This benefit is crucial in a number of different ways. First, when dealing with very large semantic ontologies, there is no need to create, and subsequently maintain, a large collection of Java class files and corresponding type system descriptors. As an example, the Gene Ontology contains 27,742 terms as of this writing. Large numbers of classes may be daunting in regards to code maintenance, but it becomes an intractable problem because ontologies may support multiple inheritance, whereas Java and the UIMA type system implementation do not. Ontologies that support multiple inheritance cannot be accurately reflected in the UIMA type system, their size not withstanding.

Equally as beneficial to representing large ontologies is the inherent domain and application independence gained with a meta-model type system. No extensions to the type system are needed when switching domains. Further, since

relationships between semantic types have been abstracted away from the type system, we are no longer restricted to representing only the hierarchical parent/child relationships as can be mirrored in the Java class hierarchy. We can take advantage of arbitrary relationships that might exist in a complex ontology.

The meta-model approach is not free of faults however. Here we discuss several drawbacks and how we combat them. In regards to developer efficiency, perhaps the most significant drawback comes from the loss of compile-time semantic type checking when using a meta-model type system. This compile-time checking is present when using a fully specified type system. It provides valuable feedback to the developer that is lost when using a meta-model type system. We have replaced this compile-time checking with two runtime validation procedures. The first checks the validity of the annotation structure in general. This is necessary because the internal UIMA collections are not genericized, i.e. it is important to check that a `ComplexSlotReferent` is not inserted into an `FSArray` storing `NonComplexSlotReferents`, or storing `CCPSpans` for that matter. We turn back to the ontologies to validate the semantic consistency of the annotations and use the formal ontology specification as a basis for type checking. Another, somewhat less crucial in our experience, drawback to using the meta-model is the inability to use the integrated annotation indexes to extract specific annotation types. We come back to this point in Section 6.

5 Enabling Analysis Engine interoperability

One of the key goals of the UIMA framework is to facilitate interoperability of analytics. The type system plays a critical role in achieving this goal, as it specifies the types available for analytics to both operate on and produce. For instance, an analytic that performs part-of-speech tagging on a text may require that `TokenAnnotation` objects marking tokens have been previously stored in the CAS. A tokenizer that produces some other kind of token annotation, say `Token` objects, cannot be combined directly with the tagging analytic.

Adopting an abstracted type system means that the annotations produced by an analytic will be grounded in an external semantic resource, which is more easily shared among research groups, and whose structure likely reflects a consensus regarding the entities and relationships defining the domain. While the development of a common type system shared among diverse research groups would also solve the interoperability problem, there has yet to emerge a widely adopted type system, despite several proposals[10, 13] and the diverse range of applications to which UIMA is applied is likely to preclude widespread adoption of one. Use of an abstracted type system gives more flexibility to UIMA system developers to add fields to their types as necessary, and at runtime, while adhering to a common semantics for domain entities. Also, because the type system is very shallow, it is more likely to be applicable to the full range of applications.

Another solution to the interoperability issue, the definition of analytics that convert from one type system to another[14], is also facilitated through the use of an abstracted type system. This is again because the domain semantics is

external to the UIMA framework: what must be converted is limited to the meta-model, i.e. whatever specific features have been added to the high-level types, while the references to the external semantics remain unchanged.

However, there does exist a limitation on interoperability using the meta-model approach. It still requires different modules referencing the same external data source to agree on the basic meta-model for that data source. For `Referent` objects which only need to refer to an external element identifier, there is little issue, but for the more complex objects we aim to capture with our `CCPClassReferent` object – including events and relations – there must be agreement on how to map the various ontology elements to the abstracted representation. For instance, conventions for naming slots and a specification of which slots are to be included in the representation must be decided for maximal compatibility. Essentially, this is a question of how the UIMA meta-model aligns to the meta-model of the underlying data source (e.g. a database schema). We feel that this level of consensus is more straightforward to achieve, and in future work we will examine automated methods for deriving the UIMA type system meta-model automatically from the meta-model of the data itself.

6 Implications for the UIMA framework

Adoption of a meta-model representation for the UIMA type system leads to certain technical desiderata for the UIMA framework, in particular with regards to indexing of objects in the CAS. In the current Apache UIMA implementation, these objects are indexed by type. For systems which implement a full semantic model in the type system, this means that it is straightforward to access only those objects that correspond to a particular semantic class, as each class corresponds to a unique type. For the meta-model approach to be as efficient, it must be possible to index and access the CAS meta-data via, at least, the two core fields of the `Referent` object. It is currently possible to define a filtered iterator over objects in the CAS that achieves the result of only iterating over objects that contain particular feature values, but this is not as efficient as direct support for the meta-model representation. While this has not so far been an issue for us, one could imagine a system requirement concerning optimization of index functionality.

The same issue impacts representation of behavioral meta-data for analytics. Input/output capabilities of analytics are specified in terms of type system types. Ideally, analytics would be able to declare capabilities in terms of feature values. This would allow analytics to the domain types specified in the `Referent` fields.

7 Conclusion

We have described an approach to constructing UIMA type systems in which the semantic complexity of the application domain is allowed to remain fully external to the type system definition itself. This has important implications for the robustness and interoperability of analytics developed in UIMA.

8 Acknowledgements

The authors acknowledge NIH grants R01LM009254, R01GM083649, and R01LM008111 to Lawrence Hunter for supporting this research. Thanks to Philip Ogren for contributing to early discussions on our type system implementation.

References

1. Apache UIMA project. http://incubator.apache.org/uima
2. Baumgartner WA Jr, Cohen KB, and Hunter L (2008): An open-source framework for large-scale, flexible evaluation of biomedical text mining systems. *J Biomed Discov Collab.* 29;3:1
3. Buyko E and Hahn U. Fully (2008): Embedded Type Systems for the Semantic Annotation Layer. In Proceedings of First International Conference on Global Interoperability for Language Resources (ICGL 2008).
4. UC Denver Ctr. Comp. Pharm., tools for BioNLP. http://bionlp.sourceforge.net
5. Cohen KB, Verspoor K, Johnson H, Roeder C, Ogren P, Baumgartner W, White E, Tipney H and Hunter L (2009): High-precision biological event extraction with a concept recognizer. Proceedings of the BioNLP09 Shared Task Workshop, p.50-58.
6. Ferrucci D, Murdock W, and Welty C (2006): Overview of Component Services for Knowledge Integration in UIMA (a.k.a. SUKI), IBM Research Report RC24074.
7. Ferrucci D, Lally A, Verspoor K, and Nyberg A (2009): Unstructured Information Management Architecture (UIMA) Version 1.0. Oasis Standard. http://docs.oasis-open.org/uima/v1.0/uima-v1.0.pdf
8. The Gene Ontology Consortium (2000): Gene ontology: tool for the unification of biology. *Nat. Genet.* May 2000;25(1):25-9.
9. Gruber, T.R. (1993): Toward principles for the design of ontologies used for knowledge sharing. In Formal Ontology in Conceptual Analysis and Knnowledge Representation, G. Poli, Editor. Kluwer academic publishers: New York.
10. Hahn U, Buyko E, Tomanek K, Piao S, Tsuruoka Y, McNaught J, and Ananiadou S (2007): An UIMA Annotation Type System for a Generic Text Mining Architecture. UIMA Workshop, GLDV Conference, April 2007.
11. Heinze T, Light M, and Schilder F (2008): Experiences with UIMA for online information extraction at Thomson Corporation. UIMA for NLP workshop at Language Resources and Evaluation Conference (LREC), May 2008.
12. Hunter, L, Lu Z, Firby J, Baumgartner WA, Johnson HL, Ogren PV, Cohen KB (2008): OpenDMAP: An open-source, ontology-driven concept analysis engine, with applications to capturing knowledge regarding protein transport, protein interactions and cell-type-specific gene expression. *BMC Bioinformatics* 2008, 9:78.
13. Kano Y, Nguyen N, Saetre R, Yoshida K, Miyao Y, Tsuruoka Y, Matsubayashi Y, Ananiadou S, Tsujii J (2008): Filling the Gaps Between Tools and Users: A Tool Comparator, Using Protein-Protein Interactions as an Example. *Pacific Symposium on Biocomputing* 13:616-627.
14. Kano Y, Baumgartner WA, McCrohon L, Ananiadou S, Cohen KB, Hunter L (2009): U-Compare: share and compare text mining tools with UIMA. *Bioinformatics*, doi:10.1093/bioinformatics/btp289
15. Ogren PV, Wetzler PG, and Bethard S (2008): ClearTK: A UIMA toolkit for statistical natural language processing. UIMA for NLP workshop at Language Resources and Evaluation Conference (LREC), May 2008.