RC24074 (W0610-047) October 10, 2006
Computer Science

IBM Research Report

Overview of Component Services for Knowledge Integration in UIMA (a.k.a. SUKI)

David Ferrucci, J. William Murdock, Christopher Welty
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Overview of Component Services for Knowledge Integration in UIMA (a.k.a. SUKI)

David Ferrucci, J. William Murdock, and Christopher Welty
IBM Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
{ferrucci,welty,murdockj}@us.ibm.com

Abstract

This white paper describes ongoing research work in the Semantic Analysis and Integration department at IBM Watson Research Center related to bridging unstructured information analysis with formal knowledge representations.

It proposes a high-level architecture for using UIMA-based analytics and various Knowledge Integration (KI) components. The development of the KI components and related methodologies is led by Chris Welty and Bill Murdock and in part funded by ARDA's NIMD program.

From Text Analysis to Knowledge

SUKI is a set of component services designed to enable transforming the results of UIMA analysis into knowledge representations (e.g., OWL) suitable for formal reasoning and processing by widely available knowledge base and semantic web tools like Protégé and Jena.

UIMA is a framework for composing and deploying text and multi-modal analytics to discover the latent meaning in these original unstructured sources. The open-source software development kit for UIMA is available on IBM's alphaWorks site and on sourceforge.net.

The results of a set of cooperating UIMA analysis components are captured in a CAS or Common Analysis Structure. The UIMA CAS manages a collection of interrelated objects that contain the results of analysis of unstructured sources. For example, *annotations*, which label regions of a document, are represented in a CAS. An annotation can, for example, classify a region as a type of entity, relationship or property. The specific nature the annotations are user-defined. Classic examples include mentions of names, organizations, times and events in textual documents. Technically the CAS can be thought of as a Java object model with some additional APIs for manipulating annotations, indices and views over documents generated by workflows of UIMA analysis engines.

UIMA applications take the results of analyzing large collections of documents, in the form of CASes, and build resources that provide precise, application-specific access to

the most relevant content discovered therein. For example, having discovered mentions of financial organizations and relationships that link them to each other, the application builds a search index incorporating these types so that documents containing this information can be precisely targeted by end-user queries.

Another important application of this technology is building more formal representations of discovered content. Rather than, for example, a search engine index that includes specific types in addition to just keywords, certain applications need to build a formal OWL knowledge base (KB) from the results captured in a stream of CASes.

Once a KB, in a standard representation like OWL, is built, off-the-shelf tools such as OWL reasoners and KB processing tools can be applied to these results to find, extend or further refine the knowledge extracted from a collection of raw documents by the UIMA analysis engines. For more discussion of the challenges to generating formal knowledge bases from text analysis, see [8, 9, 10].

UIMA Knowledge Integration Services (SUKI) is a set of components used within a UIMA pipeline that provides capabilities for generating and providing access to formal knowledge extracted from the results of a UIMA analytics.

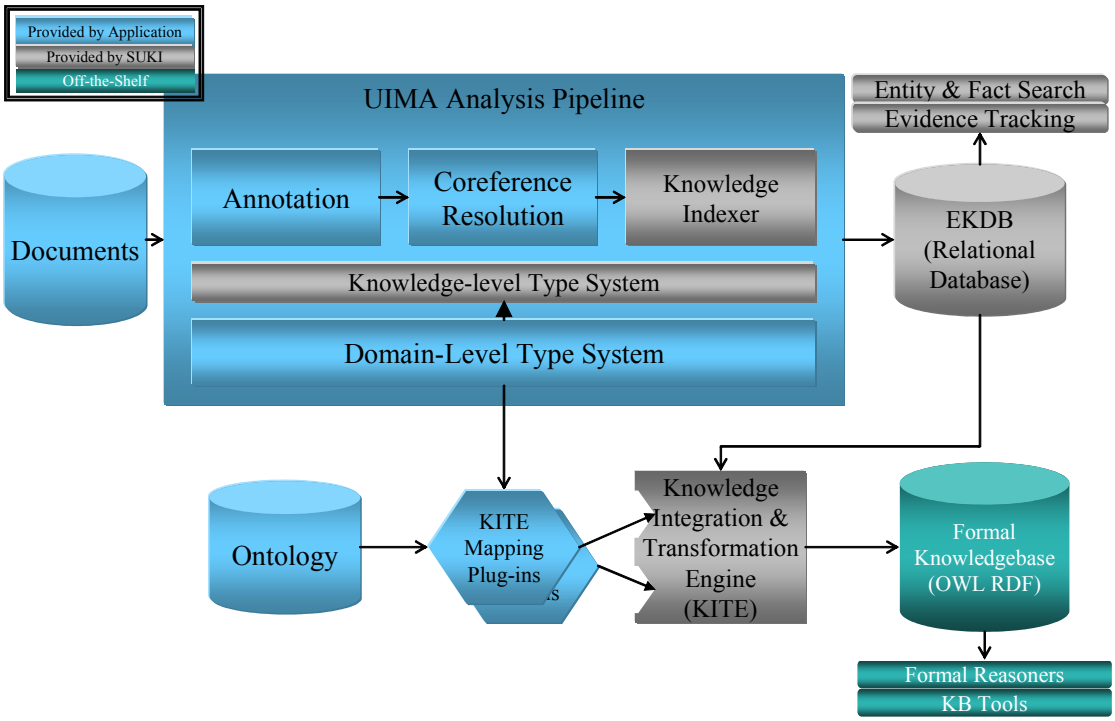

Figure 1: UIMA Knowledge Integration Component Architecture

SUKI includes four major components illustrated in the SUKI architecture in figure 1:

1) **Knowledge-Level Type System (KLT):** A minimal UIMA upper-level type system for categorizing analysis results to enable their mapping to formal knowledge representations. Developers who want to transform their analysis results to a formal OWL KB, for example, would define their type system as an extension of the KLT [6].
2) **Knowledge Indexer:** A UIMA CAS Consumer that takes the results of per-document analysis and the results of cross-document coreference resolution and indexes this information in a relational database (i.e., the EKDB – see below for details) capturing discovered annotations, entities, relationships and provenance (including links back to the engines that discovered them and to the documents in which they were discovered) [7].
3) **Extracted Knowledge Database (EKDB):** A relational database and a set of JDBC services for storing, querying and accessing the raw knowledge extraction results produced by the Knowledge Indexer [2, 7].
4) **Knowledge Integration and Transformation Engine (KITE):** A component framework for building and applying mappings between formal structured representations. In particular, KITE has been instantiated to map from a UIMA Type System (that extends the KLT) to an OWL Ontology. This KITE instance can then apply these mapping to produce the RDF for an OWL KB from extracted knowledge in the EKDB [5, 9].

Knowledge-Level Type System (KLT)

In UIMA, a Type System is a schema for the CAS. It defines the types of things that may be instantiated in a CAS by UIMA analysis engines. For example, it might include a taxonomy of legal, medical or military classes and their properties with the intent that analysis engines would detect specific mentions of these classes in text (or other, e.g., image, video) documents. The Knowledge-Level Type System (KLT) is a UIMA type system that defines the basic distinctions required by SUKI.

Relation and Entity Annotations

Some analysis engines annotate mentions of entities. For example, an analysis engine could associate a label, like *Person* to a span of text that is the name of a person. Similarly, another analysis engine might associate the label *Vehicle* to a region of an image that depicts a vehicle.

Relationship detectors are analysis engines that annotate mentions of relationships. For example the span of text “Fred Center is the CEO of Center Micros,” might be annotated by an analysis engine with the label *CeoOf* to indicate that it is a mention of that relationship.

The transformation of analysis results into formal knowledge requires the explicit distinction between annotation of entities like Persons and the annotations of relationships like being the CEO of a particular company.

The KLT extends the built-in Annotation type with the high-level UIMA types:

- *EntityAnnotation* and
- *RelationAnnotation*

so that this distinction may be explicitly captured in a standard way.

Relation and Entity Annotations

Analysis engines called coreference resolvers associate annotations with individuals, and determine if two annotations refer to the same individual. For example the mentions "Mr. Center", "Fred Center" and "CEO of Center Micros" may each have a different *EntityAnnotation*. A coreference resolver would record in the CAS that these three annotations are coreferential – they refer to a single unique individual representing Fred Center. The same can be done for relationship annotations. For example, the *CeoOf* annotation on the text "Fred Center is the CEO of Center Micros" and another on the text "Mr. Center, the top executive at Center Micros," would be associated with the single, unique relationship representing that Fred Center is the CEO of Center Micros.

The transformation of analysis results into formal knowledge requires the explicit distinction between mentions of individuals and the individuals themselves.

The KLT defines the high-level UIMA type tree:

- *Referent*
  - *Entity*
  - *Relation*
- *Link*
  - *HasOccurrence* (links referents to annotations that represent occurrences)

so that the distinction between annotations of and the unique individuals or relationships they refer to (in general "referents") may be represented in a standard way and linked to each other.

Relation Arguments

Knowledge bases store entities and relationships between entities. In order to reason about this knowledge the linkage between relationships and their arguments has to be explicitly present in the KB. For example, the individuals representing Fred Center and Center Micros must be explicitly linked to the *CeoOf* relationship to capture the fact that Fred Center is the CEO of Center Micros. In this example, we consider the individuals to be arguments to the relation.

The transformation of analysis results into formal knowledge requires the explicit linkage of relationships to their arguments or participating entities.

The KLT defines the high-level UIMA type tree:

- *Argument* (links relation referents to their argument referents)
- *RelationArgs* (links relation annotations to their argument annotations)

to connect relations and relation annotations to their arguments.

Using the KLT

SUKI requires an application or domain-level type system *extend* the KLT. Therefore, the KLT was designed with minimal commitments that make it easy to extend or adapt existing type systems. As a simple example, the *Person* type in a domain type system with *Annotation* as its super type. The *Person* type's super-type must be changed to *EntityAnntotation* to conform to the KLT. Similar modifications would have to be made for relations. See [6] for more details.

Knowledge Indexer

The Knowledge Indexer requires that:

1. a set of UIMA analysis engines have produced entity and relation annotations in a stream of CASes,
2. a set of within-document coreference resolvers have produced entity and relation referents that link to the entity and relation annotations in those CASes,
3. a set of cross-document coreference resolvers have decided which entity and relation referents in different documents are the same.

The Knowledge Indexer selects from the results of annotation and coreference resolution and populates an instance of the Extracted Knowledge Database (EKDB). It stores discovered annotations, entities, relationships and links back to the engines that discovered them and to the documents in which they were discovered.

Cross-document coreference resolution results in the merging of referents from multiple CASes into single referents within the EKDB. For example, a within-document coreference resolver might find a person referred to as "Fred Center," "Center," and "he" in one document and a person referred to as "F. W. Center" and "Mr. Center" in another document; cross-document coreference resolution could determine that those two entities are the same person.

Coreference Table: A Shared Analytic Resource

The UIMA descriptor for the Knowledge Indexer declares a dependency on a resource object that implements an interface named *CoreferenceTable*. This interface specifies methods for binding referents to each other and for asking for sets of bound referents. The Knowledge Indexer merges the referents in the EKDB that are bound together in the coreference table. Developers of cross-document coreference resolvers are expected to provide an implementation of *CoreferenceTable* and to populate it.

One existing implementation simply stores all of the referents in hash tables in memory; however, that implementation does not scale to large corpora. Another existing implementation retains a fixed-size buffer in memory and uses common caching heuristics (recency & frequency of changes) to remove less valuable referents from

consideration for binding with referents in documents that have not been processed yet. One could develop implementations of this interface that use secondary storage or network resources to provide different trade-offs among speed, memory requirements, and quality of results.

Extracted Knowledge Database (EKDB)

The Extracted Knowledge Database is a relational database that primarily stores the entities and relationships produced by the coreference resolvers. In addition, it stores all the provenance information [1] that links this knowledge back to:

- annotations associated with the entities,
- spans of text for the annotations,
- documents in the spans of text occur,
- analysis engines that produced each result.

This information is organized in relational tables, and the EKDB includes a set of JDBC services that allow the user to search for facts about entities (i.e., relationships for which an entity is an argument) and to trace these back through the engines and sources that produced them. The contents of the EKDB are applicable for generating a variety of formal knowledge representations, tracing the knowledge extraction process and providing justifying evidence for the extracted knowledge.

The EKDB stores identifiers for the documents that the knowledge was extracted from and for the components that performed the extraction. It does not store the full text of the documents and it does not store detailed descriptions of the components or the documents.

Linking to Provenance

The design of SUKI assumes that systems that want to provide detailed explanations and justifications of extraction results will rely on an external repository of provenance that describes the source documents and the analysis components in detail.

For example, if EKDB reports that a particular annotator created some annotation, a user may wish to know the names and contact information for the authors of the annotator, formal and/or informal descriptions of the capabilities of the annotator, etc. Similarly, the user may wish to see the full text of the document that was annotated, the author of that document, the date that it was written, etc. Identifiers stored in EKDB can be used to look up component and document information in the provenance repository. One provenance framework/repository that has been used with SUKI is Inference Web; document and component identifiers in EKDB can be used to relate extraction steps to Inference Web repository records to produce coherent explanations of extraction [3,4].

Knowledge Integration and Transformation Engine (KITE)

The Knowledge Integration and Transformation Engine (KITE) is a SUKI component with a plug-in architecture that supports the transformation of data from one structured system into another, and/or from one ontology into another. The KITE approach to data integration should be contrasted with the *federated* approach, in which data in heterogeneous systems remain in their original form and are wrapped to provide integration with other systems. In KITE, the data itself is transformed into a new system that has a single description (schema, ontology, type system).

KITE consists of a core framework plus the KITE Commons, a set of generic, broadly applicable plugins [5]. Of particular interest is the configuration of KITE Commons plugins that provide transformation from referents stored in an EKDB to an RDF graph that instantiates an OWL ontology; given an OWL ontology and an EKDB that has stored the knowledge extracted from a set of annotators and coreference resolvers that extend the KLT, KITE Commons plugins allow for the generation of RDF data from the EKDB.

The transformation of analysis results into formal knowledge requires the explicit mapping of types in a UIMA type system to classes and properties in a formal ontology.

The mappings in KITE are specified through a set of mapping plugins. For example, a simple type of mapping could be that the Person type in a UIMA type system maps to the Person class in an OWL ontology, or that the Company type maps to the Business class, etc. In other words, for each entity in the EKDB that is an instance of the Person type, there should be a corresponding node (or resource) in the RDF graph that is an instance of the Person class. The KITE Commons includes a plugin for which these simple direct mappings are specified in a table as plugin metadata.

A slightly more complex mapping could be that the type Agent in a type system corresponds to either the class Person or Company in an ontology. In other words, for each entity in the EKDB that is an instance of the Agent type, there should be a corresponding instance in the RDF graph that is an instance of the class that is the union of the Person class and the Company class (OWL provides the expressiveness to state this). Mapping of this sort is also supported by the plugins in the KITE Commons.

Developers of KITE-based applications are not restricted to only the plugins in the KITE Commons; they can develop new plugins that provide more complex functionality. Thus developers can create mapping systems that depend on elaborate context, require more expressivity than OWL provides, etc.

KITE also provides for provenance of the transformations, i.e., a record of what mappings took place in generating the RDF graph. For example, it is often the case that a user, when presented with an RDF triple (a binary relationship), will want to know where it came from. KITE provides for the information in the RDF graph to be connected back to the referents and annotations from which they were generated.

The KITE framework is very general and can be extended with plugins to read/write data from different sources (e.g. EKDB, RDF store, XCAS), read/write data in different representation languages (e.g. UIMA type system, OWL), as well as different ways of recording provenance for the data transformations (e.g. EKDB, Inference Web).

Improving Extracted Results using Domain-Level reasoning

Often a target ontology that is intended for downstream processing will have a more elaborate domain ontology than the type system that a typical extraction engine might use. Such a domain ontology is likely to impose additional semantic constraints, thus indicating that some sets of assertions are inherently nonsensical. When mapping knowledge from the latter to the former, it is possible to exclude knowledge that violates the more constraints of the target ontology.

For example, a domain ontology might indicate that aircraft can not be located inside people. If some extraction component indicates that there is an entity with type *Person* that is named "Ronald Regan" and some other extraction component indicates that there is an *Aircraft* that is located inside that entity, then reasoning with a domain ontology could detect a contradiction. Given this contradiction, a system could either withdraw the statement that the "Ronald Regan" entity is a *Person* (which would be the right change to make if the original text was referring to Ronald Regan National Airport) and/or withdraw the statement that the *Aircraft* is inside that entity.

KITE Commons includes a plugin that uses OWL-DL reasoning to exclude candidate mapped knowledge that violates the constraints of an OWL-DL ontology. Using simple heuristics to guide the order in which statements are considered for possible exclusion, we have shown that mapped knowledge can be substantially more precise than the original extracted information; see [9] for additional details.

Contact Information

For general information about UIMA Knowledge Integration Services please contact:

- David Ferrucci, ferrucci@us.ibm.com

For specific technical information about KLT and the EDKB please contact:

- J. William Murdock, murdockj@us.ibm.com

For specific technical information about KITE please contact:

- Christopher Welty, welty@us.ibm.com
- J. William Murdock, murdockj@us.ibm.com

References

[1] David Ferrucci. Text Analysis as Formal Inference for the Purposes of Uniform Tracing and Explanation Generation. IBM Research Technical Report RC23372. 2004.

[2] Anthony Levas, Eric Brown, J. William Murdock, and David Ferrucci. The Semantic Analysis Workbench (SAW): Towards a Framework for Knowledge Gathering and Synthesis. Proceedings of the International Conference on Intelligence Analysis. McClean, VA, May 2-6, 2005.

[3] J. William Murdock, Deborah L. McGuinness, Paulo Pinheiro da Silva, Christopher Welty and David Ferrucci. Explaining Conclusions from Diverse Knowledge Sources. Proceedings of the 5th International Semantic Web Conference (ISWC'06), Athens, GA. 2006.

[4] J. William Murdock, Paulo Pinheiro da Silva, David Ferrucci, Christopher Welty and Deborah L. McGuinness. Encoding Extraction as Inferences. In Proceedings of AAAI Spring Symposium on Metacognition on Computation, AAAI Press, Stanford University, USA, pages 92-97, 2005.

[5] J. William Murdock and Christopher Welty. Obtaining Formal Knowledge from Informal Text Analysis. IBM Research Report RC23961. 2006.

[6] J. William Murdock, Christopher Welty, David Ferrucci. UIMA Knowledge-Level Type System. IBM White Paper (to appear).

[7] J. William Murdock, Christopher Welty, David Ferrucci. Overview of UIMA Knowledge Representation and Storage Facilities. IBM White Paper (to appear).

[8] Christopher Welty and J. William Murdock. The Semantics of Multiple Annotations. IBM Research Technical Report RC22979. 2003.

[9] Christopher Welty and J. William Murdock. Towards Knowledge Acquisition from Information Extraction. Proceedings of the 5th International Semantic Web Conference (ISWC'06), Athens, GA. 2006.

[10] Christopher Welty, J. William Murdock, Paulo Pinheiro da Silva, Deborah L. McGuinness, David Ferrucci, Richard Fikes. Tracking Information Extraction from Intelligence Documents. Proceedings of the 2005 International Conference on Intelligence Analysis (IA 2005), McLean, VA, USA, 2-6 May, 2005.

