A Survey of Text Mining Architectures and the UIMA Standard

Mathias Bank[†], **Martin Schierle**[‡]

[†]Pattern Science AG, 63579 Freigericht, Germany
[‡]Mercedes-Benz R&D North America, Palo Alto, USA
m.bank@cid.biz, martin.schierle@daimler.com

Abstract

With the rising amount of digitally available text, the need for efficient processing algorithms is growing fast. Although a lot of libraries are commonly available, their modularity and interchangeability is very limited, therefore forcing a lot of reimplementations and modifications not only in research areas but also in real world application scenarios. In recent years, different NLP frameworks have been proposed to provide an efficient, robust and convenient architecture for information processing tasks. This paper will present an overview over the most common approaches with their advantages and shortcomings, and will discuss them with respect to the first standardized architecture – the Unstructured Information Management Architecture (UIMA).

Keywords: Language Engineering, NLP Architecture, UIMA

1. Introduction

With the rapid progress in technology and world wide web usage the amount of textual data is growing steadily, thereby carrying and concentrating more and more knowledge. Luckily the development of Natural Language Processing (NLP) methods is also progressing and offers today a manifold functionality to annotate text, extract information and build corpus based resources as for example used for Information Retrieval. Toolboxes and libraries like the ASV Toolbox[1], the JULIE NLP Toolsuite[2] or OpenNLP[3] offer impressive methods and workflows for a convenient usage. But the interchangeability of those modules and their arbitrary composition to new workflows is not always given. Therefore the development and application of applications is a time consuming task with lots of reimplementations and modifications not only for the scientific community but also for companies dealing with their urge for information.

Today there are several linguistic processing architectures freely available to solve this problem. We will present, sketch and compare the architectures having had the strongest impact on research in recent years, namely TIPSTER (Grishman, 1996, chapter 1.0), Ellogon (Petasis et al., 2002), GATE (Cunningham, 2000) (including newest modifications (Cunningham et al., 2010)) and Heart of Gold (Schäfer, 2006). With respect to scientific issues we will focus on architectures which are well documented and freely available. Furthermore we will focus on infrastructural capabilities rather than included libraries and GUI tools (e.g. U-Compare[4]). These can always be supplemented and some of the architectures and frameworks are evolving fast, so that every survey would be outdated soon. The underlying infrastructural ideas and theories however represent the current scientific state of the art in information processing and show as well limitations as opportunities.

The discussed architectures are finally compared to UIMA (Lally et al., 2009), the first official OASIS standard for Unstructured Information Management Architectures, which is realized in the Apache UIMA architecture. We will discuss specification shortcomings and provide possible alternatives to improve applicability and interoperability among different software developers in the linguistic scenario. It will be shown that UIMA can be seen as the most evolved and comprehensive architecture available to the date of this work – at least regarding the infrastructural capabilities and neglecting the pure toolbox size. We encourage the scientific community to extend the OASIS standard with linguistic standards to support an improved interchange among researchers which would increase code reusability and quality.

2. Language Engineering Architectures

In this section, we will present and sketch different architectures for information retrieval. Each one is analyzed to which extent it can serve as a framework for information retrieval and how it can be used to create individual analysis systems. Six different categories are analyzed specifically, based on thoughts of (Cunningham, 2000):

1. *Modularity and interchangeability:* Modularity guarantees high quality and short development cycles. A framework for information retrieval should encourage to develop modular and independent modules dealing with one special task only. It's to the architecture to provide communication interfaces to other resources (processing resources and data resources).

2. *Workflow management:* Each specialized processing resource must be arranged in a superordinated workflow. Depending on the analysis complexity, different

[1]`http://wortschatz.uni-leipzig.de/~cbiemann/software/toolbox/index.htm`
[2]`http://www.julielab.de/Resources/Software/NLP_Tools.html`
[3]`http://opennlp.sourceforge.net`
[4]`http://u-compare.org/`

types could be necessary: serial, parallel, conditional, iterative, nested or even cascaded[5] workflows. It is analyzed, which types are supported and if the architecture supports *analysis aware* systems in which it is necessary to change workflows, settings or resources depending on prior analysis results, languages or domains.

Only native architecture support for standardized workflows is considered. We neglect programmatical workflow control, although all presented architectures allow to create individual and thus complex processing units on code level.

3. *Configuration management:* Processing resources should be configured by configuration parameters to guarantee reusable code. It is analyzed in which way the architectures enforce the usage of outsourced metadata and which possibilities are available to modify these parameters depending on the fulfilled workflow. In this context it is also analyzed, if configuration parameters can be defined analysis aware which means that the parameters can be defined in relation to already extracted data.

4. *Resource handling:* Many processing resources require data resources (e.g. dictionaries) and corresponding access structures in addition to parameter settings. Shifting the resource management to the architecture would enable complete domain independent and thus reusable processing units. A formal specification and restricted interfaces provided by the framework improve parallelization, distribution and even analysis aware behavior.

5. *Parallelization and distribution:* With the increasing amount of unstructured data, it becomes more and more important to provide an architecture that enables parallelization and distribution. It is analyzed, if the architectures provide methods for parallelization and distribution and in which way they support processing units in using these possibilities.

6. *Annotation model:* A typical Information Retrieval toolchain enriches unstructured data with relevant informations like part of speech tags. It is analyzed in which way the architectures model and store the information, and if efficient and convenient access structures exist. Typical questions are if annotations are stored inline or as stand-off markup, if they are typed to provide formal verification and declaration and if annotation types can be inherited. Furthermore the architecture should allow annotations as fields of other annotations (e.g. for parse trees) as well as indexing mechanisms, iterators and subiterators for given types.

2.1. TIPSTER

TIPSTER describes a common architecture developed to provide means for efficient information retrieval and information extraction to government agencies and general research ((Grishman, 1996), chapter 1.0). TIPSTER was not only designed for multilingual applications in a wide range of software and hardware environments, but also introduced the thought of interchangeable modules from different sources. While being defined in an abstract (yet object-oriented) way, the TIPSTER architecture is implemented in a number of programming languages like C, Tcl and Common Lisp.

The TIPSTER architecture can be seen as document centric. Each document may be contained in one or more collections and is the atomic unit of information retrieval which is considered as the repository of any extracted data. It is possible to derive documents from other documents, thereby forming logical documents.

Any information about the text is given by stand-off annotations. Each annotation can be defined by the system engineer using arbitrary annotation names with arbitrary attributes. Each annotation name has to be defined in a type declaration, which is merely used for documentation, but intended to serve as a base for formal verifications. It is possible to assign each annotation to one or more spans of text in the document. Attributes allow primitive data types as values as well as references to other annotations or documents, thereby allowing even hierarchical structures such as parse trees. Some annotation types and general annotation attributes are predefined according to the *Corpus Encoding Standard* (*CES*, (Ide, 1998)) to facilitate the interchangeability of modules and the usage of the architecture. Annotations are managed and indexed to ensure efficient access for different use case scenarios.

TIPSTER's strength is the sophisticated typed annotation model – adopted by as well UIMA (Ferrucci and Lally, 2004), GATE (Cunningham et al., 2002) and Ellogon (Petasis et al., 2002) – and the integration of existing standards like CES. The main shortcoming is the sparse specification of processing resources. Besides being able to work with the Tipster document model no further characteristics are defined. There is no parameter or resource management by the architecture, and no sophisticated workflow management. This makes interchangeability, a standardized parallelization and distribution of processing and language resources impossible.

2.2. Ellogon

The Ellogon platform (Petasis et al., 2002) is intended to be a multilingual and general purpose language engineering environment aiming to help as well researchers as companies to produce NLP systems. In comparison to other language engineering architectures at the time of creation, Ellogon was designed to support a wide range of languages by using Unicode to work under the major operating systems and to be more efficient with respect to hardware requirements (cp. (Petasis et al., 2002)). The architecture is build around three basic subsystems (cp. (Petasis et al., 2002)):

1. The *Collection and Document Manager* (*CDM*), which is implemented according to the TIPSTER architecture, is a well-defined programming interface to all processing resources. The central element is a collection of documents, each consisting of textual data

[5] corpus and document based processing iteratively applied

and linguistic information stored as stand-off annotations and attributes. Programmatically, the collection can be modified online. Annotations as well as attributes are typed using user-defined textual identifiers.

2. Ellogon uses pre- and postconditions of processing resources (components) to establish relations among each other and to automatically generate workflows. These workflows can be aggregated to *systems* by the user. On this level, the Ellogon architecture can change from document level processing to corpus based processing so that next to basic information extraction algorithms machine learning algorithms working on the complete data set are possible.

3. A modular and pluggable component system to load and integrate processing resources (called *modules* in Ellogon) at runtime. Modules comprise the implementation part doing the analysis and the interface part declaring metadata for the framework. The interface describes pre- and postconditions, parameters and GUI elements to provide user access to the component. Conditions are specified using annotation types and attributes of the documents or the collection, parameters are restrained to a small set of predefined types.

Although Ellogon provides an integrated workflow management system based on pre- and postconditions, only serial and cascaded workflows are possible. Parallel, conditional, nested or iterative workflows are not supported by the architecture. The same applies to the distribution of processing and language resources. Ellogon offers however the possibility to use components as web services. There is no resource management and parameterization is very basic and not analysis aware. It is neither possible to inherit already available type definitions for annotations nor is it possible to reuse metadata in derived processing resources (metadata inheritence).

2.3. GATE

The *General Architecture for Text Engineering* (*GATE*, (Cunningham, 2000)) was developed to provide an infrastructure for a wide range of language engineering activities that also considers the prior infrastructural findings of the scientific community. It was originally released 1996 and is today available in version 5. The current version is implemented completely in Java, uses Unicode as default encoding and is also capable of processing audio-visual content. Besides comfortable GUI editing tools, it comprises two central elements (cp. (Cunningham, 2000), chapter 7):

1. The *GATE Document Manager* (*GDM*) is implemented according to the TIPSTER specifications about document management. Therefore the core of the manager is given by a collection of documents containing text and annotations, which – similar to Ellogon – can be modified online. With the GDM being the common interface to all processing resources it is also the central data repository. All processing resources obtain the annotated documents from the GDM and return them for later processing steps. Annotations on documents are organized in so-called annotation graphs (Bird et al., 2000). Except the information about start and end node, every annotation defines an identifier, a type declaration and additional attributes. Annotation schemes similar to TIPSTER define common annotations with their attributes (cp. (Bontcheva et al., 2004)). Although one annotation is determined to refer to only one span of text, the architecture offers the possibility to create multiple annotation graphs per document.

2. A Collection of REusable Objects for Language Engineering (CREOLE) which can be seen as a library of processing resources, language resources and data structures for general usage (cp. (Cunningham, 2000), chapter 7.2). Users can extend the CREOLE objects by own implementations using the CREOLE API and initialize and run them on documents using the GGI or programmatic access. All necessary information for the processing resource (PR) is provided by the GDM in the form of documents with text and annotations and results are written back respectively.

   Every CREOLE component must specify its configuration to facilitate workflow creation, accessibility by the Gate Graphical Interface (GGI) and interchangeability. This metadata comprises parameters as well as pre- and postconditions (in the form of annotations and attributes). It is expressed in XML or by using *Java Annotations* (Cunningham et al., 2010), which simplifies inheritance of processing resources significantly.

Besides the infrastructural capabilities GATE offers an exhaustive library of GUI tools, data access structures, language resources and import filters for common document formats. The workflow management offers possibilities for conditional processing and collection level processing. Although language resources may be distributed and applications may run on different machines, there is still no sophisticated workflow management allowing iterative, nested or parallel processing (cp. (Bontcheva et al., 2004), (Cunningham et al., 2010)).
An impressive feature is the possibility for finite state processing over annotations based on regular expressions. This *Java Annotation Pattern Engine* (*JAPE*) operates on given pattern/action rules which define a pattern of annotations and their features in the input document, and a corresponding action to perform if the pattern is matched. Corresponding actions may also include the creation of new annotations or the modification of the matched ones.
GATE can be seen as quite exhaustive. Resources are separated and described using metadata that can be composed in workflows. Inheritance of modules is facilitated using Java Annotations, collection level processing is possible and the document model with typed annotations is as well comprehensive as well defined. Shortcomings of GATE are the lack of a sophisticated workflow management (especially with respect to parallelization) and that formal declarations

of resources are not analysis aware – neither pre- or post-conditions nor parameters can be defined with respect to annotations and attributes. Although many different kinds of resources can be accessed via predefined structures, there is no formal specification for individual resource management. Type inheritance is not possible.

2.4. Heart of Gold

Heart of Gold (Schäfer, 2006) has been developed within several research projects[6] funded by the EU and the German ministry for education and research BMBF and is described as a lightweight and XML-based middleware architecture for shallow and deep processing workflows of NLP components (Schäfer, 2008).
The main architectural design principle behind Heart of Gold is the use of open XML standoff markup to represent the input and output of all components as it is easily exchangeable and transformable using for example XSLT[7]. Unicode handling is directly given by the XML standard. The core of the architecture is the *Module Communication Manager* which serves as an interface to the application by getting requests and returning results (cp. (Callmeier et al., 2004)). Internally the manager organizes the workflow of processing resources, the persistence layer and the data exchange between components. Processing resources can be implemented in Java or may be called using XML-RPC – even on remote machines. Workflows are specified using the *System Description Language SDL* (Krieger, 2003) which covers sequential, parallel and iterative execution. By defining so-called sub-architectures consisting of other modules, SDL also allows nested and cascaded workflows. Analysis results are represented as stand-off annotations in an RMRS-XML format (Copestake et al., 2006). Every input document can be enriched by a collection of annotations, which may also refer to other annotations and collections by the use of unique identifiers. If modules create output in different formats (or two cooperating modules use different annotation schemes), the Heart of Gold architecture supports XSLT transformation to support module communication. XSLT is also utilized to combine and query annotations.
Heart of Gold offers no capabilities for the definition of pre- and postconditions and there is no parameter or resource management. Furthermore conditional workflows are not supported by the architecture. The usage of standard XML formats and transformation techniques makes the architecture however very flexible, although requiring expensive transformation algorithms.

2.5. Other NLP software

A widespread and common toolbox is OpenNLP[8], which considers itself to be "an umbrella for various open source NLP projects to work with greater awareness and (potentially) greater interoperability". With respect to this work OpenNLP is of minor importance, as it does not define any infrastructural base, but is just a collection of perhaps even completely different NLP tools.

[6] `http://heartofgold.dfki.de/index.html`
[7] `http://www.w3.org/TR/xslt`
[8] `http://opennlp.sourceforge.net/`

The *Advanced Language Engineering Platform* (*ALEP*) is neglected here, because its restrictions with respect to operating systems (only Unix) methods and resources prevented it from being used in a large scale.
Another toolbox widely used is LingPipe[9], which sees itself as a "suite of Java libraries for the linguistic analysis of human language". The Java based software includes a wide range of machine learning algorithms for classification and clustering like k-means, SVM or Naïve Bayes, but unfortunately does not provide a very sophisticated infrastructure and is only available under a very restrictive license.
Other toolboxes and libraries which are freely available for research purpose but does not provide sophisticated infrastructure capabilities beyond simple pipelines are FreeLing (Atserias et al., 2006), MALLET (McCallum, 2002) and NLTK (Loper and Bird, 2002). Another toolkit which provides more complex workflows and stand-off annotations is LinguaStream (Bilhaut and Widlöcher, 2006).

2.6. Overview

An overview over a set of features selected with respect to application development is presented in table 1, including features of the UIMA framework presented and discussed in the next section.

3. UIMA - an OASIS Standard

3.1. Architecture properties

The *Unstructured Information Management Architecture* (*UIMA*) was originally developed and published by IBM to facility the analysis of unstructured information like natural language text, speech, images or videos (Lally et al., 2009). The Java based framework was accepted as Apache Incubator project in 2006 and has been standardized by OASIS in 2009[10]. 2010 UIMA graduates from the Incubator. Although the framework explicitly targets different kinds of data, its focus lies on the analysis of text. Nevertheless, UIMA uses the term *artifact* to denote the subject of analysis in contrast to document as used in other architectures.
UIMA specifies six central elements, thereby defining the architecture and its usage as well (cp. (Lally et al., 2009) chapter 3 ff.):

1. The *Common Analysis Structure* (*CAS*) is the commonly shared data structure to represent the artifact as well as according metadata. The artifact is encapsulated in one or more *Subjects of Analysis* (*Sofas*). Similar to the GATE document model the CAS can be considered to be the common interface for sharing data between all analytics with all contained objects being modeled using the UIMA *Type System* (see below). Stand-off annotations are allowed to reference other annotations or objects in general, thereby allowing hierarchical structures such as parse trees. According to the specification, annotations may be enriched with additional metadata about the annotation. All annotations are indexed and it is possible to access each of them efficiently via iterators and subiterators.

[9] `http://alias-i.com/lingpipe/`
[10] `http://incubator.apache.org/uima/news.html`

Table 1: Comparison of NLP architectures: "+" fully supported, "0" partially supported, "-" not supported.

	Tipster	Gate	Ellogon	HoG	Uima
Stand-off annotations	+	+	+	+	+
Typed annotations	0	+	+	+	+
Annotation Type inheritance	-	-	-	-	+
Processing Resource inheritance	-	+	-	-	0
Processing Resource interchangeability	0	+	+	+	+
Language Resource interchangeability	-	0	-	-	-
Access Structure interchangeability	-	0	-	-	-
Parameter Management	-	+	+	0	+
Analysis Awareness	-	-	-	-	0
Resource Management	-	-	-	-	0
Workflow Management	-	0	0	0	+
Parallelizable	-	-	-	-	+
Distributable	-	-	-	-	+
Tool-Box	0	+	+	-	+

Import and export of CAS objects is achieved by using the XML Metadata Interchange specification[11] (*XMI*). XMI was chosen because of being a widespread standard and being aligned with object-oriented programming and UML.

2. Every CAS must conform to a user-defined type system, which is described within the *Type System Model*. The modeling language used to define this model is the *Ecore* standard of the *Eclipse Modeling Framework* (*EMF*). Although UIMA does not define specific type systems for analytics, it does define a *Base Type System* containing some commonly-used and domain independent types, thereby allowing a fundamental level of interoperability between different applications. These include primitive types as defined by Ecore and general source document information like an URI pointing to the source document.

3. *Abstract Interfaces* are provided to define the standard component types and operations. The specification defines two fundamental types of *Processing Elements* (*PE*): *Analytics* and *Flow Controllers*. Analytics process the CAS and update its content with new or modified annotations. Each one is able to process data additionally at batch and collection level so that UIMA is able to switch from document based to collection based processing. *CAS multipliers* as special form of analytics are able to map a set of input CASes to a set of output CASes by creating new ones or merging existent ones. Flow Controllers determine the route CASes take through a workflow of multiple analytics. Describing the desired flow in a flow language like BPEL[12] results in a powerful, flexible, distributable and reusable workflow management.

4. Every analytic describes its processing characteristics using *Behavioral Metadata*. This metadata declaratively describes in terms of type system definitions prerequisites to the CAS, what elements in the CAS are analyzed and in which way the CAS contents are altered or modified. Using this information, UIMA can automatically discover required analytics and their composition can be supported by an automated process. Additionally the metadata helps in facilitating efficient sharing of CAS content among processing elements. Behavioral Metadata specifies required inputs and the types of objects which may be created, modified or deleted. Although implementations according to the UIMA specification are free to use any expression language to represent these conditions, the specification defines a mapping to the *Object Constraint Language*[13] (*OCL*).

5. Every Processing Element has to publish *Processing Element Metadata* to support component discovery and the composition of processing elements. This data includes parameterization, the priorly defined behavioral metadata and identification information like version, vendor and description. The processing element metadata is provided as a valid XMI document. It has to be defined for each processing resource separately, but it is possible to aggregate a group of processing resources and to override individual settings (e.g. a common encoding). This is also possible for complete workflows. Therefore it is possible to define general settings for each processing unit and to create specialized settings on application level.

6. UIMA facilitates the publication of analytics as web services by specifying a WSDL[14] description of the abstract interfaces. Additionally a binding is defined to a concrete SOAP interface, which must be implemented by compliant architectures.

UIMA can be seen as the most evolved and comprehensive architecture available to the date of this work – at least regarding the infrastructural capabilities and neglecting the

[11] `http://www.omg.org/spec/XMI/2.1.1/`

[12] `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`

[13] `http://www.omg.org/spec/OCL/2.0/`

[14] `http://www.w3.org/TR/wsdl20/`

pure toolbox size. UIMA's strengths lie particularly in its standardization and the consequent integration of existing standards like XMI, Ecore, XML, OCL or BPEL. Most workflow types pose no challenge to the architecture, nor does parallelization or distribution of processing resources. All metadata is declared formally, including pre- and post-conditions, parameters and language resources. UIMA is the only architecture providing at least partially resource management and analysis aware parameter handling.

3.2. Shortcomings of the UIMA specification

The UIMA specification is the first architecture for unstructured information management that is standardized. It is realized by the scientifically established UIMA architecture. The specification relies on well defined other standards, therefore providing the basis for quick adoption and wide dissemination. But besides all the advantages, there are some shortcomings from an application point of view:

Analysis Awareness: An impressive feature of the UIMA *architecture* is the capability for defining named parameter groups. The annotator can decide which group to use depending on arbitrary conditions. For language-specific annotators for example it is possible to define language dependent parameters that are selected based on the document language. A language fallback can be defined as a default group, which allows the usage of e.g. the *en* settings if there are not settings for *en-GB*. This feature is not only advantageous, but absolutely required. In application scenarios it might be necessary to process documents in many domains and languages. Defining, requesting and handling separate parameters (and resources) for all these cases is time-consuming and error-prone if done in the processing resource.
In contrast to the current implementation, the UIMA *specification* is missing this very important feature. We furthermore suggest to specify the complete processing element metadata as analysis-aware using the same OCL notation as the behavioral metadata. Every parameter and data resource needs to be defined with respect to conditions on arbitrary annotation types and attributes and the architecture needs to handle the management instead of the annotator (which right now has to ask for the right parameter group on its own).

Meta-Data inheritance: Inheritance in software engineering aims in increasing code-reusability and semantic subtyping. While the architecture itself is completely object-oriented and therefore supports inheritence, there is one weakness: There is no applicable model for metadata inheritance. Although a user is free to subclass any annotator, the metadata has to be rewritten – thereby introducing error sources and complicating maintenance. A possible solution to this problem is given by GATE: Metadata can be expressed using Java Annotations, which can be inherited with the code.

Resource Handling: Although the UIMA specification is very comprehensive with respect to processing resources, document and annotation model there is a lack of definition of data resources and access structures. The OASIS standard does neither include already defined language standards like for example given by ISO TC37SC4[15], EAGLES[16] or LMF[17] nor does it provide an according library of resources and access structures, as for example provided by GATE. Every application has to care about standards on its own.
Although we are aware of the fact that UIMA only describes the processing architecture and the users should rely to resource standards on their own, this lack is an important definition shortcoming. Without defining resource standards and common interfaces for resource handling, the framework is not able to care about optimal parallelization and distribution. Additionally, a real exchange of processing resources in the scientific community is prevented because of different data structures.
We suggest to extend the basic approach for resource handling already realized in the architecture implementation and to include already defined standards to the specification. A common access library and interfaces – similar to GATE – can additionally increase code reusability, maintenance and global system performance because the framework could efficiently care about resource distribution. Thus even new standards and approaches for improved resource representations (e.g. LexInfo[18] (Buitelaar et al., 2009)) can be implemented in a reusable way.

NLP Type System: Similar to the lack of standardization of data resources and access structures, the definition of commonly usable type systems is by far not comprehensive enough. Beside the definition of a very general *Source-DocumentInformation* type, no other types are defined in the standard. Interchange of NLP modules within scientific community and industry will only succeed if there are at least some well defined type system standards. Even simple information like the part-of-speech of a word can be modeled in several ways (independent type, attribute of type token etc.) thereby hindering the interchangeability of for example syntax parsers. Although it is clear that no universal type system standard can be derived, it is however possible to define such standards with respect to specific domains or applications. At least the inclusion of already existing standards like the corpus encoding standard CES into the UIMA specification would be a significant improvement. Positive examples for this are given by Tipster and GATE. The currently available OASIS specification enforces the usage of wrappers or – similar to Heart of Gold – the inclusion of transformation analytics, which decrease system performance significantly.

Workflow Management: Workflow management is supported by the UIMA specification (and the architecture respectively) through the usage of *FlowControllers*. The well specified description language BPEL may be used to enable iterative, conditional, parallel and even distributed workflows, so that most document based analytical projects can be realized by the use of descriptor files. Collection based analyses are supported in a limited way as UIMA provides special method interfaces that are called when all docu-

[15]`http://www.tc37sc4.org`

[16]`http://www.ilc.cnr.it/EAGLES96`

[17]`http://www.lexicalmarkupframework.org/`

[18]`http://lexinfo.net/`

ments are processed.
Unfortunately, aggregated workflows are only defined as collections of different analytics. The OASIS specification does not allow nested or cascaded workflows on descriptor level. So it is neither possible to use document and collection based analytics iteratively nor is it possible for a single processing resource to call other workflows to accomplish its tasks. Aggregation tasks, which are used in machine learning algorithms for example, are thus only possible if a processor programmatically calls a subworkflow, which is possible in UIMA. This possibility is however not standardized and in consequence the approach does not support code interchangeability.
In the eyes of the authors, aggregation tasks are as important as document-centered text mining and information extraction tasks. UIMA can only become accepted if all types of analyses are supported. We propose to also support *FlowController* definitions next to analytic definitions. Thus arbitrarily nested and cascaded workflows would be possible. Depending on the implementation, these subworkflows could be defined in the same or in a separate workflow description file. A simple realization would be to provide a standardized processor that calls subworkflows at its own. This extension would create a completely configurable analysis system.

Annotation model: UIMA supports a very flexible typed annotation model with a small definition lack: Most processing resources have the ability to create different outputs with associated probabilities. Considering part of speech tagger or spelling correctors it is common that several tags or corrections are created, and usually only the one with highest probability is added to the document as annotation. The alternative outputs are however of big interest and may be used and even resolved by later modules. A syntax parser for example might switch back to a part of speech tag with inferior probability if the parse of the sentence is not possible otherwise. Although UIMA supports the addition of alternative suggestions as additional annotations or features, other modules cannot use this information reliably if it is not standardized and managed by the architecture itself. A scientific exchange of modules will not be possible, if every module encodes alternative solutions and their certainties in different ways - or even skips them completely. A more satisfactory option is to store all possible annotations according to their probability. A following analysis resource is able to use this information to judge on its own, which annotation is probably the best with respect to its task. The definition of such annotation groups containing alternative annotations marked with distinct probabilities allows more powerful workflows by changing and reweighting annotations in later steps. In this way, the architecture could also provide fast access structures and efficient annotation management. Modules which do not need alternative annotations would just use (and maybe only see) the representative of each group – the one with the currently highest certainty.

4. Conclusion

This paper presented and discussed some widely spread NLP architectures and compared their features with the UIMA specification from an application developer's point of view. Concluding this work it can be said, that the specification is very comprehensive, well defined on commonly accepted standards and finally offering an impressive representational power not only for NLP applications. Especially the component based approach that can be combined via BPEL description language makes UIMA a great framework for interchangeability and research issues with the possibility to create real analysis applications fast. It however must be stated that there is still space for improvements, especially with respect to resource management and workflow architecture. A real interchangeability of methods will only be given if all components and all interfaces are specified. As this is nearly infeasible to achieve, a set of corresponding standards for common NLP applications and domains would be of great advantage. Beside these shortcomings there are some smaller facets of the specification which could be improved, like the analysis awareness of descriptor metadata.
Finally it is to state that the UIMA specification and the Apache implementation is a promising effort, which should be accepted and used in the community. It provides an impressive and fast architecture not only for textual information retrieval but for every kind of unstructured data. Although there are some specification issues left, the current OASIS standard could be very helpful to the scientific community for unified module representations.

5. References

Jordi Atserias, Bernardino Casas, Elisabet Comelles, Meritxell González, Lluis Padró, and Muntsa Padró. 2006. Freeling 1.3: Syntactic and semantic services in an opensource nlp library. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC'06)*, pages 48–55.

Frédérik Bilhaut and Antoine Widlöcher. 2006. Linguastream: an integrated environment for computational linguistics experimentation. In *EACL '06: Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics: Posters & Demonstrations*, pages 95–98, Morristown, NJ, USA. Association for Computational Linguistics.

Steven Bird, David Day, John S. Garofolo, John Henderson, Christophe Laprun, and Mark Liberman. 2000. Atlas: A flexible and extensible architecture for linguistic annotation. *CoRR*, cs. CL / 0007022.

K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham. 2004. Evolving GATE to Meet New Challenges in Language Engineering. *Natural Language Engineering*, 10(3/4):349—373.

Paul Buitelaar, Philipp Cimiano, Peter Haase, and Michael Sintek. 2009. Towards linguistically grounded ontologies. In *The Semantic Web: Research and Applications*, volume 5554 of *Lecture Notes in Computer Science*, pages 111–125. Springer Berlin / Heidelberg.

Ulrich Callmeier, Andreas Eisele, Ulrich Schfer, and Melanie Siegel. 2004. The deepthought core architecture framework. In *Proceedings of LREC 04*, Lisbon, Portugal.

Ann Copestake, Peter Corbett, Peter Murray-rust, Advaith Siddharthan, Simone Teufel, and Ben Waldron. 2006. An architecture for language processing for scientific texts. In *In Proceedings of the 4th UK E-Science All Hands Meeting*.

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 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*.

Hamish Cunningham, Diana Maynard, and Kalina et al. Bontcheva, 2010. *Developing Language Processing Components with GATE Version 5 (a User Guide)*. The University of Sheffield, 01.

Hamish Cunningham. 2000. *Software Architecture for Language Engineering*. Ph.D. thesis, University of Sheffield. `http://gate.ac.uk/sale/thesis/`.

D. Ferrucci and A. Lally. 2004. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering*.

Ralph Grishman. 1996. Tipster text phase ii architecture design. In *Proceedings of a workshop on held at Vienna, Virginia*, pages 249–305, Morristown, NJ, USA. Association for Computational Linguistics.

Nancy Ide. 1998. Corpus encoding standard: Sgml guidelines for encoding linguistic corpora. In *In Proceedings of the First International Language Resources and Evaluation Conference*, pages 463–70.

Hans-Ulrich Krieger. 2003. Sdl - a description language for building nlp systems. In *In Proceedings of the HLT-NAACL Workshop on the Software Engineering and Architecture of Language Technology Systems, SEALTS*, pages 84–91.

Adam Lally, Karin Verspoor, and Eric Nyberg. 2009. Unstructured information management architecture (uima) version 1.0, March.

Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*.

Andrew Kachites McCallum. 2002. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu.

Georgios Petasis, Vangelis Karkaletsis, Georgios Paliouras, Ion Androutsopoulos, and Constantine D. Spyropoulos. 2002. Ellogon: A new text engineering platform. In *In Proceedings of the Third International Conference on Language Resources and Evaluation (LREC 2002), Las Palmas, Canary Islands*, pages 72–78.

Ulrich Schäfer. 2006. Middleware for creating and combining multi-dimensional nlp markup. In *NLPXML '06: Proceedings of the 5th Workshop on NLP and XML*, pages 81–84, Morristown, NJ, USA. Association for Computational Linguistics.

Ulrich Schäfer. 2008. Shallow, deep and hybrid processing with uima and heart of gold. In *Proceedings of the LREC-2008 Workshop Towards Enhanced Interoperability for Large HLT Systems: UIMA for NLP, 6th International Conference on Language Resources and Evaluation. LREC-2008, May 26 - June 1, Marrakesh, Morocco*, pages 43–50. ELRA.