A Development Environment for Configurable Meta-Annotators in a Pipelined NLP Architecture

Youssef Drissi, Branimir Boguraev, David Ferrucci, Paul T. Keyser, and Anthony Levas
IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA
E-mail: {youssefd, bran, ferrucci, pkeyser, levas}@us.ibm.com

Abstract

Information extraction from large data repositories is critical to Information Management solutions. In addition to prerequisite corpus analysis, to determine domain-specific characteristics of text resources, developing, refining and evaluating analytics entails a complex and lengthy process, typically requiring more than just domain expertise. Modern architectures for text processing, while facilitating reuse and (re-)composition of analytical pipelines, place additional constraints upon the analytics development, as domain experts need not only configure individual annotator components, but situate these within a fully functional annotator pipeline. We present the design, and current status, of a tool for configuring model-driven annotators, which abstracts away from annotator implementation details, pipeline composition constraints, and data management. Instead, the tool embodies support for all stages of ontology-centric model development cycle – from corpus analysis and concept definition, to model development and testing, to large scale evaluation, to easy and rapid composition of text applications deploying these concept models. With our design, we aim to meet the needs of domain experts, who are not necessarily expert NLP practitioners.

1. Introduction

Information Management is a fully integrated discipline that can give users access to the broad range of information they need. This information comes not only from traditional structured sources, but extends to a broad range of unstructured information (email, web content, etc.) which comprises over 80% (Moore, 2002) of all information available. Information analysis and extraction of this unstructured information from large multi-media data repositories (text, speech, video, etc.) are critical to Information Management solutions that expose this information to users.

The process, however, of developing, refining, and evaluating the analytics, required for information extraction, is complex and lengthy, requiring knowledge of algorithmic techniques, programming in a variety of languages and processing content using underlying runtime frameworks. This process is iterative and cyclical in nature – analytics are incrementally enhanced and evaluated against previous results.

This work addresses a class of problems arising from the needs of domain experts – who are not necessarily expert NLP practitioners – to develop text analysis applications rapidly and adapt these to different types of language resources and domains. Our approach focuses on

1) understanding the cyclic/iterative nature of the overall process that a practitioner engages in,
2) developing a software environment that provides tooling to assist in each step of the process, and
3) providing a methodology for the user to follow for each process step/tool choice they make.

Our goal is to make available a rich set of tools that users can apply depending upon their level of expertise as well as the technical challenges that the analytic development brings. Some tools may be very easy to use, require less expertise and work effectively for a limited range of problems, whereas other tools may require deeper knowledge and address problems that require more complex analytical processing and greater expertise.

The Unified Domain Adaptation Tool (UDAT) is a software environment we have built to address these challenges. In this paper we will focus primarily upon a few distinguishing features and how these contribute to the extensibility and ease-of-use of UDAT. These features are:

- The Ontology-centric approach where concepts are first class citizens that aggregate *models* (*e.g.* rules, pattern files, etc) used for finding them,
- The Configurable Annotator Tool (CAT) Framework for easily developing concept models using different approaches (dictionary or pattern-based, etc.), and
- The Knowledge Gathering and Synthesis (KoGS) Framework for exploring results.

Sections 2 through 4 provide the requisite background and motivate our vision and the approach we took in this work. In Section 5, we discuss the analytics development process a practitioner follows. Sections 6 and 7 describe the CAT and KoGS frameworks which are central to the extensibility of UDAT. We provide a walk-through scenario in Section 8 that highlights the ease-of-use of UDAT and gives the reader an understanding of many of the capabilities of the system from the point of view of a practitioner who is not an NLP expert. Section 9 provides brief system notes. We end with a discussion and some concluding thoughts in Section 10.

2. UIMA Configurable Annotators

Frameworks like UIMA (Ferrucci & Lally, 2004), GATE (Cunningham, 2002), NLTK (Bird, 2006) facilitate a decomposition of the analysis process into individual components, each of which is responsible for a certain aspect of the analysis (Cunningham & Scott, 2004).

Broadly speaking, analysis components (also referred to as *annotators* or *analysis engines*) find and annotate a number of concept instances: domain or application specific named entities, their properties, and/or relations among them (e.g. Person, Company, Location, Product, ManagerOf, etc). Assembling applications in such frameworks involves building processing pipelines from annotators organized sequentially, or in a more complicated flow.

We focus on systems in which models[1] are created to represent a concept, which can then be used by an annotator to find instances of the target concept. A very simple model for a "City" concept would be a list of city names. Common model types include simple term lists, (semantic) dictionaries, grammars, pattern files, and statistical models. Looking at this broad variety, we seek to assist end-users in developing models and configuring the model-based annotators, and composing a coherent set of these into a pipeline.

Domain experts are likely to be unfamiliar (or less comfortable) with the notions of analysis engines, NLP pipelines, and the practical skills required to build complete annotators and applications based on those notions. A challenge, then, is to allow their domain knowledge and insights to be interpreted by a tool which, in effect, mediates between an abstract description of how a concept may be expressed (*e.g.* in text) and a specification for finding instances of such expressions by applying an appropriate analysis engine.

3. The Ontology-Centric Approach

We present an ontology-centric approach for analytics/annotators development. Our design associates concepts with concept models, and delegates the interpretation of models to suitably configured *meta-annotators*. A meta-annotator is a reusable analysis component, which can interpret concept models as abstract specifications for its behavior. Examples of meta-annotators are a dictionary lookup component (whose dictionary may describe a set of semantically meaningful terms in a domain), or a pattern matching engine (which interprets a set of grammars to find patterns of expressions), or a classification module (which applies a separately trained model to the data). Meta-annotators, also called *configurable annotators*, crucially maintain separation between the models for the concepts they annotate, and the underlying analysis engines which apply the models. It follows that the meta-annotator abstraction is a useful one, for the purpose of insulating domain experts from run-time details, and enabling them, and us, to focus on model elicitation and formulation.

Our runtime framework is provided by UIMA (Ferrucci & Lally, 2004), the Unstructured Information Management Architecture, which defines mechanisms and interfaces for the purposes of defining, configuring and running technology-agnostic multi-component NLP annotators over text, speech and other multi-media. UIMA has been gaining broad acceptance both in industry contexts (Dale, 2005) and in academic communities (see the UIMA Component Repository at CMU[2]; recently, it has become widely available as an Apache Open Source Incubator project[3], and is currently the subject of an Open Standard initiative[4].

4. The Consumability Challenge

From the point of view of domain experts there are obstacles to overcome before a fully functional application emerges from the set of domain concepts in which they traffic. Not only do the models (in the sense described above) need to be developed, in order to compose a complete application from individual components, but individual models will also need to be tested and debugged against representative corpus data.

This means that mechanisms need to be in place for readily associating a state of the model with any particular combination of test/development resources and analysis results. This is a necessary part of the full cycle of model development.

In general, we refer to the set of challenges presented by the complexity of such operations as the framework "consumability" challenge. Our objective here is to minimize the expertise required and difficulty that is encountered throughout the process thus increasing the consumability, or ability of practitioners to engage in the process.

A goal for meeting this challenge is to provide a set of configurable annotators that require little or no programming. "Configuring" these annotators is accomplished through the direct manipulation of graphical widgets that in the end result in executable annotators. Our focus in UDAT is ease-of-use for novice users, powerful capabilities for expert users and a commitment to an extensible architecture through the CAT and KoGS frameworks.

5. The Analytics Development Process

Our work focuses on meeting the challenge of consumability through the design of UDAT, which assists users throughout the process of *full-cycle* analytics development. Figure 1 illustrates this process. Beginning at the outer far left and proceeding clockwise we observe the process steps performed by the user:

[1] There may, and likely will, be multiple models for a single concept.

[2] http://uima.lti.cs.cmu.edu:8080/UCR/Welcome.do

[3] http://incubator.apache.org/uima/

[4] http://www.oasis-open.org/committees/uima/

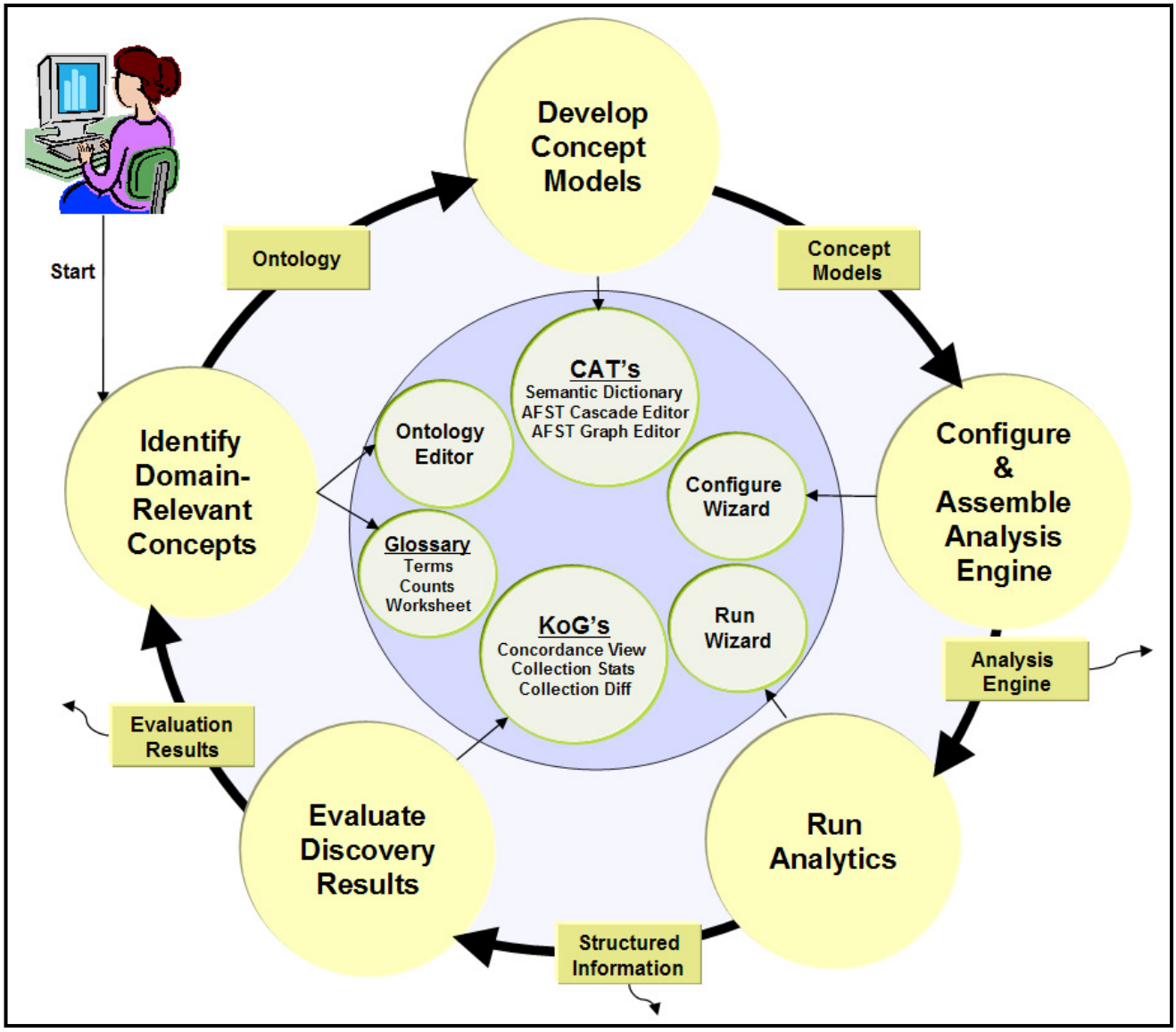

Figure l: The Analytics development Process

1) **Identify** domain relevant concepts and produce an *Ontology*,
2) **Develop** executable *Models* for each concept,
3) **Configure** and assemble the annotators into an Aggregate *Analysis Engine*,
4) **Run** the Analysis Engine on the unstructured content to extract *Structured Information,* and
5) **Evaluate** the quality of the *Results* derived from the structured information.

The user may iterate through the process, possibly starting at different points in the cycle.

We have developed a set of graphical tools for each step that abstract and simplify the process, which can otherwise be complicated and time consuming, requiring specialized knowledge of algorithm techniques, programming languages and processing content using the underlying UIMA framework. We illustrate the UDAT tooling that is associated with each process step within the inner circle of Figure 1. Section 8 will illustrate, through an example scenario, how each of these tools is used.

Before a meta-annotator is configured for composition within a larger engine/pipeline, its model needs to be developed and tested. Therefore, UDAT also supports the notion of a *tight loop*, which is a set of steps focused on the incremental development and testing of a specific meta-annotator model. This tight-loop pathway places heavy requirements on model editors, model generators, annotator pipeline builders and special purpose viewers for results exploration.

6. Developing Analytics using CATs

UDAT provides a visual editor for each model type that serves as a specific development environment for the models of that type (*e.g.* dictionary-based models, pattern-based models). In the back-end of the editor a model generator component interprets the UI gestures to generate a concrete model, typically, within a set of notational and/or representational devices (e.g. XML

dictionary tables, a grammar language, a trained model). The model editor, the model generator, and the configurable annotator are bundled into a Configurable Annotator Tool component (CAT). A CAT is a pluggable architectural component, which enables the development of a certain type of models, configuring the corresponding analysis engine, and applying the model to a corpus. A distinguishing characteristic of UDAT is that it provides the framework that allows additional CAT components to be plugged in; the framework then manages their interactions with each other, and with other parts of the toolkit.

The CAT framework is thus essential to UDAT's extensibility. To support a new model type, the toolkit can be extended by adding a new pluggable CAT for that new model type. Users can choose from different CATs to develop concept models best suitable to application needs. UDAT currently provides three CATs (see section 8).

7. Exploring Analysis Results using KoGS

In order to get meaningful insights from the analysis results, the collection of annotation metadata produced for the data corpus is processed to generate additional structured information and use it for advanced results exploration. Examples of such structured information include indices for enabling semantic search over the results, and databases of statistics about the concepts and instances found in a corpus.

The advanced viewers of this kind fall into another category of architectural components called Knowledge Gathering and Synthesis tools (KoGS). The KoGS framework affords plugability of tooling that contains

1) a back-end for building structured information (*e.g.* creating indices, databases, etc.), and
2) a front-end for exploring results (*e.g.* Results Viewers).

KoGS tools provide a broad set of functions, largely via the different Viewers, needed for exploring and evaluating the results, including tools to query the analysis results, such as tools for semantic search and database lookups.

UDAT can be extended by adding KoGS tools that provide new Viewers to facilitate results analysis evaluation. The plugability provided by the CAT and KoGS frameworks affords an open-ended architecture that supports development of components by third parties as well as those developed in-house.

8. A Walk-through Scenario

In this section we illustrate the usage of UDAT through an example in which the domain analyst is not an NLP expert. The task facing the user is to extract "car sentiment" phrases from a document corpus which consists of 80 short car reviews collected from various automotive web sites. The user starts to tackle this task by defining three concepts:

- *Sentiment* (*e.g.* being impressed, liking, hating, loving, etc.),
- *CarFeature* (*e.g.* pedals, engine, brakes, transmission, ride, etc.), and
- *CarSentiment* (*e.g.* being impressed with the adjustable pedals).

Figure 2 informally illustrates the kind of analysis that the domain expert/user would like to implement, a sample sentence, and the annotations we would expect our analytics to produce.

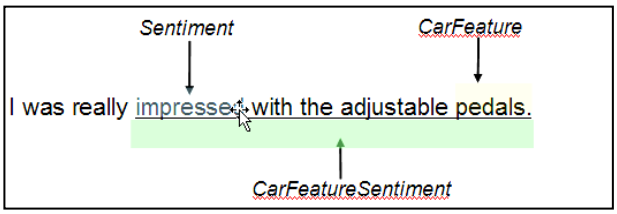

Figure 2: Example of annotations

8.1 User Interface Overview

Figure 3 is a screenshot taken from UDAT and will be used throughout the example scenario discussion. It consists of 4 panes labeled 1 through 4. Pane 1 is called the Explorer and is the place where the user manages the four primary UDAT abstractions: *Concepts*, *Documents*, *ConceptFinders* and *Results Sets*. Pane 2 is the Editor pane and is the place where the user builds concept models. An Editor is available for each type of model that can be built in UDAT and always appears in this location. The space occupied by the panes labeled 3 and 4 is used to explore results using KoGS tools.

8.2 Concept Definition

The first process step is to define the concepts which the expert elucidates as representative for his/her domain. In UDAT, a domain is represented a the root node of a hierarchy of concepts. An ontology editor (pane 1 in Figure 3) is used to manipulate the hierarchy of concepts in the domain. In our scenario, the user defines the "Automotive" domain and then adds the three requisite concepts under it as illustrated. Each concept may have one or more associated concept models. The toolkit is pre-loaded with a configurable annotator tool for each different model type. We currently support three CATs:

1) A Semantic Dictionary CAT which enables the graphical development of concepts that can be defined by listing terms that represent instances of the concept.
2) An Annotation-Based Finite State Transducer (AFST) (Boguraev & Neff, 2007 and 2008) CAT provides the ability to graphically create models that consist of patterns that consider upstream annotations as well as other features of the text.
3) An AFST Cascade Editor CAT that allows advanced users to develop and manage sequences of AFST grammars written in the AFST language.

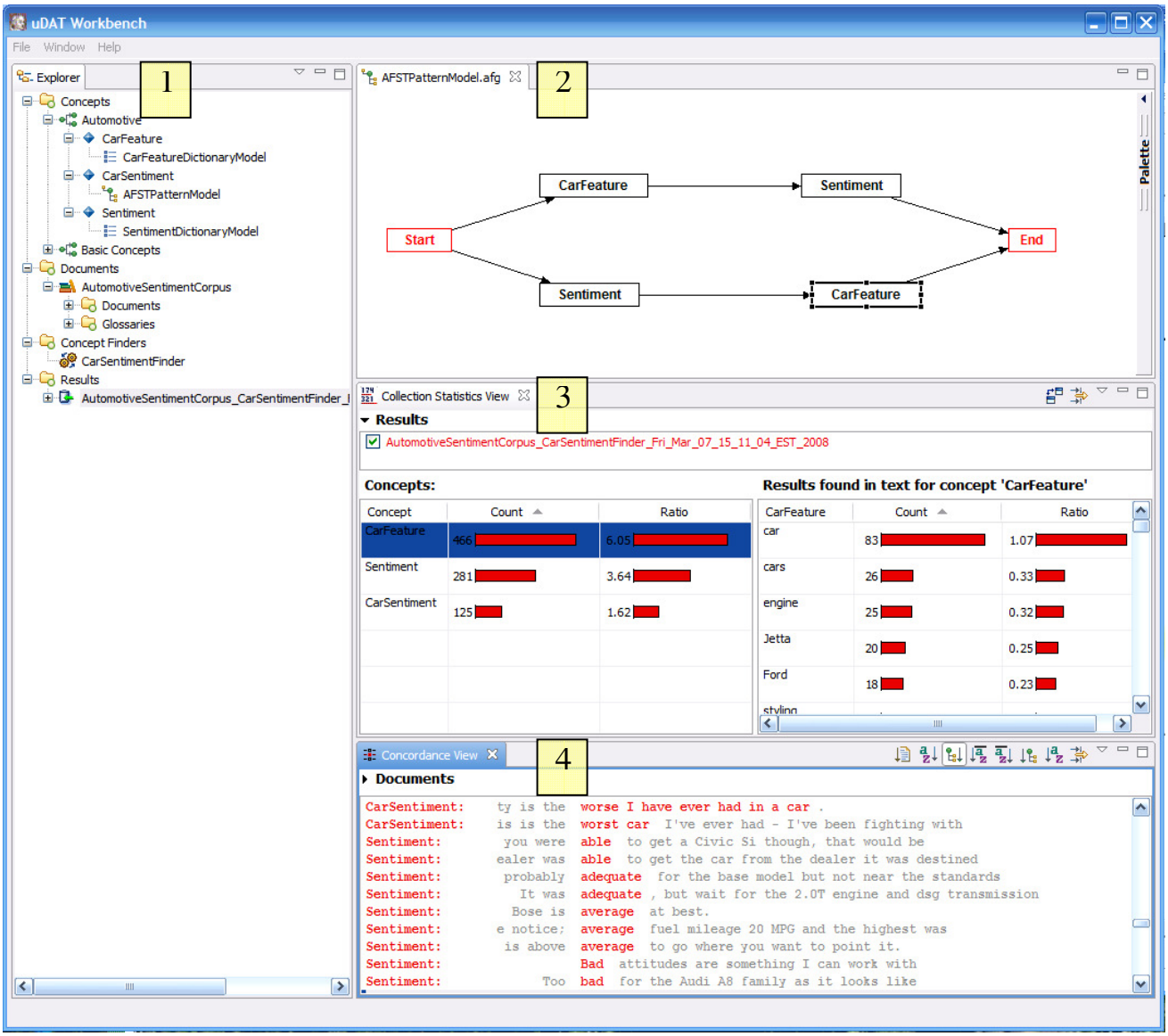

Figure 3: UDAT User Interface

The *Concepts* are allowed to have multiple models (of the same or different model type); this enable combining different underlying technologies associated with the model types. For example, a concept can have any number of Semantic Dictionary models as well as AFST models.

8.3 Corpus Specification

A Corpus is a collection of documents to be analyzed. Users sometimes define corpora by extracting documents that contain a specific concept (or concepts) of interest. This allows them to focus upon the specific task at hand (e.g. development of an annotator for a specific concept) without the burden of analyzing a large corpus. Users are free to define corpora to meet the needs of the analytic development process.

The next step in our example scenario discussion is to specify the Corpus. This is accomplished using the Explorer - by selecting the Documents Folder, defining a new Corpus and pointing it to the "Automotive" documents.

8.4 Corpus Exploration

Once a corpus is defined, the user can run the GlossEx KoGS tool (Park et al., 2002) to extracts the domain specific terms and phrases from the corpus. It provides a frequency count, the part of speech, and a metric for the domain specificity for each term/phrase. Exploration using this tool provides the user with the terms and phrases that are appearing in the corpus and allows them visibility into the lexical data that represent

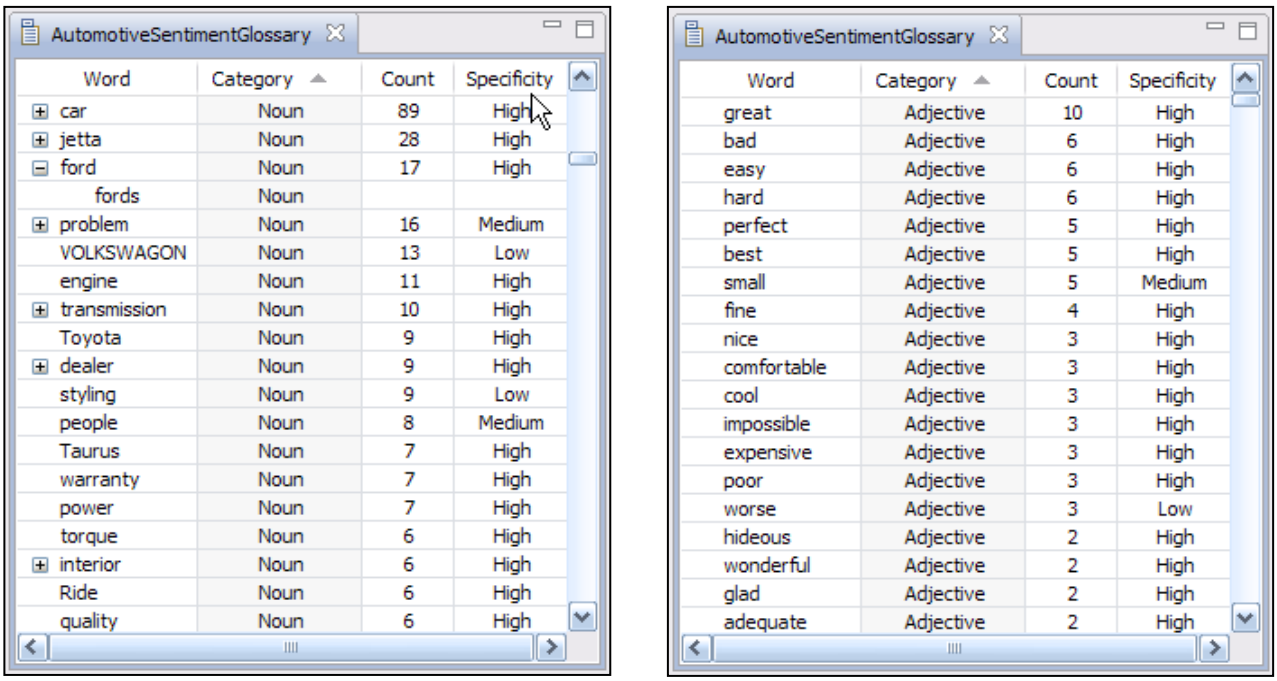

Figure 4: Extracted Glossary

the concepts to be found. Users can analyze these data to help them articulate *concepts* and *instances*, as well as position them in an ontological hierarchy. The GlossEx KoGS tool is tightly integrated into the UDAT environment and affords a wide range of capabilities for working with domain terms, and building concepts, ontologies and associated Semantic Dictionary models. It presents data in a spreadsheet form that contains both system-defined as well as user-defined columns.

Figure 4 shows some of the resulting nouns and adjectives that are most frequently occurring in the corpus. Note that a number these *nouns* represent the *CarFeatures* we are interested in and *adjectives* largely indicate the *Sentiments*.

8.5 Concept Model Creation

Dictionary Models can be easily created for *Sentiment* and *CarFeature* concepts by highlighting the terms of interest in the GlossEx viewer and dragging and dropping them onto the respective concept model in the Explorer. In this process step the user defines a CarFeature Dictionary Model and a Sentiment Dictionary Model and then drags and drops the appropriate highlighted data into the respective model. The Dictionary Model (CAT) graphical editor affords all necessary operations for building a dictionary model for a given concept. This also includes access to WordNet[5] for getting suggestions on variants of a term.

The AFST CAT is used to define patterns that specify the concept model of *CarSentiment*. The AFST CAT provides a graphical editor that abstracts the underlying AFST syntax and allows the user to express AFST patterns graphically. Pane 2 on Figure 3 illustrates the process of building an AFST pattern. This is easily accomplished by dragging and dropping concepts from the Explorer onto this pane and interconnecting them with arcs. The semantics of this graph specify that the span of text delimited by the pattern {*CarFeatures* followed by *Sentiment*} or {*Sentiment* followed by *CarFeatures*} will be annotated with the concept *CarSentiment*.

User interactions with a CAT graphical interface result in an intermediate representation tailored for graphical editing that captures the specification of the intended model semantics. This intermediate representation is transformed by the CAT *Model Generator* to produce a CAT *Meta-Annotator Model* suitable for running on a CAT *Meta-Annotator*. This process is transparent to the user and handled by UDAT.

8.6 ConceptFinder Generation

Once developed, sets of user-selected models drive the automatic definition of aggregate annotators that are used to find concepts of interest. We call this aggregate annotator a ConceptFinder. To create a ConceptFinder the user simply selects all the concepts to be found from the Explorer and assigns a name used to refer to it. This selection action is a signal whereby the user delegates to UDAT the task of composing ConceptFinders, configuring underlying meta-annotators and ensuring coherence of the underlying framework pipeline. In the Explorer pane (pane 1 in Figure 3) we show the concepts and resulting ConceptFinder called *CarSentimentFinder*.

UDAT hides a number of complex issues associated with the generation of ConceptFinders. The most challenging one is the dynamic construction of a complete deployable

[5] http://wordnet.princeton.edu/

and ready-to-run NLP pipeline of annotators given simply a set of concepts and models as input. Annotators may require other annotators to run before them in the pipeline, in order to detect concepts upon which the target concepts depend. Such dependencies translate into complex ordering constraints among the available annotators for constructing the pipeline. The challenge is to construct a coherent and well-configured annotator pipeline which satisfies complex constraints. This is further compounded by the possibility that some constraints will not be explicitly declared.

Full architectural discussion of UDAT is outside the scope of this paper. Here we briefly mention that this is informed by the need to analyze the network of dependencies between concept models in the underlying pipeline construction process. This analysis, complemented by the use of pipeline templates enables the configuration of ConceptFinders.

8.7 Results Generation and Exploration

Concept finders can be run over data: a single document, or an entire corpus, or previously saved analysis results (whether these have been generated within the tool's development cycle, or imported from an outside analysis process). Results sets are explicitly managed by UDAT and are key enablers to flexible evaluation and experimentation regimes themselves crucial for quality analytics development.

The results data are generated and stored in the file system as well as in embedded databases. The basic part of the results consists of annotated documents that can be viewed using an annotation viewer which suitably displays (by means of highlighting or other appropriate visual metaphors) the annotations, and the features for the concept instances found in the corpus.

For this scenario, the user can simply select a ConceptFinder, (e.g. *CarSentimentFinder), a* corpus (e.g. *AutomotiveSentimentCorpus*) and the KoGS tools that they would like to use to explore the results. UDAT manages all actions related to the UIMA framework - running the ConceptFinder over the corpus, as well as building all structured repositories required by each of the KoGS tools requested. Once processed, each document can be browsed using the Annotation Viewer (a standard KoGS tool) to explore annotations and their respective feature.

KoGS tools afford exploration of results in novel ways. In Figure 3 (panes 3 and 4) we show the *Collection Statistics* and the *Concordance* Viewers provided by the respective KoGS tools. The *Collection Statistics* Viewer has 2 panes. The Statistics Viewer presents the *Concepts* found, their frequency in the corpus and the average of specific concepts per document. For example, there were 466 *CarFeature* annotations occurring on an average of 6.05 times per document. The second pane gives a further breakdown for the concept selected and indicates

1) the specific terms that were found,
2) their frequency count in the corpus, and
3) the average occurance per document.

For Example, the term *Car* occurred 83 times in the corpus and on average appeared 1.07 times per document. This Viewer allows a user to quickly evaluate the effectiveness of the concept models that they create against a particular corpus. The Statistics Viewer can display up to seven runs to be compared side by side (from prior saved runs). This provides at-a-glance user-selected comparisons that are crucial in the *tight-loop* for evaluating the effect of iterative development of a concept.

The *Concordance* Viewer (Fig. 3 - pane 4) is a KoGS tool that allows users to explore annotations by aggregating them in a variety of ways, in the context, in which they occur. The tool bar on the top right (8 icons) provides eight sorting and viewing possibilities. The sorting option shown in the figure displays the concepts on the left of the pane sorted in alphabetical order followed by the textual context of the term with that annotation. The term is shown in red and includes some distinguishing context to the left and to the right. Terms can also sorted alphabetically within a concept allowing the concurrent viewing of the context of multiple sentences that contain the same annotation and term. Another KoGS provides the 'Difference View', through which the analyst can compare in detail the difference between two similar analyses, by looking at the concordances produced by each (not shown in the figure).

Both the *Concordance* and the *Collection Statistics* Viewers have the ability to present comparisons of results derived from different runs. The interplay of the *Collection Statistics* Viewer, the *Concordance* Viewer, and other result Viewers provides a set of exploration tools that are instrumental in the evaluation and incremental improvement of analytics. Providing a rich set of these Viewers as well as the ability to view them together has been an important design point in UDAT.

9. System Notes

UDAT was developed as an Eclipse [6] Rich Client Platform [7] (RCP) application using the JAVA programming language. The plugin architecture of the Eclipse framework serves as a low-level robust foundation for making UDAT extensible by adding plugins, as needed. Each plugin may provide a certain feature of the UDAT framework, or may wrap architectural components such as CATs and KoGS components. The Graphical User Interfaces are based on Eclipse JFace[8] and SWT[9].

[6] http://www.eclipse.org
[7] http://wiki.eclipse.org/index.php/Rich_Client_Platform
[8] http://wiki.eclipse.org/index.php/JFace
[9] http://www.eclipse.org/swt/

UIMA is used as the underlying runtime framework for running the annotators in pipelines. The UIMA annotators, provided by the CATs, and the KoGS components might be implemented using independent technologies. However, once these technologies are wrapped in the UDAT architectural components, UDAT provides seamless interoperability between what started as an independent set of tools and technologies. For instance, UDAT supports a flexible flow through the full cycle development which might combine a number of CATs, KoGS components, and other elements which are based on technologies that would not normally "talk" to each other, otherwise. The detailed description of the UDAT architecture is outside of the scope of this paper, but may be the subject of a subsequent paper.

10. Discussion & Conclusion

The UDAT vision has generated a set of key requirements and ideas that have driven its design and implementation. These include:

1) Consumability
2) Extensibility
3) Full Analytics Development Cycle Support
4) Ontology-Centric Development Support
5) Re-usability of Concepts and Analytics through Configurable Annotators

The example use-case described in section 8 focused on the rapid development of text analysis applications by domain experts that are not expert NLP practitioners. This is one of UDAT's main strengths. UDAT is, however, also designed to support users of varying levels of expertise. In addition, users can import results derived from UIMA analytics that were produced outside of the UDAT environment, and can build new concepts using imported annotation meta-data. The wide range of user support highlights our focus on **Consumability** – a requirement to provide a rich set of easy to use tools that a practitioner can apply depending upon their level of expertise as well as the technical challenges that the problem at hand brings to the analytics development.

The CAT and KoGS frameworks are design points in UDAT that allow developers to easily plugin new *configurable annotators* for adding new analytics capabilities as well as adding new tooling that provide a rich set of exploration possibilities through Viewers. These broadly distinct, yet fully interoperable, categories of architectural components account for the diversity and open-endedness of function supporting domain-focused model development and application configuration. This allows for the layer of abstraction above the implementation and configuration details layers, which ultimately facilitates the domain expert users in building elaborate and deployable concept models. These frameworks support our requirement for **Extensibility -** the ability to easily add new pluggable components into UDAT to extend its functionality.

Yet another important aspect of UDAT is our focus on understanding the full cycle of user task models that a practitioner engages in during the **Full Analytics Development Cycle**.

Concept re-use has been a key requirement resulting in the pivotal design point of **ontology-centric development**. Users define ontological concepts and build analytic models for finding these concepts using techniques that are provided through configurable annotators. New concepts can be built upon previously developed concept models. This layering facilitates the creation of rich and complex concept analytics that are constructed out of simpler models.

Concepts and their associated models can be assembled into *ConceptFinders* from repositories of domain independent and domain specific ontologies. The composability of existing concept analytics allows users to build applications in new domains through the re-use of existing concept repositories. These repositories provide **Re-useable Intellectual Capital** that can be deployed in a variety of established or new business settings that bring business value to a broad range of enterprises.

In this paper, we have presented the UDAT vision, key design points and many of the features that are currently implemented. We believe that UDAT provides a powerful environment for rapidly developing analytics and deploying applications in broad range of domains. We are currently evaluating this hypothesis by using UDAT in a range of problem domains in IBM research.

11. References

Bird, S. (2006). NLTK: The Natural Language Toolkit. In *Demonstration Session, 45th Annual Meeting of the ACL*, Sydney, Australia.

Boguraev, B., Neff, M. (2007). An annotation-based finite state system for UIMA: User Documentation and Grammar Writing Manual, *IBM T. J. Watson Research Center*, Yorktown Heights, New York.

Boguraev, B., Neff, M. (2002). Navigating through Dense Annotation Spaces. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation LREC,* Marrakech, Morocco.

Cunningham, H. (2002). GATE, a general architecture for language engineering. *Computers and the Humanities,* 36, pp. 223--254.

Cunningham, H., Scott, D. (2004). Software architectures for language engineering. *Special Issue, Natural Language Engineering*, 10(4).

Dale, R. (2005). Industry watch. *Natural Language Engineering* 11, pp. 435--439.

Ferrucci, D., Lally, A. (2004). UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering, 10, Special Issue on Software* .

Moore, C. (2002). Diving into Data, *InfoWorld, October* 25.

Park, Y., Byrd, R. and Boguraev, B. (2002). Automatic glossary extraction: beyond terminology identification. In *Proceedings of the 19th International Conference on Computational Linguistics,* Taiwan. pp. 772--778.