Proceedings of the International Conference on Electrical and Computer Systems
Ottawa, Ontario, Canada, 22-24 August 2012
Paper No. 203

Adaptive Rules-Driven Architecture for Knowledge Extraction (ARDAKE) with UIMA

Wassim El-Kass, Stéphane Gagnon, Michal Iglewski
Université du Québec en Outaouais
101, rue Saint-Jean-Bosco, Gatineau, Québec, Canada J8Y 3G5
elkw01@uqo.ca; gagnst02@uqo.ca; iglewski@uqo.ca

Abstract - We present a prototype named Adaptive Rules-Driven Architecture for Knowledge Extraction (ARDAKE). Our environment integrates a Business Rules Engine (BRE) or Semantic Rules Engine with the Unstructured Information Management Architecture (UIMA). It provides domain experts with a simple rules-driven framework to develop context-aware, adaptive knowledge extraction systems. Our approach differs from existing workflow oriented annotation environments that allow the insertion of annotators into static workflows, whereas the end results in our case is a reusable UIMA annotator with an auto-generated dynamic workflow.

Keywords**:** Text Mining, Unstructured Information, Knowledge Extraction, UIMA, Workflow Systems

1. Introduction

The Unstructured Information Management Architecture (UIMA) has recently gained unprecedented attention in the domain of knowledge extraction, following the victory of IBM Watson, running DeepQA, a question-answering system based on UIMA, against two Jeopardy champions. Watson's success depended greatly on being able to control when and how text annotators are executed to ensure efficient knowledge extraction, with multiple components acting on large corpora and relying on the results of other annotators to ensure semantic integrity.

We present a prototype aiming at further increasing the effectiveness of such knowledge extraction systems. Our applied research project, named Adaptive Rules-Driven Architecture for Knowledge Extraction (ARDAKE), provides a new development environment integrating a Business Rules Engine (BRE) or Semantic Rules Engine with UIMA. It provides domain experts with a simple rules-driven framework to develop context-aware, adaptive knowledge extraction systems.

ARDAKE offers a more user-centric approach focused on easier creation and manipulation of complex UIMA annotators. Precondition and post-processing rules are used to ensure more expressive, automated, and adaptive knowledge extraction.

We begin this paper with a discussion of the knowledge extraction research context. We then survey existing UIMA-based platforms for rules-based and workflow-oriented information extraction. We finally present our proposed architecture for integrating new adaptive rules-driven annotation capabilities within the UIMA. We conclude with a discussion of future research on dynamic rule testing and validation.

2. Towards Rules-Driven Adaptive Annotation

2. 1. Unstructured Information Management Architecture (UIMA)

UIMA is a component architecture and software framework implementation for the analysis of unstructured content like text, video and audio data. For example, UIMA can be used to process business news and identify entities, such as companies, owners, etc., as well as relationships among entities, such as merger or acquisition.

The core of UIMA is the Analysis Engine (AE), a collection of Analysis Components (or Annotators). A primitive AE contains only one Analysis Component, and the UIMA Software

Development Kit makes it easy to statically combine AEs to form an Aggregate Analysis Engine. This is done through an XML descriptor where no code is required. For more complex (non-sequential) annotation workflows, flow controllers can be inserted into an Aggregate Analysis Engine to determine the order in which the components of the aggregate are invoked. The UIMA framework runs AEs by looping through their Analysis Components and calling their methods in a well-defined order.

While creating primitive annotators is relatively simple, creating aggregate annotators that require more than a simple sequential execution of their component annotators require many additional steps. Those steps include writing and compiling the code that defines the actual execution flow. This makes aggregate annotators harder to implement and maintain, and less adaptable to changes because changing the execution flow requires code changing and recompiling.

To further increase the adoption of UIMA and the reusability of annotators, it must be fairly easy to create and modify aggregate annotators even when the execution flow of their component annotators is complex. This can be achieved by automating the creation of annotators' execution flows based on domain-specific rules.

2. 2. Integration, Validation, and Monitoring of Annotation Rules

We rely on a Business Rules Engines (BREs) to evaluate two types of annotation rules. The first type of rules is to define annotators' preconditions, allowing the system to determine whether or not an annotator should be executed at a given time. Precondition rules can also be used to instruct the BRE to do some initializations before running an annotator. The second type of rules is to define post-processing steps in order to set some properties or environment variables after executing an annotator.

Our approach encourages, and makes it easy, to use ontologies to guide the execution of the annotation process and to assign meanings to extracted entities. The BRE can be substituted with a Semantic Rules Engine when ontologies are used to define preconditions and post-processing rules. Using ontologies to define preconditions and post-processing rules covers two of the three main characteristics of Ontology-Based Information Extraction Systems (OBIES) as defined in (Wimalasuriya and Dou 2010). The third main characteristic is covered if annotators are extracting information from unstructured or semi-structured natural language text which is the case for most UIMA annotators.

Although some BREs are able to detect conflicts in business rules, it would be good to detect and resolve conflicts at design time (i.e. when creating the rules). Cycles or circular dependencies among annotators should also be prevented at design time by the business rules composer. This can be a challenging task especially when complex rules are required for a large number of annotators. We plan to use a semantic reasoning engine such as Pellet to help with the validation and the detection of inconsistencies in annotators' rules especially for ontology-based rules.

Another desirable feature is to be able to monitor the annotators' execution process to make sure the right annotators are called at each step and that annotators are producing the expected results. The monitoring can be enabled by having special rules to raise informational and exceptional events and have an external entity capture and analyse those events. An example of an informational event is to signal the truthfulness of an annotator's precondition which indicates that the annotator is ready to run. An exceptional event can be raised if, for example, an assert rule to check the value of an annotation feature does not pass.

3. Survey of Existing UIMA-Based Annotation Platforms

3. 1. Rules-Based and Workflow-Oriented Information Extraction

Workflows and business rules are now popular for computational knowledge extraction and are used to enable science on a large scale by managing data preparation and analysis pipelines (Goble and De Roure 2009). Some information extraction workflow platforms were developed to simplify

the complex task of information and knowledge extraction, especially for non-developers. Workflows creation still requires some level of programming skills that domain experts may not have. Rules-based platforms can therefore be better alternatives for non-technical users. The following subsections present some UIMA-based platforms for creating or simulating information extraction workflows.

3. 2. TextMarker

Built upon UIMA, TextMarker is an open source rule-based tool for information extraction and text processing tasks (Kluegl, Atzmueller et al. 2009). Knowledge engineers can use the tool to identify common text patterns that are annotated to create extraction rules. The corpus is then processed for further extraction and rule validation as test set.

Rules are built using a specialized representation language for knowledge formalization, with the possibility of creating new annotation types and integrating them into a taxonomy. They can either be extracted directly from unstructured sources or coded in the scripting rules language. Each rule can be defined within a rule set or type, and associated with specific patterns and actions. Robust rules can be extracted using scoring and filtering to ensure accurate matching of rules to annotation targets.

3.3. U-Compare and Taverna

U-Compare is an integrated text mining/natural language processing system based on the UIMA Framework. U-Compare offers a GUI for easy drag-and-drop workflow (UIMA component descriptor) creation, a comparison by U-Compare parallel component, evaluation, statistics and visualizations. U-Compare workflows have the same limitations that UIMA aggregate components have; that is, only linear fixed flow and language dependent flow are available for use. For more complex execution flow, code is required.

The U-Compare integration within the generic workflow system Taverna offers more capabilities by enabling more flexible and richer annotations and text mining workflows (Goble and De Roure 2009; Kano, Dobson et al. 2010). However, those workflows are static Taverna workflows that cannot be reused by any other UIMA compatible system.

3.4. IBM LanguageWare

IBM LanguageWare allows extending UIMA annotations using a rules engine. The LanguageWare rules engine supports three kinds of matching rules:

- Break Rules: specify how documents are split into lexical components such as paragraphs, sentences and tokens. A token can be anything from a word to a punctuation symbol, a number, a currency, etc….
- Character Rules: these are characters expressions used to match desired sequences of characters such as postal codes, telephone numbers, email addresses, and so on. The rules engine creates annotations when text sequences are found to match the character expression.
- Parsing Rules: use textual patterns from custom dictionary entries and different parts-of-speech, in addition to some previously created annotations.

LanguageWare allows adding features to the newly created annotation type by dragging and dropping features from existing annotations. However, LanguageWare works only on already annotated documents (for reusing existing annotators) and does not support features' computations (only drag-drop of features is supported) nor does it allow Boolean conditions for creating new annotations (IBM 2011).

4. Adaptive Rules-Driven Architecture Knowledge Extraction

4.1 Integrating a BRE with UIMA

UIMA allows specifying annotators' input and output in XML format using the descriptor editor but does not enable specifying the preconditions for running those annotators nor does it enable

specifying post-processing steps to run once the annotator finishes executing. Preconditions and post-processing in UIMA are assumed to be part of the annotator itself or the flow controller in case of an aggregate annotator. In both cases, preconditions and post-processing events are hardcoded (except for trivial predefined cases) and changing them requires changing code, recompiling, and redeploying.

We propose adding preconditions and post-processing rules to UIMA annotators using business rules and/or semantic rules. As shown in Figure 1, our prototype's interface helps defining rules graphically using our UIMA Rules Composer. Each rule in ARDAKE has a condition and an action part. If rules are built from ontology concepts and their relationships, they can be encoded in any semantic rules language (e.g., SWRL, RIF). Conditions are Boolean expressions and can be based on contextual or configuration information as well as on previous annotations and their features. Actions can be anything from creating new annotations on the fly to setting or calculating annotations' features values, or calling external web services. A rules engine evaluates rules conditions and handles actions when conditions are met.

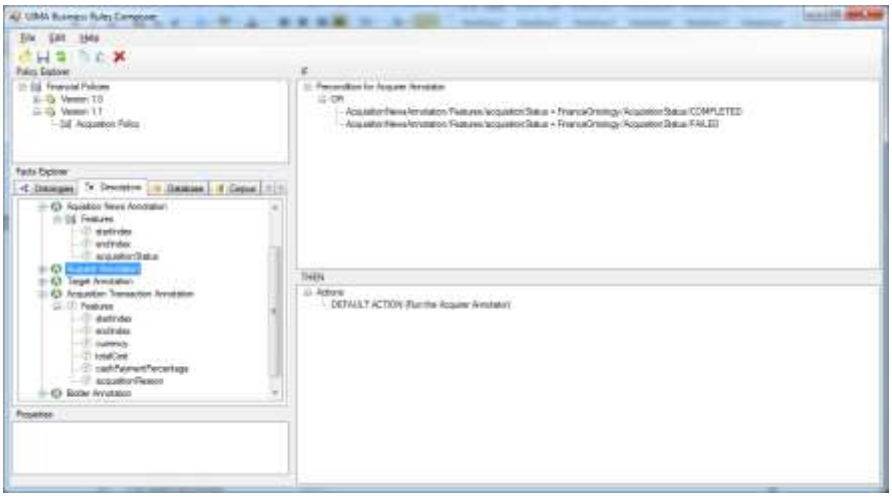

Fig. 1. ARDAKE's Interface - Adding Preconditions

The annotator's precondition rules are evaluated prior to calling the annotator; conditions in this case are meant to determine whether or not the annotator should be called. The actions of precondition rules are more to perform some initializations (pre-processing) before running the annotator.

Post-processing rules of an annotator are evaluated once the annotator finishes executing and can be used to validate the annotator's results (ex. Assert the existence of certain annotations or the values of some features) as well as to run post-processing actions such setting the values of some features or mapping annotations to concepts in ontologies. This helps separating the annotator business logic from its matching logic.

As a simple example, consider the case of a price annotator that matches product prices in text and has a tax feature for showing the taxes estimate for the matched price. UIMA allows passing parameters to annotators but the annotator code should implement its logic using those parameters. To set the value of the tax feature, the tax calculation formula can be passed as a parameter to the price annotator making the annotator quite generic. Modifying the tax calculation formula, in this case, can be done in the annotator descriptor's XML and does not require changing the annotator's code. However, if we decide to use a web service to calculate taxes, the annotator will have to be modified. Externalizing the business logic from the annotator itself makes the annotator code simpler, easier to maintain, easier to modify by

business users, and less likely to change over time. This can be done by handing any business logic task to the business rules engine.

4.2. Adaptive Annotators

ARDAKE's approach enables the automatic generation of dynamic annotators' execution flows (execution plans). It also helps creating better, simpler, more efficient and more reusable UIMA annotators by externalizing the annotators' matching and business logics. It also allows business users to visually create and maintain rules-based annotators without having to write any code.

Unlike U-Compare/Taverna workflows, the end result in our case is a UIMA annotator that can be re-used by other more complex aggregate annotators or by any system that knows how to deal with UIMA annotators. Our solution resembles more to the matching rules of LanguageWare but goes beyond just the matching and creation of new annotations based on existing ones or some other patterns. Business rules allow creating richer and more complex annotations than those created using the LanguageWare matching rules and can, in fact, replace those rules. LanguageWare matching rules are based on annotations/expressions and do not make use of annotation features or Boolean conditions for creating new annotations. Furthermore, the tool allows adding features to newly defined annotations using drag and drop but does support conditional and computational value settings of those features.

4.3. Generating the Annotators' Execution Flow

One of the big benefits of defining preconditions for annotators is that preconditions can be used to automatically generate an execution flow that can used to replace the UIMA flow controller(s) for the annotators being executed.

Based on the way our precondition rules are defined, the following assumptions can be made:

- A precondition can have zero or more predecessor annotators. These are the annotators referenced in the condition part of the rule.
- A precondition can have one and only one successor annotator. This is the annotator for which the precondition is defined.
- Preconditions with equivalent conditions have the same predecessor annotators.
- Different annotator preconditions can have equivalent conditions. Those preconditions can be merged during the workflow building process.

We use the following procedure to automatically generate the execution workflow for selected annotators.

On the first level of the workflow, add all annotators that have no preconditions or have preconditions with no predecessor annotators (ex. preconditions that are based on some contextual and/or environment variables only).

On the next level of the workflow, add preconditions whose predecessor annotators are subsets of all previously added annotators. (This includes preconditions with no predecessor annotators).

On the next level, add successor annotators for preconditions added in step 2.

Repeat Steps 2 and 3 until all annotators and preconditions are added to the workflow. If an empty level is reached before including all annotators, it could be an indication that the remaining annotators are not needed or that they rely on missing annotators that the user should include before regenerating the workflow.

Merge equivalent conditions. This step is optional but can be done to reduce the size of the annotators' execution workflow.

4.4 Example: Financial News Annotator to Detect Mergers and Acquisitions

Consider the case of a financial news annotator, aimed at detecting mergers and acquisitions of publicly traded companies. We would like to capture different information depending on the acquisition status. For example, if the acquisition is already done, we want to know the total cost and the currency in case a cash payment is made. If the acquisition is in progress, we are interested in knowing if there is more than one bidder or if any lawsuit is involved. These parameters can be used as indicators to determine the impact of the news being analysed on parties involved in the acquisition as well as on their competitors or even on currency exchange rates in some cases.

Although more annotators can be created to identify extra knowledge in corpus, ontologies and reasoning engines can help draw conclusions based on existing annotations, given that those annotations are linked to concepts in the ontology. Note that linking annotations to concept in ontologies can be done using post-processing business rules.

For our acquisition example, the ontology might define concepts such as Company, Acquirer, Target, Acquisition, and Bidder along with their properties and relationships. Concept CompanyWithGoodShare can be defined, in the ontology, as the union of Target with a relationship to a completed Acquisition and Target with relationships to an in-progress Acquisition and multiple Bidders.

If a Company X was annotated as a Target for an in-progress acquisition and two or more Bidder annotations are identified for the acquisition, then a reasoning engine can conclude that company X is a company with good share.

An acquisition annotator can be implemented in different ways using a number of primitive and aggregate annotators running in sequence and/or in parallel. We present here one possible implementation and we show how the annotator execution workflow is dynamically generated based on preconditions using a formal process. Figure 1 shows ARDAKE's interface with an example of a precondition for the Acquirer Annotator created using our tool. This can be done with a few mouse clicks, drag and drop. Post-processing rules are created in the same manner.

Consider the case where relevant UIMA annotators are already created using our tool or any other UIMA-based tool. Table 1 shows an example of preconditions for running each annotator. Post-processing rules can be used to associate annotations to concepts in the domain ontology so that reasoning can later be done on those annotations. A post-processing action to map an Acquisition News Annotation to an AcquisitionNews concept in a Finance Ontology may look like: AcquisitionNewsAnnotation/Features/relatedConcept = FinanceOntology/AcquisitionNews

Since all post-processing rules in this particular example are similar, they are omitted from further notations.

Table 1. Predefined annotators for the Acquisition Annotation example. (Acq = Acquisition)

Annotator	**Description**	**Precondition**
Acquisition News	Determines if a given financial news article is about an acquisition. Annotations created using this annotator have a feature called acquisitionStatus that can have one of three values (COMPLETED, IN_PROGRESS, or FAILED)	None
Target	Finds company acquired	EXIST(AcqNewsAnnotation)

Acquirer	Identifies and annotates the acquiring company	AcqNewsAnnotation/Features/acqStatus = FinanceOntology/AcqStatus/COMPLETED OR AcqNewsAnnotation/Features/acqStatus = FinanceOntology/AcqStatus/FAILED
Acq.Transaction	Captures the acquisition transaction details. Creates annotations with features such as the total cost, cash payment percentage, currency, acquisition reason, etc…	AcqNewsAnnotation/Features/acqStatus = FinanceOntology/AcqStatus/COMPLETED
Bidder	Captures interested buyers	AcqNewsAnnotation/Features/acqStatus = FinanceOntology/AcqStatus/IN_PROGRESS
Lawsuit	Annotates any lawsuit related to the acquisition	AcqNewsAnnotation/Features/acqStatus = FinanceOntology/AcqStatus/IN_PROGRESS
Competitor	Highlights different competitors	AcqNewsAnnotation/Features/acqStatus = FinanceOntology/AcqStatus/COMPLETED OR AcqNewsAnnotation/Features/acqStatus = FinanceOntology/AcqStatus/FAILED

Our goal is to dynamically determine the right set of annotators to run and the order in which they should be called based on their preconditions. Post-processing rules can link generated annotations to relevant concepts in ontologies for further validation and reasoning. When annotating acquisition news, we would like different annotators to be called depending on the value of the acquisitionStatus feature of the AcquisitionNewsAnnotation generated by the Acquisition News annotator. We distinguish between three types of specific Acquisition annotators (CompletedAcquisitionAnnotator, InProgressAquisition-Annotator, and FailedAcquisitionAnnotator) each of which requires running a different subset of the annotators described in Table 1. We finally define a generic Acquisition annotator as the combination of the three specific Acquisition annotators. Note that creating any of the four annotators mentioned here is a matter of selecting required annotators, right-clicking and selecting Create Annotator then giving it a name. This will automatically generate a precondition for the newly created annotator with an existence condition for every selected annotator. Table 2 shows the four new acquisition aggregate annotators with their descriptions and auto-generated preconditions.

Table 2. The Acquisition Aggregate Annotators. (Acq = Acquisition)

Annotator	**Annotators Used**	**Auto-Generated Precondition**
Failed Acquisition	- Acquirer - Competitor	EXIST(AcquirerAnnotation) AND EXIST(CompetitorAnnotator)
Successful Acquisition	- Acquirer - Competitor - AcqTransaction	EXIST(AcquirerAnnotation) AND EXIST(CompetitorAnnotation) AND EXIST(AcqTransactionAnnotation)
In Progress Acquisition	- Bidder - Lawsuit	EXIST(BidderAnnotation) AND EXIST(LawSuitAnnotation)
Acquisition Annotator	Integrates the 3 previous Aggregate Annotators	EXIST(FailedAcqAnnotation) OR EXIST(SuccessfulAcqAnnotation) OR EXIST(InProgressAcqAnnotation)

Based on the preconditions in Tables 1 and 2, and by applying the procedure described in section 4.3, the first level of the auto-generated execution workflow contains only the Acquisition News Annotator since this is the only annotator with no preconditions or having preconditions with no predecessor annotators. We can now, on the second level, add all preconditions that depend on the Acquisition News Annotator or don't depend on any annotator. At level 3, add annotators that constitute the target of all preconditions added at level 2. At level 4, add all preconditions whose predecessor annotators are already in the workflow (levels 1 and 3) and so on.

At the end, all equivalent conditions are merged and we get, as in Figure 2, a formal auto-generated annotators' execution workflow that the BRE can use to decide at each step what annotator(s) to run next. Users can manually update the auto-generated workflow and save a copy of the modified workflow before passing it to the BRE.

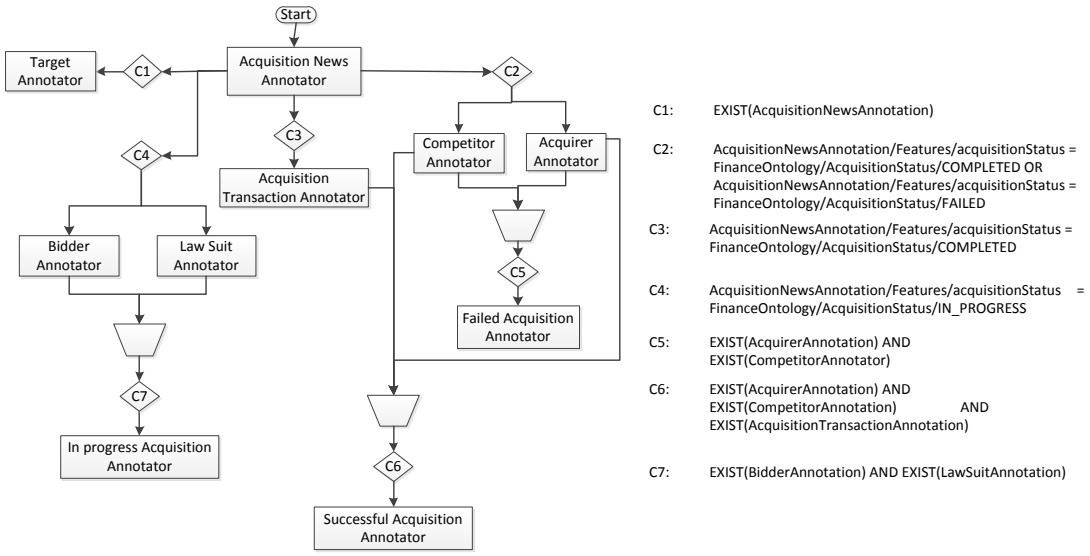

Fig. 2. The auto-generated annotators' execution workflow.

5. Conclusion

In this paper, we presented a prototype called Adaptive Rules-Driven Architecture for Knowledge Extraction (ARDAKE). Our development environment integrates a Business Rules Engine (BRE) or Semantic Rules Engine with the Unstructured Information Management Architecture (UIMA). It provides domain experts with a rules-driven framework to develop context-aware, adaptive knowledge extraction systems. We surveyed existing annotation platforms, in particular LanguageWare and TextMarker, two UIMA-based tools that use matching rules to create new annotations. We will be comparing ARDAKE with other UIMA-based annotation tools by asking a number of business users and developers to build the same aggregate annotators twice; once using ARDAKE and once using a tool of their choice starting with ARDAKE in some cases and with their preferred tool in other cases. Information such as the creation time, annotators' quality (complexity, performance, accuracy, etc…), reusability, and more will be collected to compare the different tools.

We will also add a new functionality to ARDAKE leveraging our precondition and post-processing rules, in combination with the UIMA annotators' input and output, so as to generate complete functional test sets using Extended Finite State Machine (EFSM) specifications and the Stream-X machine testing methodology as explained in (Ramollari, Kourtesis et al. 2009). A semantic reasoning engine (e.g., Pellet) will also be used to ensure rules coherence. Workflows will be dynamically generated in conformance with this quality assurance.

References

Goble, C. and D. De Roure (2009). "The impact of workflow tools on data-centric research." Data Intensive Computing: The Fourth Paradigm of Scientific Discovery: 137-145.

IBM (2011). "LanguageWare Resource Workbench 7.2 - Create Parsing Rules." IBM Corporation.

Kano, Y., P. Dobson, et al. (2010). "Text mining meets workflow: Linking U-compare with Taverna." Bioinformatics **26**(19): 2486-2487.

Kluegl, P., M. Atzmueller, et al. (2009). "TextMarker: A tool for rule-based information extraction." Proceedings of the Biennial GSCL Conference 2009, 2nd UIMA@GSCL Workshop: 233-240.

Ramollari, E., D. Kourtesis, et al. (2009). Leveraging Semantic Web Service Descriptions for Validation by Automated Functional Testing. Lecture Notes in Computer Science. Berlin, Springer**:** 593-607.

Wimalasuriya, D. C. and D. Dou (2010). "Ontology-based information extraction: An introduction and a survey of current approaches." Journal of Information Science **36**(3): 306-323.