A Model-driven approach to NLP programming with UIMA

Alessandro Di Bari, Alessandro Faraotti,
Carmela Gambardella, and Guido Vetere

IBM Center for Advanced Studies of Trento
Piazza Manci, 1 Povo di Trento

Abstract. In Natural Language Processing, more complex business use cases and shorter delivery times drive a growing need of smoother, more flexible and faster implementations. This trend also requires integrating and orchestrating different functionalities delivered by services belonging to different technological platforms. All these needs imply raising the level of abstraction for NLP components development. In this paper we present a Model Driven Architecture approach suitable to develop an open and interoperable UIMA-based NLP stack. By decoupling UIMA NLP models from other solution specific platforms and services, we obtain major architectural improvements.

1 Introduction

As Natural Language Processing (NLP) approaches complex tasks such as Question Answering or Dialog Management, the capability for NLP tools to seamlessly interoperate with other software services, such as knowledge bases or rules engines, becomes crucial. Such level of integration may require linguistic models to be shared among a variety of different platforms, each of which comes with its own information representation language. Platforms like UIMA[1] or GATE[2] consist of middleware and tools for designing and pipelining NLP specific tasks, including support for modeling data structures for text annotation, such as lexical, morphological and syntactic features, which may be embedded in interprocess communication protocols. However, while perfectly suited for annotation purposes, NLP specific schema languages, such as the UIMA Type System, fall short on fulfilling solution-level modeling needs. Model-Driven software Architectures (MDA), on the other hand, are specifically aimed at tackling the complexity of modern software infrastructures, with emphasis on the integration and the orchestration of different technological platforms. The MDA approach is based on providing formal descriptions (models) of requirements, interactions, data structures, protocols, and many other aspects of the desired system, which are automatically turned into technical resources, such as schemes and software modules, by activating transformation rules.

[1] `http://uima.apache.org/`
[2] `http://gate.ac.uk/`

Based on this consideration, we adopted an MDA approach to develop a "Watson ready"[3], UIMA-based NLP stack for Italian, as part of the activity of the newborn IBM Language & Knowledge Center for Advanced Studies of Trento[4]. We wanted our stack to be as open and interoperable as possible, to help users leveraging the availability of NLP resources and tools in the Open Source / Open Data space. In addition, our stack aims at being independent from language specific issues and domains, to facilitate its reuse across projects and within our (multinational) Company. The basic idea was to design a highly modularized general model including all the required structures, and to obtain technical platform-specific resources from a suitable set of model-to-model transformations. Also, we embraced the idea of abstracting semantic information away from the UIMA Type System, as in [5] and in [7], and evaluated the benefit of representing such kind of information by specific means. In sum, we looked at UIMA as a well-suited platform for linguistic analysis, which allows the integration of analytic components into managed workflow pipelines, but regarded at the UIMA Type System as a schema specification for that platform, rather than as a general modeling language for any NLP-based solution.

Here we present an overview of the basic ideas behind our approach, introduce our project, and discuss future directions. At the present stage of development, we can share our vision on MDA positioning and motivation with respect to NLP development (section 3), and we can report our first implementation experiences (section 4). Finally, we outline some related topic and introduce future works.

2 Motivating Scenario

Natural Language based solutions may require the NLP stack to cooperate with other components in a complex system. Such cooperation typically involves data exchanges with reference to a shared information model. The picture 1 shows the integration of an NLP stack with a Knowledge Base (e.g. an Ontology-based Data Access System) and a Rule Engine.

An UIMA-based NLP pipeline produces an annotated text (step 1 in the picture 1) contained in an UIMA CAS (Common Annotation Structure). A wrapper of the UIMA Type System defines all the operations needed for a consumer (the Rule Engine in this case) in order to access the CAS and invoke the appropriate operations within the cooperating subsystem when needed (see 4.2). When developing and maintaining the solution, an Engineer builds a rule set (see step 3) in order to process linguistic structures and interact with a Knowledge Base (step 4), which, in turn, uses the annotated text to store assertions as the result of an Information Extraction process (step 5). In a separate flow, the Knowledge Base can be queried by a User through a Question Answering System based on a suitable query language (step 6). The integration of all components involved is guaranteed by a common abstract model (Platform Independent Model) that contains the overall conceptualization of the system. The transition from one

[3] `www.ibm.com/watson/`
[4] `www.ibm.com/ibm/cas/`

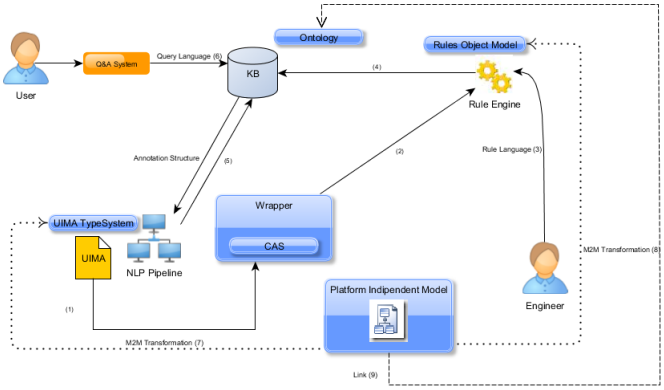

Fig. 1. Architectural sketch

platform specific data structure to another is handled by a set of Model-to-Model transformations (steps 7 and 8). The figure also shows the link to legacy (possibly huge) conceptual models, such as the KB ontology (step 9).

3 Model Driven Architecture for NLP

Model Driven Architecture (MDA) [6] is a development approach, strictly based on formal specifications of information structures and behaviors, and their semantics. MDA is managed by Object Management Group (OMG)[5] based on several modeling standard such as: Unified Modeling Language (UML)[6], Meta-Object Facility (MOF), XML Metadata Interchange (XMI) and others. MDA supports Model Driven Development/Engineering (MDD, MDE).

The key idea behind MDA is to provide a higher level of abstraction so that software can be fully designed independently from the underlying technological platform. More formally, MDA defines three macro "modeling" layers:

- **Computation Independent Model** (CIM)
- **Platform Independent Model** (PIM)
- **Platform Specific Model** (PSM)

The first one can be related to a Business Process Model and does not necessary imply the existence of a system that automates it. The PIM is a model that is independent from any technical platform; the third (PSM) layer is the actual implementation of the model with respect to a given technology and it is automatically derived from the PIM. Notice that the PIM allows a comprehensive representation of the structure and behavior of the system being developed.

[5] `http://omg.org/`
[6] `http://www.uml.org/`

The modeling language is typically UML or EMF[7], but it could actually be any other Domain Specic Language (DSL).

Developing powerful NLP tasks, such as Question Answering systems, requires combining a great variety of analytic components, which is what UIMA has been designed for. We consider UIMA the standard solution for document workflow analysis. Within this framework, MDD tools can be effectively used to better manage the UIMA Type System. In particular, we decided to look at it as a PSM dedicated to text annotation. The motivation for leveraging MDD (in the NLP field) can be summarized as follows:

- **Formalization**: MDA languages are well studied in logics and reasoning mechanisms can be developed upon. [1]
- **Expressiveness**: MOF meta-modeling allow great and well-founded expressiveness [4], including modeling behaviors.
- **Support**: The availability of tools, including diagramming and code generation, improves software life-cycle and team collaboration.

In particular, with respect to our architecture, we modeled UIMA annotations by defining *classes* rather than just (data) types, so that a consumer is able to invoke operations designed for those objects. Access to UIMA annotation is then achieved by means of automatically generated wrappers. Another motivation for a model driven approach was the need to represent complex linguistic data, and exploit existing tooling and resources for generating training data for a statistical parser.

In sum, we tried to exploit the maturity and flexibility of MDD tools while keeping up the power of UIMA as a framework for component integration, pipeline execution, and workflow management in general. As the PIM language, we chose EMF because it is already integrated with UIMA and provides powerful and mature model driven features. Once also the code is generated (by UIMA JCASgen), the type system correspond to an implementation of a (business) domain model, limited to the structural aspects (as opposed to behavioral aspects).

At PIM level, we also have to represent those properties that, once transformed against a target model, give specific characteristics on that model. For instance, in order to generate the UIMA Type System (PSM) starting from the PIM, we have to represent on the source model whether a class (that is a root in a hierarchy on the PIM model) will be generated as an UIMA annotation or not (UIMA TOP). Here we have taken two possible scenarios into account:

- Having an UML PIM, this specification is easily accomplished by using an UML *profile*[8]. Profiles define stereotypes that can be further structured with custom properties. This way, we have a generic "Unstructured Information" profile that at least, encompasses an `Annotation` stereotype; thus a class that is thought to become an annotation will be simply "marked" with this stereotype.

[7] `http://www.eclipse.org/modeling/emf/`
[8] `http://www.omg.org/spec/#M&M`

– Having an EMF PIM (such as our current implementation), we can represent the same thing as an EMF annotation. Therefore, (we apologize for the words conflict) we will have a class annotated as `Annotation`.

In any case, a class stereotyped as `Annotation` on the PIM will take the role of a generic annotation for document analysis, independently from the underlying framework.

The main benefit of our approach is the ability to represent NLP objects independently from any particular implementation: we are using different (generated) PSMs (that are better explained in section 4) all deriving from the starting (PIM) model, as shown in the picture 2.

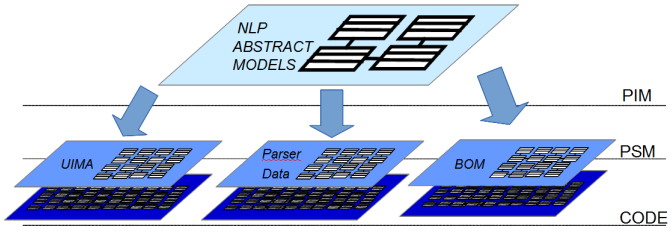

Fig. 2. Mdd for NLP: different abstraction layers

These benefits have certainly a price, that is essentially represented by the cost of developing the necessary transformations. However, following basic assumptions of the MDD approach, we estimate that those cost are well paying back, especially when heterogeneous components have to be integrated, development is managed iteratively, and models are subject to high volatility.

4 Model Driven Implementation Aspects

In order to better clear up how we are leveraging the Model Driven approach, we list here the artifacts (PSMs and code) we are generating through appropriate transformations that we have developed.
Starting from our "application" model:

– UIMA type system (we modified the existing transformation from EMF in order to avoid any further modification on the UIMA type system)
– EMF wrapper of UIMA type system
– this wrapper also acts as the input for creating the model for the Rule engine as explained below

Starting from (our) models of common standard data for parser training such as CONLL, PENN and others we generated all necessary (OpenNLP-specific) data for training the parser on:

– Tokenization
– Named Entities
– Part of Speech tagging
– Chunking
– Parsing

To represent the model (PIM), we use the Eclipse Modeling Framework[9] (EMF), which represents a *de facto* Java-based standard for meta-modeling. Informally, we may say EMF represents a subset of UML (the structural part) with very precise semantics for code generation. In the future, we could move this representation to a profiled UML, as mentioned above (see section 3). Furthermore, EMF offers very powerful generation features. Summarizing, in the current implementation we use EMF in two ways:

1. A language to represent the model
2. A PIM model to generate different target PSM

4.1 NLP Parser

The NLP Parser component is implemented using Apache OpenNLP[10] and UIMA[11]; it is based on a UIMA Type System built from the Syntax and the Abstract models using the UIMA transformation utility. The training corpora for the parser has to be provided in a specific format required by OpenNLP. Since the data that we had available for training were in standard formats such as PENN[12], CONLL[13] and others, some transformations were required. Ecore models have been created for the purpose of representing source formats. Furthermore, some simple JET[14] transformations has been developed in order to generate our corpora (in specific OpenNLP formats)

Compared to other solutions, this makes our infrastructure extremely flexible: should the parser be replaced or the data formats changed, the only operation we will have to make is to modify the JET template accordingly.

4.2 Type System EMF Wrapper

As anticipated in 2, in the higher layers of our architecture, we have a Rule Engine that acts as a reasoner on annotation objects coming from the UIMA pipeline. We wanted this layer to be able to call operations implemented on those objects (as explained in section 3) and those objects always implementing the exact interfaces of the (Ecore) PIM model. Given these requirements, we developed a transformation that generates a wrapper of the UIMA type system and that

[9] `http://www.eclipse.org/modeling/emf/`
[10] `http://opennlp.apache.org/`
[11] `http://uima.apache.org/`
[12] `http://www.cis.upenn.edu/~treebank/`
[13] `http://ilk.uvt.nl/conll/#dataformat`
[14] `http://www.eclipse.org/modeling/m2t/?project=jet`

fully reflects the starting PIM model, including operations. Once implemented, the code will be kept up also against future re-generations, thanks to merging capabilities of this transformation. Thus, as shown in figure 1, the Rule Engine "consumes" instances of this wrapper, and still can access the underlying UIMA annotation. We considered the possibility of directly adding these operations on classes generated by UIMA (via JCAS generation utility) but this would not be consistent with our model-driven approach since those operations would not be part of a general, system-wide model.

4.3 Rule Engine

As far as the Rule Engine is concerned, we chose IBM Operational Decision Manager (ODM)[15]. ODM rules have to be written against a specific model, called Business Object Model (BOM), that allows a user-friendly business rule editing; ODM provides tools to set up a natural language vocabulary: users can use it to write business rules in a pseudo-natural language. Once defined, the rules are executed on a BOM-related Java implementation named Execution Object Model (XOM). We obtained the BOM by reverse engineering the XOM, and the XOM directly from Java classes (implementing the type system wrapper) generated from our PIM (EMF) model. Therefore, the BOM model can be seen as just another manifestation of our PIM model.

4.4 Knowledge Base

Our architecture is backed by a Knowledge Base Management System which stores and reasons on information extracted from many sources. Leveraging on the Knowledge Model included in the PIM, we were able to integrate an external pre-existing system, named ONDA (Ontology Based Data Access) [3]. ONDA supports Ontology Based Data Access (OBDA) on OWL2-QL ([16]), by ensuring sound and complete conjunctive query answering with the same efficiency a scalability of a traditional database [2]. Because the ONDA underlying Knowledge Model was already designed with EMF, we simply adopted it in order to be included in the PIM. This way, reasoning and query answering services have been included in the PIM model as operations available to all other components (i.e. the Rule Engine).

5 Conclusion and future works

We have outlined here an innovative approach to NLP development, based on the idea of setting UIMA as the target platform in a Model-Driven development process. A major benefit of this approach consists in giving NLP models a greater value, especially in terms of generality, usability, and interoperability.

[15] `http://www-03.ibm.com/software/products/us/en/odm/`
[16] `http://www.w3.org/TR/owl2-profiles/`

While developing this idea, we understood that a suitable Model-Driven machinery for NLP should be supported by specific design patterns for concrete models. In particular, the model we have developed has been abstracted both from morphosyntactic specificity and from semantic aspects. The former (including part-of-speech classes, genders, numbers, verbal tenses, etc) may significantly vary among different languages; the latter (including concepts like persons, events, places, etc) are related to specific application domains. By decoupling these layers, we achieved a lightweight "generic" UIMA type system[7], we designed a powerful generic model for morphosyntactic features, and we managed ontological information with proper expressive means. Refining and extending this model is part of our future plans.

We implemented a first prototype of a Knowledge Base query system based on the Eclipse Modeling Framework (EMF). For the future, we are considering the possibility of representing the model in UML, in order to have a greater representational power (such as modeling sequence diagrams).

The work presented here is still at an early stage. More work is needed to complete the linguistic model, for instance in the area of argument structures, such as verbal frames. From an implementation standpoint, our priority is to consolidate, improve and extend the set of Model-to-Model transformations, and to further exploit MDD tools.

References

1. A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. A formal framework for reasoning on uml class diagrams. In *Proceedings of the 13th International Symposium on Foundations of Intelligent Systems*, ISMIS '02, pages 503–513, London, UK, UK, 2002. Springer-Verlag.
2. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Dl-lite: Tractable description logics for ontologies. In *AAAI*, volume 5, pages 602–607, 2005.
3. P. Cangialosi, C. Consoli, A. Faraotti, and G. Vetere. Accessing data through ontologies with onda. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '10, pages 13–26, Riverton, NJ, USA, 2010. IBM Corp.
4. Liliana Favre. A formal foundation for metamodeling. In F. Kordon and Y. Kermarrec, editors, *Reliable Software Technologies Ada-Europe 2009*, volume 5570 of *Lecture Notes in Computer Science*, pages 177–191. Springer Berlin Heidelberg, 2009.
5. D. Ferrucci, J W. Murdock, and C. Welty. Overview of component services for knowledge integration in uima (aka suki). Technical report, IBM Research Report RC24074, 2006.
6. J. Miller and J. Mukerji. Mda guide version 1.0.1. Technical report, Object Management Group (OMG), 2003.
7. K. Verspoor, W. Baumgartner Jr, C. Roeder, and L. Hunter. Abstracting the types away from a UIMA type system. *From Form to Meaning: Processing Texts Automatically. Tübingen:Narr*, pages 249–256, 2009.