Towards Model Driven Architectures for Human Language Technologies

Alessandro Di Bari
IBM Center for
Advanced Studies of Trento
Povo di Trento
Piazza Manci 12
alessandro_dibari@it.ibm.com

Kateryna Tymoshenko
Trento RISE
Povo di Trento
Via Sommarive 18
k.tymoshenko@trentorise.eu

Guido Vetere
IBM Center for
Advanced Studies of Trento
Povo di Trento
Piazza Manci 12
guido_vetere@it.ibm.com

Abstract

Developing multi-purpose Human Language Technologies (HLT) pipelines and integrating them into the large scale software environments is a complex software engineering task. One needs to orchestrate a variety of new and legacy Natural Language Processing components, language models, linguistic and encyclopedic knowledge resources. This requires working with a variety of different APIs, data formats and knowledge models. In this paper, we propose to employ the Model Driven Development (MDD) approach to software engineering, which provides rich structural and behavioral modeling capabilities and solid software support for model transformation and code generation. These benefits help to increase development productivity and quality of HLT assets. We show how MDD techniques and tools facilitate working with different data formats, adapting to new languages and domains, managing UIMA type systems, and accessing the external knowledge bases.

1 Introduction

Modern architectures of knowledge-based computing (cognitive computing) require HLT components to interact with increasingly many sources and services, such as Open Data and APIs, which may not be known before the system is designed. IBM's Watson[1], for instance, works on textual documents to provide question answering and other knowledge-based services by integrating lexical resources, ontologies, encyclopaedic data, and potentially any available information source. Also, they combine a variety of analytical procedures, which may use search, reasoning services, database queries, to provide answers based on many kinds of evidence (IJRD, 2012). Development platforms such as UIMA[2] or GATE[3] facilitate the development of HLT components to a great extent, by providing tools for annotating texts, based on vocabularies and ontologies, training and evaluating pipeline components, etc. However, in general, they focus on working with specific analytical structures (annotations), rather than integrating distributed services and heterogeneous resources. Such integration requires great flexibility in the way linguistic and conceptual data are encoded and exchanged, since each independent service or resource may adopt different representations for notions whose standardization is still in progress. Therefore, developing HLT systems and working with them in such environments requires modeling, representing, mapping, manipulating, and exchanging linguistic and conceptual data in a robust and flexible way. Supporting these tasks with mature methodologies, representational languages, and development environments appears to be of paramount importance.

Since the early 90s, Computer Sciences have envisioned methodologies, languages, practices, and tools for driving the development of software architectures by models (Model Driven Architecture, MDA)[4]. The MDA approach starts from providing formal descriptions (models) of requirements, interactions, data structures, protocols and many other aspects of the desired system. Then, models are turned

[1]http://www.ibm.com/innovation/us/watson/

[2]http://uima.apache.org/

[3]https://gate.ac.uk/

[4]http://www.omg.org/mda/

into technical resources (e.g. schemes and software modules) by means of programmable transformation procedures. Model-to-model transformations, both at schema and instance level, are also supported, based on model correspondence rules (mappings) that can be programmed in a declarative way.

As part of the development of UIMA-based NLP components, easily pluggable in services ecosystems, we are working on an open, flexible and interoperable pipeline, based on MDA. We want our platform to be language agnostic and domain independent, to facilitate its use across projects and geographies. To achieve this, we adopted the Eclipse Modeling Framework[5] as modeling and developing platform, and we generate UIMA resources (Type System) as a Platform Specific Model, by means of a specific transformation. We started by designing models of all the required components, and analyzed the way to improve usability and facilitate interoperability by providing appropriate abstraction and layering.

In the present paper we provide the motivation of our architectural choices, we illustrate the basic features, and discuss relevant issues. Also, we provide some example of how MDA facilitates the design and the implementation of open and easily adaptable HLT functionalities.

2 Model-driven Development and NLP

2.1 Model-driven Development

Generally speaking, we talk about a "modeling" language when it is possible to visually represent (or model) objects under construction (such as services and objects) from both a structural and behavioral point of view. The most popular (software) modeling language is the Unified Modeling Language (UML)(Rumbaugh et al., 2005). One of the strengths of such language is that it allows clear and transparent communication among different stakeholders and roles such as developers, architects, analyst, project managers and testers.

Model Driven Development (MDD) is a software development approach aimed at improving quality and productivity by raising the level of abstraction of services and related objects under development. Given an application domain, we usually have a "business" model that allows for fast and highly expressive representation of specific domain objects. Based on business models, the MDD tooling provides facilities to generate a variety of platform specific "executable models"[6]

2.2 Model-driven Architecture

Model Driven Architecture (MDA)(Miller and Mukerji, 2003) is a development approach, strictly based on formal specifications of information structures and behaviors, and their semantics. MDA is promoted by the Object Management Group (OMG)[7] based on several modeling standards such as: Unified Modeling Language (UML)[8], Meta-Object Facility (MOF[9]), XML Metadata Interchange (XMI) and others. The aim is to provide complex software environments with a higher level of abstraction, so that the application (or service) can be completely designed independently from the underlying technological platform. To this end, MDA defines three macro modeling layers:

- **Computation Independent Model** (CIM) abstract from any possible (software) automation; usually business processes are represented at this layer
- **Platform Independent Model** (PIM) represents any software automation (that supports the CIM) but this representation is really independent from any technological constraint such as the running platform or related middleware, or any third party software. Usually, the skeleton or structure of such a model is automatically generated from the CIM but it is expected to be further expanded for

[5]http://www.eclipse.org/modeling/emf/

[6]For a typical MDD approach, there are two different possible implementation strategies: the first one is to customize generic modeling languages (such as UML) by providing specific profiles (representing the business domain) for the language (UML is very generic and it offers several extensibility mechanisms such as profiles); the second one is a stronger approach that leads to what we call a DSL (Domain Specific Language), a fully specified, vertical language for a particular domain. Such a language is usually built in order to allow business experts to directly work on it, without the need of technological skills.

[7]`http://omg.org/`

[8]`http://www.uml.org/`

[9]`http://www.omg.org/mof/`

a fully specification. PIM can be expressed in UML (Rumbaugh et al., 2005) or EMF[10] but also in any peculiar DSL (domain specific language).

- **Platform Specific Model** (PSM) can be completely generated from the PIM. Among other things, this requires PIM to be able to represent every aspect of the solution under development: both structural and behavioral parts have to be fully specified at this level.

2.3 Eclipse Modeling Framework (EMF)

We chose to adopt EMF as the underlying modeling framework and tooling for our model-driven approach. EMF is an Eclipse[11]-based modeling framework and code generation facility. Ecore is the core meta-model for EMF. Ecore represents the reference implementation of OMG's EMOF (Essential Meta-Object Facility). EMF is at the heart of several important tools and applications and it is often used to represent meta-models and their instances (models) management. Just as an example, UML open source implementation defines its meta-model in terms of EMF-Ecore. Applications or tools that use Ecore to represent their meta model can leverage the EMF tooling to accomplish several goals such as the code generation for model management, diagramming and editing of models, transformations visual development and so on.

2.4 Model-driven NLP

We considered the main features of the Model Driven approach as a powerful way to handle the complexity of modern HLT use cases (Di Bari et al., 2013). In adopting this approach, we chose UIMA[12] as a standardized and well supported tool to deploy our Platform Specific models.

With respect to the basic features of the UIMA platform, we wanted to enhance:

- The representation, visualization and management of complex linguistic and conceptual models
- The use of many kinds of data specifications
- The interaction with many kinds of platforms and services

Therefore, we decided to design a set of models in EMF, considering this as our PIM layer. From these models, we can generate PSM components to support a variety of tasks, including:

- A UIMA Type System for implementing NLP pipelines
- A set of data format transformations for working with linguistic corpora
- A set of basic interfaces to access linguistic and knowledge services
- A Business Object Model (BOM) to work with rule engines (business rules management systems)

3 Modeling NLP with EMF and UIMA

3.1 Working with data formats

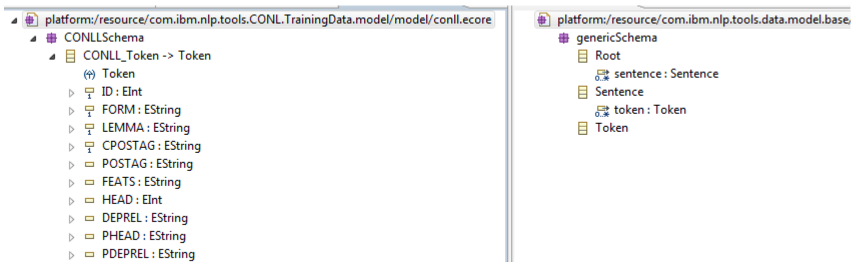
In order to train our statistical parser based on OpenNLP[13] and UIMA, we had to adapt different corpora formats, (such as PENN[14] and CONLL[15]), because OpenNLP requests them for different analysis classes (such as named entity, part-of-speech, chunking, etc.). In fact, we had to transform available corpora from standard formats to OpenNLP specific ones. We represented standard formats with EMF (an Ecore model for each one) and we created specific transformations using Java Emitter Templates (JET)[16], a

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

framework for fast code generation based on EMF. This solution gives us a lot of flexibility: if the parser or corpora formats change, we just have to adapt EMF models and/or JET templates consistently. For a better understanding, we show here an example of the JET template used for transforming from CONLL to OpenNLP POSTag format:

```
<c:setVariable select="/contents" var="root"/>
<c:iterate select="$root/sentence" var="sentence">
    <c:iterate select="$sentence/token" var="token">
        <c:get select="$token/@FORM"/>_<c:get select="$token/@CPOSTAG"/>
    </c:iterate>
</c:iterate>
```

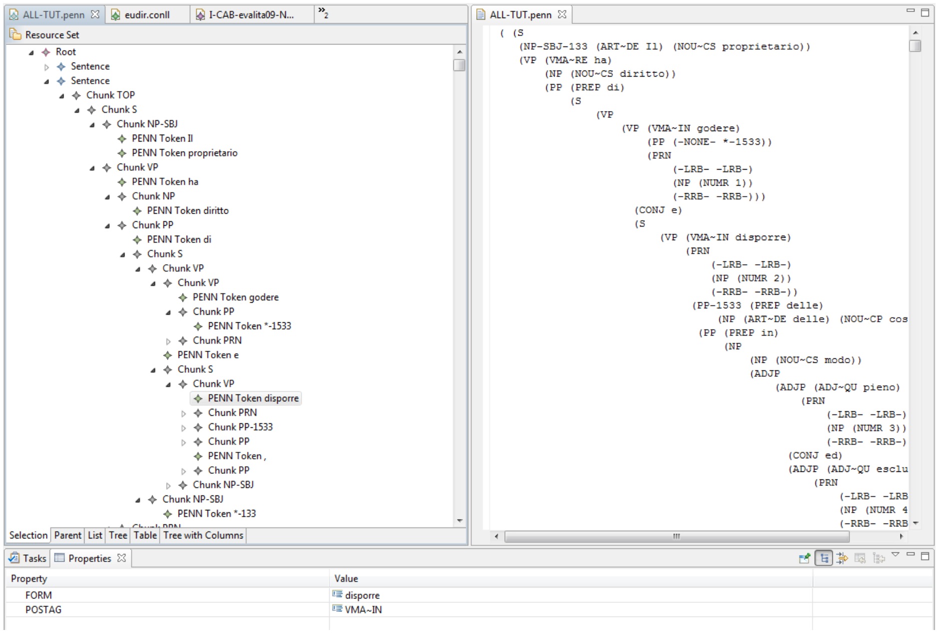
Having the simple CONLL Ecore model (in Figure 1) available, this template is the only artifact realizing the needed transformation. This template simply iterates over all sentences of a document (`root`), then over all tokens of a `sentence` and print out the `form` and its `postag` in the requested format. Other format conversions are done with the same technique. Notice that this tool allows to write (even complex) transformation without requiring programming skills, thus ensuring a greater maintainability and lowering the overall cost of the project.



```
platform:/resource/com.ibm.nlp.tools.CONL.TrainingData.model/model/conll.ecore
  CONLLSchema
    CONLL_Token -> Token
      Token
      ID : EInt
      FORM : EString
      LEMMA : EString
      CPOSTAG : EString
      POSTAG : EString
      FEATS : EString
      HEAD : EInt
      DEPREL : EString
      PHEAD : EString
      PDEPREL : EString
```

```
platform:/resource/com.ibm.nlp.tools.data.model.base/
  genericSchema
    Root
      sentence : Sentence
    Sentence
      token : Token
    Token
```

Figure 1: A simple EMF model representing CONLL format



```
Root
  Sentence
  Sentence
    Chunk TOP
      Chunk S
        Chunk NP-SBJ
          PENN Token Il
          PENN Token proprietario
        Chunk VP
          PENN Token ha
          Chunk NP
            PENN Token diritto
          Chunk PP
            PENN Token di
            Chunk S
              Chunk VP
                Chunk VP
                  PENN Token godere
                  Chunk PP
                    PENN Token *-1533
                  Chunk PRN
                PENN Token e
                Chunk S
                  Chunk VP
                    PENN Token disporre
                    Chunk PRN
                    Chunk PP-1533
                    Chunk PP
                    PENN Token ,
                    Chunk PP
                  Chunk NP-SBJ
              Chunk NP-SBJ
                PENN Token *-133
```

Property	Value
FORM	disporre
POSTAG	VMA~IN

```
( (S
  (NP-SBJ-133 (ART~DE Il) (NOU~CS proprietario))
  (VP (VMA~RE ha)
      (NP (NOU~CS diritto))
      (PP (PREP di)
          (S
              (VP
                  (VP (VMA~IN godere)
                      (PP (-NONE- *-1533))
                      (PRN
                          (-LRB- -LRB-)
                          (NP (NUMR 1))
                          (-RRB- -RRB-)))
                     (CONJ e)
                     (S
                         (VP (VMA~IN disporre)
                             (PRN
                                 (-LRB- -LRB-)
                                 (NP (NUMR 2))
                                 (-RRB- -RRB-))
                            (PP-1533 (PREP delle)
                                  (NP (ART~DE delle) (NOU~CP cos
                             (PP (PREP in)
                                 (NP
                                     (NP (NOU~CS modo))
                                     (ADJP
                                         (ADJP (ADJ~QU pieno)
                                             (PRN
                                                 (-LRB- -LRB-)
                                                 (NP (NUMR 3))
                                                 (-RRB- -RRB-)
                                           (CONJ ed)
                                           (ADJP (ADJ~QU esclu
                                               (PRN
                                                   (-LRB- -LRB
                                                   (NP (NUMR 4
                                                   (-RRB- -RRB
```

Figure 2: A sentence represented in PENN format (EMF editor on the left)

Further, as we can notice from the Figure 2, EMF provides very powerful MDD capabilities also from the modeling instantiation and editing point of view. Notice that these are two view of the very same resource (a PENN file). This is done by “teaching” the EMF framework how to parse the underlying persisted data, by providing an implementation (that can be done manually or using some parser generation tool in its turn) through a specific EMF extension point. From that point on, the framework will always be able to open and manage PENN resources as instances (with all semantic constraints and validation available) of the PENN Ecore model. Furthermore, the editor and all the specific model management tooling are automatically generated by the EMF platform. This way, all instance management (and editing) tasks are transferred to the EMF framework, which reduces costs and helps developers focusing on NLP tasks.

3.2 Managing the UIMA Type System

In order to manage a (complex enough) UIMA Type System, we fully leveraged the EMF/UIMA interoperability provided by the UIMA framework. UIMA already provides simple transformations from EMF to a UIMA type system and viceversa; furthermore, UIMA provides the XMI serialization so that an instance of a type system can be read as an instance of the corresponding EMF model. However, we had to modify the transformation from EMF to UIMA (type system) in order to reach our "model driven" goals and also to handle several dependent, but separated EMF modules. Specifically, our model is organized around the following modules (packages):

- **Text**. A model of linguistic occurrences in their context, which are given to the parser (token, sentences, named entities, parse) and get annotated by the underlying document analysis framework (e.g. UIMA).
- **Linguistics** A model that abstracts the linguistic apparatus, including categories, morphological and syntactic features, and relationships. This is the key for the multilinguality we walk through in the next section. We can see a fragment of this model in Figure 3
- **Ontology** An abstract representation of any entity and concept, along with a uniform interface to a variety of existing knowledge bases
- **Mapping** A model that allows bridging any tag-set, also explained in next section.

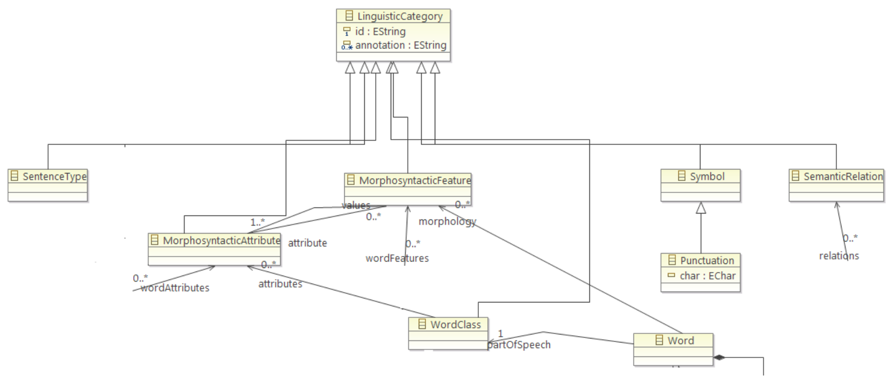


Figure 3: A fragment of the Linguistic model

3.3 Adapting to new languages and domains

Given this basis, we can now illustrate how we are able to incorporate new linguistic resources (such as vocabularies, ontologies, training corpora with different formats, tag-sets, etc) obtaining UIMA-based pipeline components. Given an (untrained) statistical library for parsing (such as OpenNLP or others) and a new natural language to represent, we do the following:

1. Analyze the format requested for the training corpora by the NLP parsing library versus the available training data for the new language. Most likely the training data will be in some standard format (e.g. CONLL, PENN, IOB) that is already represented by a suitable Ecore (EMF) model. If not yet present, we have to write a transformation (usually a simple JET template) from the standard to the specific one requested by the parser.

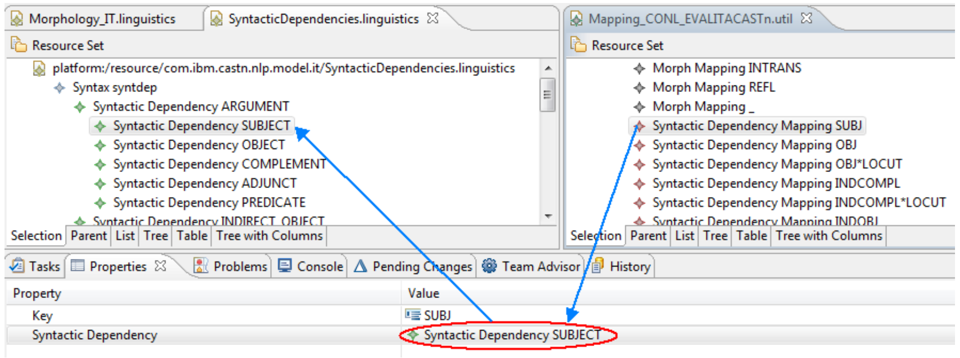
2. Train the parser for the specific language and tag-set.

3. Given the linguistic knowledge for the new language, and the task at hand, we adopt/modify/extend a suitable `Linguistics` model. If needed, we generate the UIMA Type System portion corresponding to the requested features.[17]

4. In case of a new business domain also, we adopt/modify/extend the ontology. If needed, we wrap business knowledge bases where the ontology is instantiated.[18]

5. Given the tag-set in use, we represent the mapping with the linguistic model, by instantiating a suitable `Mapping` model. At run-time, the (UIMA-based) parser asks (the mapping manager) to get a Linguistic Category (see Figure 4) of a given tag (the `key` feature in the `Mapping` model).

Given the workflow above, we can figure out how the `Linguistics` and `Mapping` model are playing a key role for achieving a multi-language solution. Just as an example, for the Italian language, the `Linguistics` model defines 55 Word Classes (part of speech), 6 morpho-syntactic attributes composed by 22 morpho-syntactic features and 27 syntactic dependencies. The mapping model for the EVALITA[19]tagset, contains 115 mapping elements.



```
Resource Set
  platform:/resource/com.ibm.castn.nlp.model.it/SyntacticDependencies.linguistics
    Syntax syntdep
      Syntactic Dependency ARGUMENT
        Syntactic Dependency SUBJECT
        Syntactic Dependency OBJECT
        Syntactic Dependency COMPLEMENT
        Syntactic Dependency ADJUNCT
        Syntactic Dependency PREDICATE
      Syntactic Dependency INDIRECT_OBJECT
```

```
Resource Set
    Morph Mapping INTRANS
    Morph Mapping REFL
    Morph Mapping _
    Syntactic Dependency Mapping SUBJ
    Syntactic Dependency Mapping OBJ
    Syntactic Dependency Mapping OBJ*LOCUT
    Syntactic Dependency Mapping INDCOMPL
    Syntactic Dependency Mapping INDCOMPL*LOCUT
    Syntactic Dependency Mapping INDOBJ
```

Property	Value
Key	SUBJ
Syntactic Dependency	Syntactic Dependency SUBJECT

Figure 4: A fragment of the mapping between tag-set and the Linguistic model

We can now prove how the specific language knowledge has been encapsulated in the suitable model so that for example, the code that manages the parser results and creates UIMA annotations does not change for a new language. Correspondingly, no programming skills are needed to represent this linguistic knowledge.

3.4 Benefits for NLP development

To show the benefits of our approach, we summarize here the following remarks:

- Whereas we consider UIMA the reference framework for document management and annotation, we also leveraged the features of a mature and stable modeling framework such as EMF. For instance, we could exploit diagramming, model and instance management and code generation. In sum, we saved significant development time by means of higher level of abstraction that allowed us to easily create transformations, create mapping models, smoothly use different format for the same data and so on.

- Having an additional level of abstraction can lead in thinking there is an additional cost; however, as stated above, this is not the case for transforming from EMF to UIMA. The only, real additional cost is represented by the effort of developing transformations. Nevertheless, this is quickly absorbed as soon as the transformation is re-used. In our case, just for data conversions, we re-used the same

[17]Steps 2, 3, 4 are done through our Eclipse-based tooling.
[18]The benefit of abstracting semantic information from the Type System has been illustrated in (Verspoor et al., 2009)
[19]`http://www.evalita.it/`

EMF models (and their underlying data parsing capabilities we implemented) several times. Each time we wanted to change a parser library, or new corpora were available, we reused the same EMF models and techniques.

- The abstraction introduced by the `Linguistic` model, allowed us to create a language-independent parsing software layer. Furthermore, the same model was leveraged in order to allow interoperability between different parsers that were using different tagsets.

4 Using an External Knowledge Base

State-of-the-art NLP algorithms frequently employ semantic knowledge about entities or concepts, mentioned in the text being processed. This kind of knowledge is typically extracted from the external lexicons and knowledge bases (KB)[20], e.g. WordNet (Fellbaum, 1998) or Wikipedia[21]. Therefore, in our NLP stack, we have to model extraction, usage and representation of semantic knowledge from the external KBs each of which might have different structure and access protocols. Moreover, we may need to plug new KBs into our software environment, with the least effort. Finally, in complex services ecosystems, we might also need to use the KB outside of the NLP system, e.g. in a remote reasoning system, and we should be able to reuse the same model for these purposes.

MDD allows us to define the a single ontology/knowledge base (KB) abstraction, i.e. KB PIM, based on the high-level requirements of our software system, regardless of the actual implementation details of the KBs to be used. In contrast to UIMA type systems, which are limited to type hierarchies and type feature structures with the respective getters and setters, MDD allows to specify also functionalities, i.e. methods, such as querying and updating the KB. Figure 5 demonstrates a diagram of a KB abstraction that we employ in the `Ontology` module. In this abstraction *KnowledgeBase* is a collection of *Individuals* and *Concepts*, *Relationships* and *Roles*, which all are subclasses of the *Entity* abstraction. The figure contains also the visualizations of some components from the `Text` module, which show how we model annotating text with references to the KB elements. For this purpose we have a special kind of *TextUnit* called *EntityToken* used to encode the fact that a certain span in a text denotes a specific *Entity*. Note that this is a high-level abstract schema without any platform-related details. This KB PIM may be invoked in the various parts of the full system PIM model, not necessarily within the NLP stack. We can use this schema to generate different PSMs depending on our needs.

One of the core benefits of MDD are the transformations. After we have defined our high-level KB PIM conceptual model we may define a set of transformations which will convert it to the PSM models, which contain the implementation details of the conceptual model within a specific platform. For example, it can be Jena TDB[22], a framework for storing and managing structured knowledge models, or a SQL relational database. PIM-to-PSM transformations can be defined programmatically, by means of special tools such as IBM Rational Software Architect[23], or by means of a specific modeling language, e.g. *ATL Transformation Language* (Jouault et al., 2006). Finally, we can use a number of tools for code generation from the PSM model, thus facilitating and speeding up the software development process.

Depending on the task, our `Ontology` PIM may be instantiated both as an UIMA PSM or as a PSM for another platform. More specifically, we have the following scenarios for the KB usage.

Within NLP stack. This may be required, for example, if some annotators in the NLP UIMA pipeline, such as a relation extractor or a coreference resolver, require knowledge about types of the individuals (entities) mentioned in a text. In such case, in order to annotate the text with information about individuals and their classes from an external KB we can transform the PIM model to a UIMA Type System (TS). We define a transformation which converts *TextUnit* to a subclass of *UIMA Annotation*[24], and the *Entity* element in `Ontology` to a direct subclass of *UIMA TOP*[25]. Therefore, for example, within our model

[20]Here we use the term “knowledge base” to refer to any external resource containing structured semantic knowledge
[21]`http://en.wikipedia.org/`
[22]`http://jena.apache.org/documentation/tdb/index.html`
[23]`http://www-03.ibm.com/software/products/en/ratisoftarch`
[24]`https://uima.apache.org/d/uimaj-2.6.0/apidocs/org/apache/uima/jcas/tcas/Annotation.html`
[25]`https://uima.apache.org/d/uimaj-2.6.0/apidocs/org/apache/uima/jcas/cas/TOP.html`

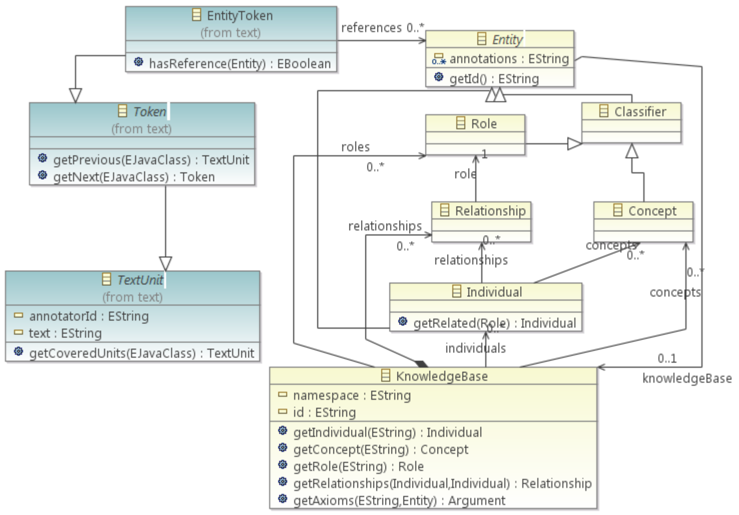


Figure 5: Fragments of the Ontology and Text models

illustrated in Figure 5, *Individual* abstraction, which is a subclass of *Entity*, becomes a subclass of *UIMA TOP*. The *EntityToken* annotations from `Text` are converted to a subclass of *UIMA Annotation* and are used to mark the spans of the *Individual*'s mentions in a text. In the original PIM model *EntityTokens* have property *references* which is a collection of pointers to the respective entities. In UIMA PSM this property is converted to a UIMA TS feature.

KnowledgeBase is an abstraction for an external KB resource to be plugged into the UIMA NLP stack. UIMA contains a mechanism for plugging in external resources[26] such as KBs. Each resource is defined by its name, location and a name of the class which implements handling this resource. MDA and PIM-to-PSM transformations can simplify modeling the resource implementations for different platforms, and EMF tools can further help with automatic code generation. We can define a transformation which would convert platform independent *KnowledgeBase* model elements to a platform-specific models, for instance a class diagram for implementing a *KnowledgeBase* within Jena TDB platform. Then we can use code generation tools for further facilitation of software development.

Within a reasoning component of a QA system. We may need to use both the output of the NLP stack and information stored in a KB within some reasoning platform. For instance, this could be needed within a Question Answering system which provides an answer to the input question based on both text evidence coming from the UIMA pipeline, e.g. syntactic parse information, and a KB. In this case, the PIM representations of linguistic annotations (`Text` and `Linguistics` packages) and KB knowledge (`Ontology` package) may be instantiated as PSMs relative to the specific reasoning (rule- or statistic-based) framework. UIMA annotations previously obtained within the UIMA can be converted to the format required by the reasoning framework by means of model-to-model transformations.

5 Related Work

In the recent years, a number of approaches to modeling annotations and language in NLP software systems and increasing interoperability of distinct NLP components have been proposed (Hellmann et al., 2013; Ide and Romary, 2006; McCrae et al., 2011).

The most widely-accepted solutions for assembling NLP pipelines are UIMA and the GATE (Cunningham et al., 2011) frameworks. They both use annotation models based on referential and feature structures and allow to define custom language models called *UIMA type system (TS)* or *GATE annotation schema* in an XML descriptor file. There are ongoing efforts to develop all-purpose UIMA-based

[26]`http://uima.apache.org/downloads/releaseDocs/2.1.0-incubating/docs/html/tutorials_and_users_guides/tutorials_and_users_guides.html#ugr.tug.aae.accessing_external_resource_files`

NLP toolkits , such as DKPro[27] or ClearTK (Ogren et al., 2009), with TSs describing a variety of linguistic phenomena. UIMA is accompanied with a UIMAfit toolkit (Ogren and Bethard, 2009), which contains a set of utilities that facilitate construction and testing of UIMA pipelines. The two frameworks are compatible, and GATE provides means to integrate GATE with UIMA and vice versa by using XML mapping files to reconcile the annotation schemes and software wrappers to integrate the components[28]. From the MDA perspective, UIMA and GATE type/annotation systems are platform specific models. Defining a language model as a platform independent EMF model results in greater expressivity. For example, in UIMA TS one can model only type class hierarchy and type features, while in EMF model we can also encode the types behavior, i.e. their methods. Moreover, MDA allows to model the usage of the annotation produced by an NLP pipeline within a larger system. For instance, this could be a question answering system which uses both information extracted from text (e.g. question parse) and information extracted from a knowledge base (e.g. query results) and provides tools to facilitate the code generation.

Certain effort has been made on reconciling the different annotation formats. For example, Ide et al. (2003) and Ide and Suderman (2007) proposed Linguistic Annotation Framework (LAF), a graph-based annotation model, and its extension, Graph Annotation Format (GrAF), an XML serialization of LAF, for representing various corpora annotations. It can be used as a pivot format to run the transformations between different annotation models that adhere to the *abstract data model*. The latter contains *referential structures* for associating annotations with the original data and uses feature structure graphs to describe the annotations. Ide and Suderman (2009) show how one may perform GrAF-UIMA-GrAF and GrAF-GATE-GrAF transformations and provide the corresponding software. However, in GrAF representation feature values are strings, and additionally, it is not possible to derive information about annotations types hierarchy from the GrAF file only, therefore, the resulting UIMA TS would be shallow and with all feature values types being string, unless additional information is provided (Ide and Suderman, 2009). At the same time, MDD tools like EMF provide a both a modeling language with high expressiveness and solid support to any transformation. We show how MDD facilitates conversion between different formats in Section 3.1. Additionaly, MDD tools like EMF have a solid sofware support for complex model visualization, this helps to facilitate understanding and managing large models.

After the emergence of the Semantic Web, RDF/OWL formalisms have been used to describe language and annotation models (Hellmann et al., 2013; McCrae et al., 2011; Liu et al., 2012). For instance, NLP Interchange Format, NIF, (Hellmann et al., 2013) is intended to allow various NLP tools to interact on web in a decentralized manner. Its annotation model, *NIF Core Ontology*, encoded in OWL, provides means to describe strings, documents, spans, and their relations. Choice of a language model depends on the developers of the NLP tools, however, they are recommended to reuse existing ontologies, e.g. Ontologies of Linguistic Annotations (OLiA) (Chiarcos, 2012). Another effort, Lemon (McCrae et al., 2011), is a common RDF meta-model for describing lexicons for ontologies and linking them with ontologies. Its core element is a lexical entry which has a number of properties, e.g. semantic roles or morpho-syntactic properties. There has also been research on converting UIMA type systems to the OWL format (Liu et al., 2012). OWL is highly expressive, and a number of tools exists for reasoning upon OWL/RDF models or visualizing them, e.g. Protégé[29], or for aligning them (Volz et al., 2009). Expressiveness of UML, which is typically used to encode the PIM models in MDD, is comparable to that of the ontology description languages (Guizzardi et al., 2004), and it may be reasoned upon (Calvanese et al., 2005). However, differently from the OWL models, there is a number of software solutions which are able to generate code on top of UML representations. Therefore, when using MDD we benefit both from the high expressiveness of the modeling language and the solid software engineering support provided by the MDD tools.

[27]http://www.ukp.tu-darmstadt.de/software/dkpro-core/

[28]http://gate.ac.uk/sale/tao/splitch21.html

[29]http://protege.stanford.edu/

6 Conclusion

Model Driven Development and Architecture is a successful paradigm for tackling the complexity of modern software infrastructures, well supported by tools and standards. We have discussed how the basic principles behind MDD/A are relevant for Human Language Technologies, when approaching the coming era of high-level cognitive functionalities delivered by interconnected software services, and grounded on open data. Model-to-model transformations, supported by specific tools, offer the possibility to rapidly integrate different platforms, and to work with many kinds of data representations. To get the best of this approach, it is important to carefully analyze the layering and interconnections of different models, and to provide them with a suitable design. In our work, we are learning the benefit of modeling aspects such as morpho-syntax and semantics separately, to foster adaptability across languages and domains. A complete and principled analysis of such general design is yet to come. Here we have presented some preliminary result of our experiences, and shared what we achieved so far.

Acknowledgements

This research is partially supported by the EU FP7 / Marie Curie Industry-Academia Partnerships and Pathways schema / PEOPLE Work Programme (Grant agreement no.: 286348, K-Drive project) and by the IBM Center for Advanced Studies of Trento, Italy.

References

D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. 2005. DL-Lite: Tractable description logics for ontologies. In *AAAI*, pages 602–607.

C. Chiarcos. 2012. Ontologies of linguistic annotation: Survey and perspectives. In *LREC*, pages 303–310.

H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, N. Aswani, I. Roberts, G. Gorrell, A. Funk, A. Roberts, D. Damljanovic, T. Heitz, M. A. Greenwood, H. Saggion, J. Petrak, Y. Li, and W. Peters. 2011. *Text Processing with GATE (Version 6)*. University of Sheffield Department of Computer Science.

A. Di Bari, A. Faraotti, C. Gambardella, and G. Vetere. 2013. A Model-driven approach to NLP programming with UIMA. In *3rd Workshop on Unstructured Information Management Architecture*, pages 2–9.

C. Fellbaum. 1998. *WordNet: An electronic lexical database*. The MIT press.

G. Guizzardi, G. Wagner, and H. Herre. 2004. On the foundations of uml as an ontology representation language. In E. Motta, N. Shadbolt, A. Stutt, and N. Gibbins, editors, *EKAW*, volume 3257 of *Lecture Notes in Computer Science*, pages 47–62. Springer.

S. Hellmann, J. Lehmann, S. Auer, and M. Brümmer. 2013. Integrating NLP using Linked Data. In *ISWC*.

N. Ide and L. Romary. 2006. Representing linguistic corpora and their annotations. In *LREC*.

N. Ide and K. Suderman. 2007. GrAF: A graph-based format for linguistic annotations. In *Proceedings of the Linguistic Annotation Workshop*, pages 1–8. Association for Computational Linguistics.

N. Ide and K. Suderman. 2009. Bridging the gaps: interoperability for GrAF, GATE, and UIMA. In *Third Linguistic Annotation Workshop*, pages 27–34. Association for Computational Linguistics.

N. Ide, L. Romary, and E. de la Clergerie. 2003. International standard for a linguistic annotation framework. In *HLT-NAACL workshop on Software engineering and architecture of language technology systems*, pages 25–30. Association for Computational Linguistics.

IJRD. 2012. This is Watson [Special issue]. *IBM Journal of Research and Development, editor Clifford A. Pickover*, 56(3.4).

F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, and P. Valduriez. 2006. ATL: a QVT-like transformation language. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 719–720. ACM.

H. Liu, S. Wu, C. Tao, and C. Chute. 2012. Modeling UIMA type system using web ontology language: towards interoperability among UIMA-based NLP tools. In *2nd international workshop on Managing interoperability and compleXity in health systems*, pages 31–36. ACM.

J. McCrae, D. Spohr, and P. Cimiano. 2011. Linking lexical resources and ontologies on the semantic web with lemon. In *The Semantic Web: Research and Applications*, pages 245–259. Springer.

J. Miller and J. Mukerji. 2003. MDA Guide Version 1.0.1. Technical report, Object Management Group (OMG).

P. Ogren and S. Bethard. 2009. Building test suites for UIMA components. In *Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, pages 1–4. Association for Computational Linguistics, June.

P. V. Ogren, P.G. Wetzler, and S. J. Bethard. 2009. ClearTK: a framework for statistical natural language processing. In *Unstructured Information Management Architecture Workshop at the Conference of the German Society for Computational Linguistics and Language Technology*.

J. Rumbaugh, I. Jacobson, and G. Booch. 2005. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Boston, MA, 2 edition.

K. Verspoor, W. Baumgartner Jr, C. Roeder, and L. Hunter. 2009. Abstracting the types away from a UIMA type system. *From Form to Meaning: Processing Texts Automatically. Tübingen:Narr*, pages 249–256.

J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. 2009. Silk - A Link Discovery Framework for the Web of Data. In *LDOW*.