ITEA Office
High Tech Campus 69 - 3
5656 AG Eindhoven
The Netherlands

T + 31 88 003 6136
E info@itea3.org
W www.itea3.org

ITEA 3 is a EUREKA strategic ICT cluster programme

<D2.1.2 Documentation of the Corpora>

ModelWriter

Text & Model-Synchronized Document Engineering Platform

Work Package: WP2
Task: 2.1 – Data Collection

Edited by:

Claire Gardent <claire.gardent@loria.fr> (CNRS / LORIA)
Mariem Mahfoudh <mariem.mahfoudh@loria.fr> (CNRS / LORIA)
Samuel Cruz-Lara <samuel.cruz-lara@loria.fr> (University of Lorraine / LORIA)

Date: 2-Sept-2015
Version: 1.0.0

Document History

Version	Author(s)	Date	Remarks
0.1.0	Claire Gardent, Mariem Mahfoudh, Samuel Cruz-Lara	2-Sept-2015	Draft
1.0.0	<name>	<date>	Initial Release

Table of Contents

1. Introduction

Role of the deliverable

This deliverable documents the data used to train, develop and text the NLP components (Semantic Annotator, Semantic Parser, Natural Language Generator) of ModelWriter. It might be updated during the project in case additional data is worked with.

Structure of the document

This document is organized as follows:

- Section 1 introduces the document.
- Section 2 describes for each use case: the scope and motivation, the approach and the available resources (corpora).

Terms, abbreviations and definitions

Abbreviation	Definition
NLG	Natural Language Generation
NLP	Natural Language Processing
RDF	Resource Description Framework
RDFS	RDF Schema
UML	Unified Modelling Language
OWL	Web Ontology Language
IDE	Integrated Development Environment
EMF	Eclipse Modelling Framework
GUI	Graphical User Interface
JDT	Java Development Tooling
WP	Work Package
UC	Use Case

2. Introduction

The development and the evaluation of natural language processing systems required data: for training, for tuning and for testing. In the ModelWriter project, this includes textual data, knowledge data and ideally bi-texts i.e., aligned corpora of text and their corresponding knowledge representation.

Three data types are distinguished:

1. The texts that are technical documents describing the rules and the services of a company. They can be txt file, pdf file, java file, etc. and they can contain both text (words, sentences, …) and pictures.
2. The models that are formal and structured representation of the technical documents (texts). They can be UML diagram, conceptual model, etc.
3. The knowledge bases that are an explicit specification of a conceptualization of a domain. They are a formal representation of domain knowledge and they can be RDF or OWL ontologies. The knowledge bases are used to identify and check the consistency of the links between text and model. They are also used to annotate both the text and model.

Figure 1 represents the relations between the different types of data.

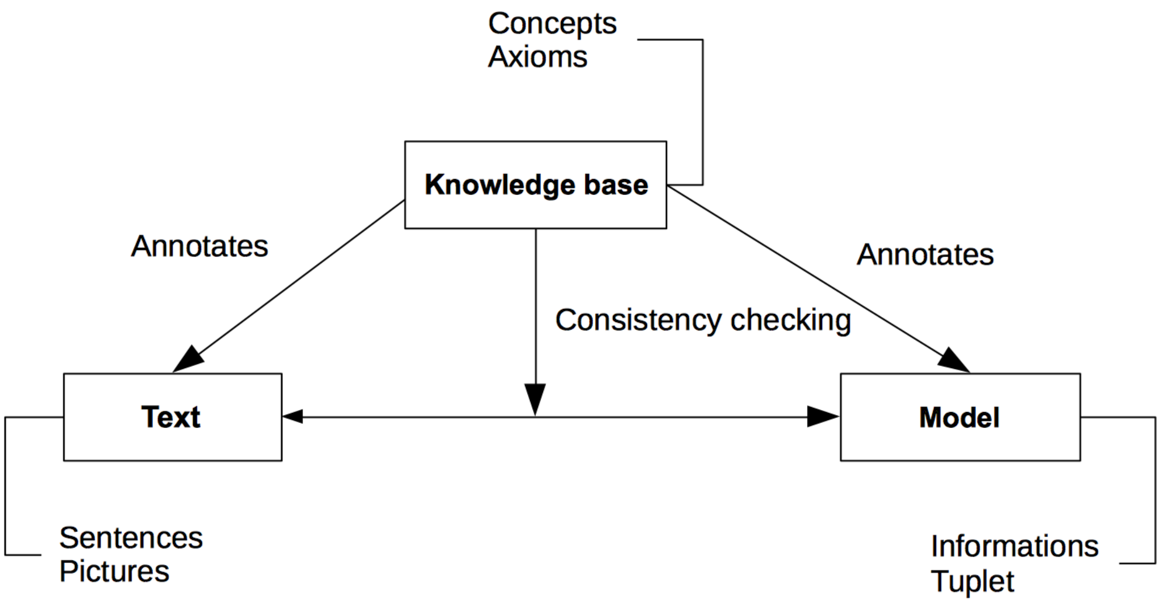


Figure 1: The relationship between the data constituting the corpora.

The rest of this document is organized as follows: section 2 represents the Airbus corpora. The section 3 describes the Obeo corpora. Finally, the section 4 shows the Turkish corpora.

Based on the use cases identified in WP1, we collected data to develop and evaluate three NLP tools necessary to achieve ModelWriter goals namely, a semantic annotator, a semantic parser and a natural language generator.

The semantic annotator is required to synchronised text and models. Its function is to annotate text with elements of the model whereby text elements may differ from model elements with respect to derivational (warn/Warning) or inflectional (pipe/Pipes) morphology, synonymy (pipe/tube) and/or syntax (procedure should be removed/procedure deletion).

This data will then be used to identify the linguistic requirements set by the use cases; to train and test the semantic processors (parser and generator); and to acquire the language models useful for disambiguation (parsing) and fluency ranking (generation).

A semantic parser converts text into model representations. It can be used to extend the model (by adding to the current model the model expression representing the meaning of the parsed text) or to synchronise complex natural language expression with one or more model elements.
Conversely, a natural language generator maps model representations to text. It can be used to update a text which is synchronised with a model whenever this model is modified/extended.

3. Airbus Data

Some This section describes the Airbus corpora which is composed of a set of texts and a knowledge base (the model).

The Airbus use cases UC-FR4 and UC-FR5 target the synchronisation of Airbus SIDP (e.g. System Installation Design Principles) documents with an RDFS model.

The overall driving need for these two use cases is to reduce the time and the burden for the designers to consult a large set of regulation documents in order to retrieve design rules. Due to reasons such as technology push, process changes, etc. an increasing number of different regulation documents are issued by different stakeholders. They contain a high number of informal rules and the designers have difficulties following the information cascade and retrieving or rebuilding the correct information. This situation results in time waste, suboptimal designs and higher risks of error. In ModelWriter, our ultimate goal is to remedy this shortcoming by providing a synchronization mechanism between these documents and a model encoding the rules contained in these documents This is an ambitious goal which in effect, requires building a semantic parser and a generator that can map arbitrary text into formal rules and vice versa. To achieve these goals, we gathered the following data.

Text

The text corpus is composed essentially of the System Installation Design Principles (SIDP). The SIDP documents are technical documents (doc files) that consists of various sets of regulations and directives about how to install a system or a set of systems in a functional area (e.g., electrical and optical system or Water Waste System). For each aircraft project, a set of such documents must be produced to ensure that the resulting system comply with the system requirements and take into consideration applicable regulations and procedures. Figure 2 presents an extract from a SIDP document. It shows a table that presents an example of component ("Bundle") with its definition and its picture. The definition specifies the rule that must be respected in the system installation of this component.

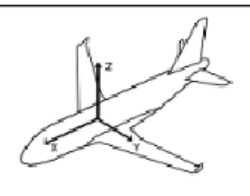
A/C axis	
Bundle	Group of wire/wires fastened together. Protection shall be considered as part of the bundle when they are used..  

Figure 2: An extract from a SDIP document.

We gathered two text corpora for developing and testing our NLP tools:

- SIDP document SIDP 92A001V. This SIDP document contains the system installation design principles applicable to the electrical and optical system installation. It provides an example of how design rules are formulated in SIDP documents and of how these documents are structured. The SIDP document SIDP 92A001 includes text and graphics and contains 6311 word forms. It is available in French-Consortium/airbus/text/SIDP92A001V.docx

- Semi-Structured SIDP rules. The Airbus System Installation team has built an SQL database of SIDP rules which encodes installation rules in a semi-structured format. In effect these rules provide a simplified, semi-normalised version of the rules contained in the SIDP documents thereby facilitating natural language processing (less diversity in the syntactic structures and lexicon, less ambiguity, rules formulated as one sentence rather than across several sentences, fewer anaphoric references etc). Table 1 shows an example of a tuple extracted from the database. It is a rule describing a segregation constraint holding between a pipe and an electrical route. This constraint is specified by Rule 1 and applies in Zone 1 of the functional area ATA38 (i.e., the water waste system).

We gathered these rules to develop a first version of the NLP tools (semantic annotator, semantic parser and text generator) that works on these semi-structured rules. Currently, the semi-structured rules available to the French consortium consists of 986 rules and 13178 word forms. These rules are available in two formats: an excel file whose columns are used to label each part of the rule (French-Consortium/airbus/text/rules.xls) and a text file where this labelling is ignored. (French-Consortium/airbus/text/rules.txt). The excel file is used to automatically construct an RDFS version of the rules while the text file is used to test the NLP tools.

ATA	Zone	Rule	Object	Auxiliary	Action Verb	Prep	Object 2
38	1	1	pipe	shall be	segregated	from	electrical route

Table 1: A rule from Airbus' model.

Model

Semantic Parsing maps text to meaning representations which can then be queried and synchronisation links text and model elements via semantic annotation To support both KB querying and synchronisation, Airbus manually developed a knowledge base modelling the domain of SIDP92A001V namely the domain of the electrical and optical system installation.

This knowledge base is composed of a set of OWL ontologies manually constructing by experts working in Airbus company. The ontologies are represented by both OWL and SKOS (Simple Knowledge Organization Systems) languages.

Currently two ontologies are fully developed (Rule and Component ontology), the others are under development.

1. The Rules ontology represents the SIDP rules concepts. It is an OWL-DL ontology and it is composed of more than 2400 concepts.

2. The Component ontology represents the system installation components used by Airbus. It is an OWL-DL ontology and it is composed of more than 2200 concepts. Figure 3 presents an extract from this ontology.

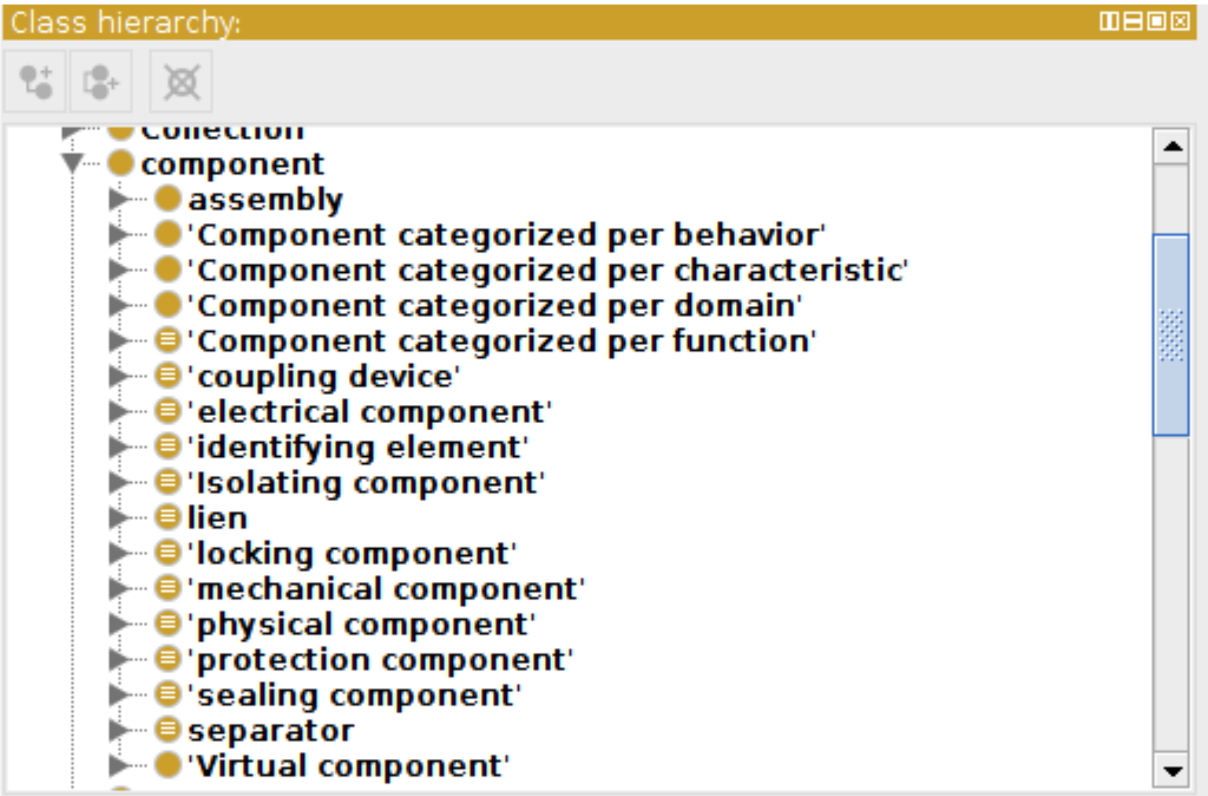


Figure 3: An extract from Component Ontology.

The knowledge base is used in different tasks. Firstly, it identifies the semantic links between the text and model (e.g. link is synonym to attach). This is based on rdfs:label and skos label (ref:label and alt:label). Secondly, the knowledge base ensures the consistency checking of the created links using the ontologies axioms (e.g. disjointWith). Thirdly, the skos:definition labels are exploited to create a raw texts describing the components used by Airbus company.

Because of confidentiality issues, the Airbus data could only be shared after a Non Disclosure Agreement was signed by all interested parties namely, all French partners. This agreement was finalised on June 1st and access to the data was given shortly thereafter.

During the first year of the project, we used this data as follows:

- The SIDP semi-structured rules were processed by CNRS/LORIA and by AIRBUS to automatically construct an RDFS knowledge base encoding the content of these rules.
- The domain specific KB was used by CNRS/LORIA for the semantic annotation of the SIDP rules. More generally, the current version of the semantic annotator can annotate arbitrary text with concepts from the domain specific KB developed by Airbus.
- The domain specific KB is also used to support SPARQL queries on the RDFS knowledge base automatically derived from the SIDP semi structured rules by allowing for e.g., subclass information to be taken into account. Suppose for instance that the KB includes the knowledge that hose pipes, electrical pipes and water pipes are all pipes, then a query asking for all SIDP rules involving a pipe will return rules involving not only pipes but also all rules involving hose pipes, electrical pipes and water pipes.

For the second year of the projet, the aim is twofold.

First, we plan to use the semantic annotator to annotate arbitrary text with KB concepts. The resulting annotated text will then be used as a basis to develop a semantic parser and a generator.

Second, we will investigate whether the parallel data-text corpus build for the SIDP semi structured rules can be used to train/develop a semantic parser capable of mapping SIDP rules contained in SIDP documents to RDFS models.

4. Obeo Data

This section describes the Obeo corpora that are related to the Eclipse IDE and Sirius that is an Eclipse project based on the Eclipse Modelling Framework (EMF).

The EMF project is a modelling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor.

EMF (core) is a common standard for data models, many technologies and frameworks are based on. This includes server solutions, persistence frameworks, UI frameworks and support for transformations. Please have a look at the modelling project for an overview of EMF technologies.

EMF consists of three fundamental pieces:

- EMF - The core EMF framework includes a meta model (Ecore) for describing models and runtime support for the models including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects generically.
- EMF.Edit - The EMF.Edit framework includes generic reusable classes for building editors for EMF models. It provides:
    - Content and label provider classes, property source support, and other convenience classes that allow EMF models to be displayed using standard desktop (JFace) viewers and property sheets.
    - A command framework, including a set of generic command implementation classes for building editors that support fully automatic undo and redo.
- EMF.Codegen - The EMF code generation facility is capable of generating everything needed to build a complete editor for an EMF model. It includes a GUI from which generation options can be specified, and generators can be invoked. The generation facility leverages the JDT (Java Development Tooling) component of Eclipse.

Sirius (see Figure 4) enables the specification of a modelling workbench in terms of graphical, table or tree editors with validation rules and actions using declarative descriptions.

A modelling workbench created with Sirius is composed of a set of Eclipse editors (diagrams, tables and trees) that allow the users to create, edit and visualize EMF models.

The editors are defined by a model that defines the complete structure of the modelling workbench, its behaviour and all the edition and navigation tools. This description of a Sirius modelling workbench is dynamically interpreted by a runtime within the Eclipse IDE.

For supporting specific need for customization, Sirius is extensible in many ways, notably by providing new kinds of representations, new query languages and by being able to call Java code to interact with Eclipse or any other system.

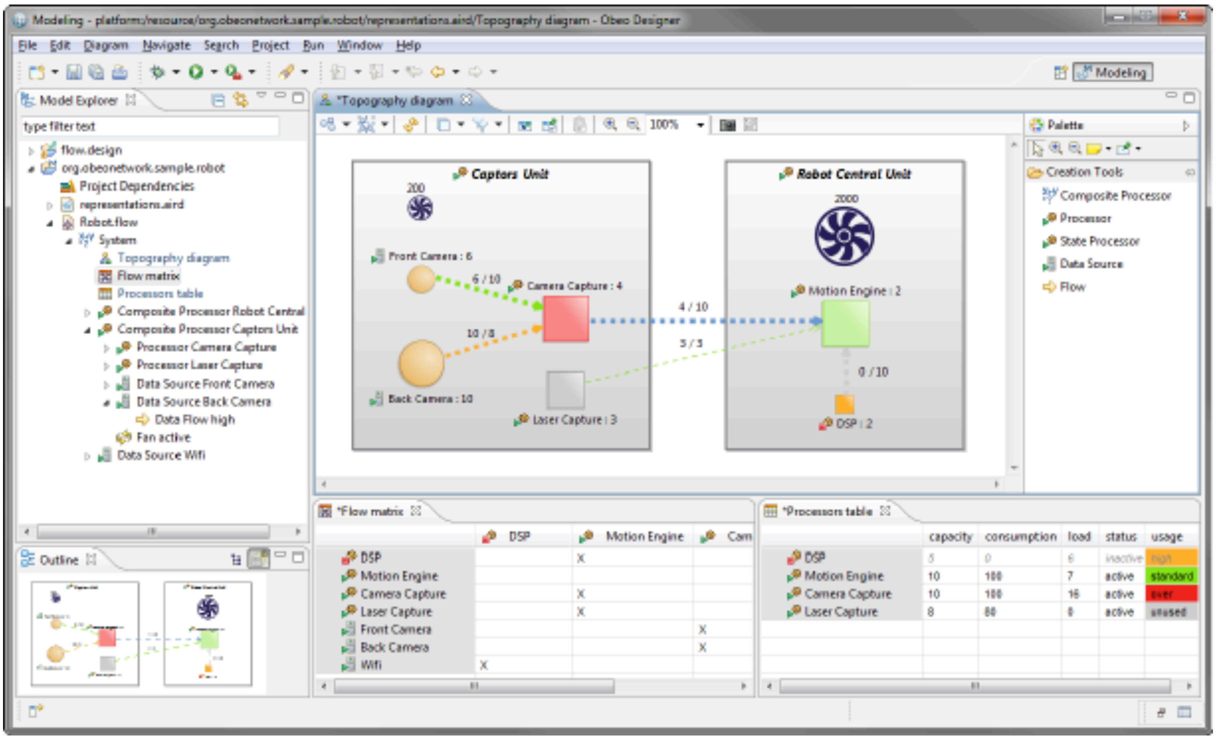


Figure 4. The Sirius Modelling Framework.

The Obeo corpora are composed by a set of three types of documents:

1. Java Concepts: a list of Java identifiers (i.e., classes, interfaces, methods, etc.) related to Sirius (see Figure 5 and Appendix 2).
2. Ecore Concepts: a list of concepts related to Ecore (the EMF meta model) and to Sirius (see Figure 6 and Appendix 2).
3. “TxStyle" Files[1]: a set of files in natural language (i.e., English) related to the documentation of the application being modelled by Sirius (see Figure 7 and Appendix 2).

```
rg.eclipse.sirius.business.api.dialect.Dialect
rg.eclipse.sirius.business.api.dialect.Dialect#DialectServices getServices()
rg.eclipse.sirius.business.api.dialect.Dialect#String getName()
rg.eclipse.sirius.business.api.dialect.DialectManager
rg.eclipse.sirius.business.api.dialect.DialectManager#DialectManager INSTANCE
rg.eclipse.sirius.business.api.dialect.DialectManager#String CLASS_ATTRIBUTE
rg.eclipse.sirius.business.api.dialect.DialectManager#String ID
rg.eclipse.sirius.business.api.dialect.DialectManager#void disableDialect(Dialect dialect)
rg.eclipse.sirius.business.api.dialect.DialectManager#void enableDialect(Dialect dialect)
rg.eclipse.sirius.business.api.dialect.DialectServices
```

Figure 5: Java Concepts (fragment).

[1] http://txstyle.org

```
Sirius Models Contents:

Ecore Model - contribution
       Abstract Class - FeatureContribution
          ref:sourceFeature (1,1)
          ref:targetFeature (1,1)
       Class - IgnoreFeatureContribution -> FeatureContribution
       Class - SetFeatureContribution -> FeatureContribution
       Class - AddFeatureContribution -> FeatureContribution
       Class - RemoveFeatureContribution -> FeatureContribution
       Class - ClearFeatureContribution -> FeatureContribution
       Class - ResetFeatureContribution -> FeatureContribution
       Interface - EObjectReference
       Class - DirectEObjectReference -> EObjectReference
          ref:value (1,1)
       Class - ComputedEObjectReference -> EObjectReference
          attr:valueExpression (1,1)
       Class - Contribution
          ref:source (1,1)
          ref:target (1,1)
          ref:featureMask (1,-1)
          ref:subContributions (0,-1)
          attr:description (0,1)
       Abstract Class - ContributionProvider
          ref:contributions (0,-1)
```

Figure 6: Ecore Concepts (fragment).

Sirius Architecture and Concepts

Introduction

This document presents an overview of the internal architecture of Sirius, and the main concepts and APIs.

Sirius relies heavily on the Eclipse platform, and reuses (and extends) many of the standard Eclipse frameworks, in particular the Eclipse Modeling Platform. This documents assumes that you are already familiar with these frameworks and libraries, in particular EMF and EMF Transaction, GEF and GMF for diagrams. Refer to these frameworks' own documentation for more details about them.

Figure 7: A “TxStyle” file (fragment).

By now, what we have developed is a basic prototype allowing to annotate the “TxStyle” files by establishing links to Java Concepts and to Ecore Concepts. Figure 8 shows how the annotation process is performed. The prototype can be accessed on the GitHub Model Writer repository (https://github.com/ModelWriter/WP6/tree/master/EcoreConcepts-JavaConcepts-Annotator).

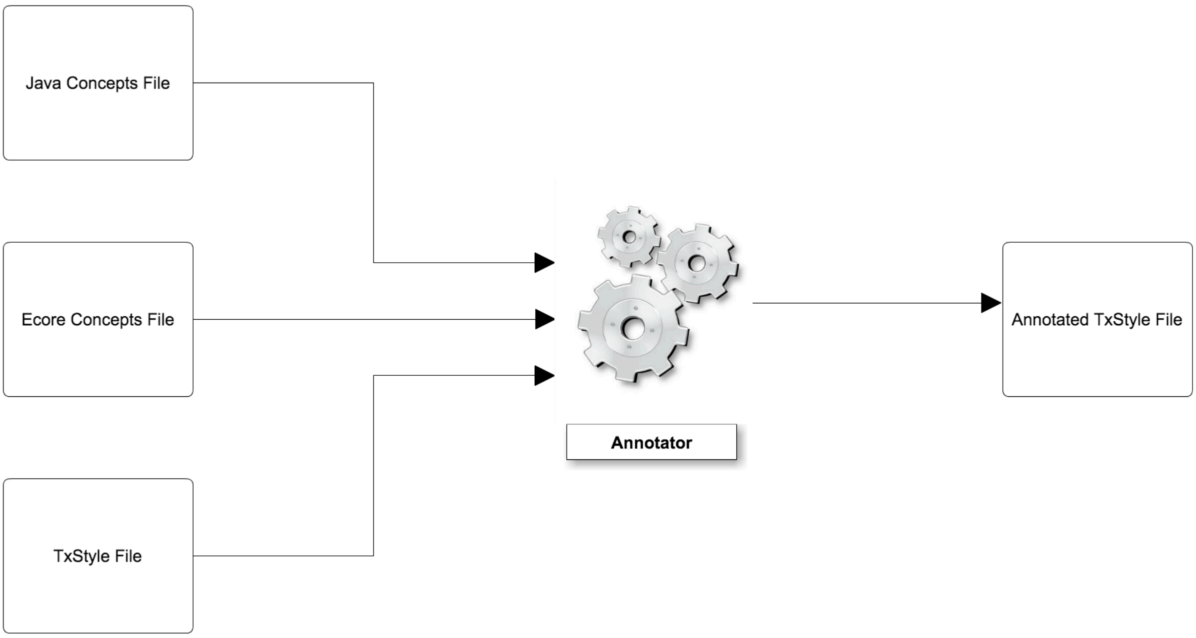


Figure 8. Annotating “TxStyle” Files.

Figure 9 shows a fragment of an annotated “TxStyle” file. Terms in red are related to Ecore Concepts and / or Java Concepts. Numerals represent line numbers.

```
Sirius relies heavily on the Eclipse platform, and reuses (and extends) many of
the standard Eclipse frameworks, in particular the Eclipse Modeling Platform.
This document assumes that you are already familiar with these frameworks and
libraries, in particular [EMF{MW:/EcoreConcepts-line={},MW:/JavaConcepts-
line={506, 2753, 21121, 21915, 21917, 26301, 28191, 26048, 36606, 39064, 39395,
40394}}] and [EMF{MW:/EcoreConcepts-line={},MW:/JavaConcepts-line={506, 2753,
21121, 21915, 21917, 26301, 28191, 26048, 36606, 39064, 39395, 40394}}]
[Transaction{MW:/EcoreConcepts-line={},MW:/JavaConcepts-line={517, 558, 699,
1392, 1492, 1498, 1538, 1895, 21328, 21360, 21395, 21865, 22211, 22212, 22213,
22588, 22589, 22590, 22591, 23089, 24010, 24712, 26459, 26460, 27538, 36178,
36180, 36183, 36327, 36379, 36413, 36438, 36716, 39028, 39123, 39195, 39217,
39761, 40273, 40274, 40311}}], [GEF{MW:/EcoreConcepts-line={},MW:/JavaConcepts-
line={24000, 36609, 36635, 39974, 39997}}] and [MW:/EcoreConcepts-
line={},GMF{MW:/JavaConcepts-line={21667, 21668, 22111, 22112, 22162, 22189,
22190, 22196, 22197, 22198, 22199, 22200, 22201, 22202, 22203, 22206, 22207,
22208, 22209, 22241, 22242, 22243, 22296, 22297, 22298, 22331, 22396, 22404,
22408, 22426, 22436, 23257, 23258, 23259, 23260, 24304, 24342, 24343, 24346,
24374, 24439, 24446, 24929, 25890, 25892, 25895, 25900, 26270, 26316, 27278,
27704, 27705, 27724, 27732, 36619, 39998}}}] for [diagrams{MW:/JavaConcepts-
line={522, 1327, 2069, 2110, 3568, 3596, 3598, 3600, 3601, 3602, 3603, 3604,
3624, 3625, 3626, 3638, 3639, 3640},MW:/EcoreConcepts-line={119, 120, 121, 122,
139, 140, 143, 171, 182, 286, 326}}]. Refer to these frameworks' own
documentation for more details about them.
```

Figure 9. An annotated “TxStyle” file (fragment).

5. Turkish Consortium Data (to be filled by Mantis)

(…)

Appendixes

Appendix 1 Airbus Data

Example SIDP Document
https://github.com/ModelWriter/French-Consortium/airbus/text/SIDP92A001V.docx

Semi-structured design rules:
https://github.com/ModelWriter/French-Consortium/airbus/text/rules.txt
https://github.com/ModelWriter/French-Consortium/airbus/text/rules.xsl

Domain Model (RDFS Knowledge Base modelling plane components)
https://github.com/ModelWriter/French-Consortium/airbus/kb/airbusComponentsKB_03072015.rdf

RDF Knowledge Base derived from Semi-Structured Design Rules
https://github.com/ModelWriter/French-Consortium/airbus/kb/rules.rdf

@Anne: please upload the RDF KB derived by your stagiaires as rules.rdf in the repository listed just above this comment.

Appendix 2 Obeo Data

Ecore Concepts File
https://github.com/ModelWriter/Deliverables/blob/master/WP2/data/obeo/model/EcoreConcepts.txt

Java Concepts File
https://github.com/ModelWriter/Deliverables/blob/master/WP2/data/obeo/model/JavaConcepts.txt

Example of a “TxStyle” file.
https://github.com/ModelWriter/Deliverables/blob/master/WP2/data/obeo/text/Architecture.textile

Example of a partially annotated “TxStyle” file
https://github.com/ModelWriter/Deliverables/blob/master/WP2/data/obeo/text/ArchitectureAutomaticallyAnnotated.textile

Appendix 3 Turkish Consortium Data (to be filled by Mantis)

(…)