Towards Better Document to Model Synchronisation: Experimentations with a Proposed Architecture

Arnaud Michot, Christophe Ponsard and Quentin Boucher
CETIC Research Centre, Charleroi, Belgium

Keywords: Model-based Engineering, Model-Driven Development, Traceability, Usability, Requirements Engineering, Eclipse Modelling Framework.

Abstract: The Model Based Engineering approach is centred around the use of a model repository and a modelling tool. A drawback of the approach is that the evolution of derived documents or other artefacts requires to go back to the model. This process involves tracing the source element back in the model editor before triggering updates. This can reveal quite inefficient and even cause user rejection. This paper presents a reusable architecture including a back channel from a document to the related model. It efficiently supports the locate operation of model elements from the document, model updates from the document (including in-place text editing, e.g. to fix typos inside a description), and even concept creation from a text document. We also report on our experience in implementing this architecture in two modelling tools: the Objectiver goal-oriented requirements engineering tool and the Eclipse Modelling Framework, both with mainstream commercial and Open Source text processors.

1 INTRODUCTION

Modelling is heavily used across many engineering disciplines like the building of bridges, electronic systems and, more recently, software-based systems. Nowadays, Model-Based Engineering (MBE), and more specifically Model-Driven Engineering (MDE) have become mainstream paradigms for software and systems development (Schmidt, 2006). Such approaches can rely on standardised and well adopted modelling languages like UML (OMG, 1997) or SysML (OMG, 2005). Those languages provide a visual syntax enabling the design and communication activities of the engineers but also have precise semantics to enable automation of parts of the System Development Life Cycle (SDLC).

Efficient modelling is achieved through computer tools typically composed of a modelling environment, a model repository and a model transformation tool chain for synchronising modelling artefacts at different steps of the SDLC. Model-to-model and model-to-text transformations are used to generate detailed models from abstract ones and code/documentation from models, respectively. Unfortunately, in practice, only part of such artefacts can actually be generated from source models as target models and texts require to be manually refined and completed. Hence, the MDE approach also has to coexist with the traditional flow of documents produced along the SDLC, like requirements documents, detailed specifications, design documents, test plans, etc. Such documents are also key project milestones required for internal workflows and/or external certification processes. They usually mix text, tables and pictures under either a text processing, or spreadsheet format.

Consequently, the synchronised management of documents and models is a complex task. A key problem is to manage traceability, i.e. how to relate textual and modelling elements. For poorly structured documents, this needs to be achieved manually and is very costly. For documents with a better structure, some form of automated extraction can be considered to initiate a model, or conversely, a structured document can be automatically generated from a model. In both cases, the traceability can be established efficiently and used to maintain a synchronisation. Depending on the context, the synchronisation may be triggered from the model, the document or both (i.e. round-trip mode).

Model to text transformations were defined in the general MOF2Text (OMG, 2008) specification with practical implementations based on XML, like XSLT/FOP, or the Eclipse Modelling Framework (EMF), like GenDoc or M2Doc both based on Ac-

Michot, A., Ponsard, C. and Boucher, Q.
Towards Better Document to Model Synchronisation: Experimentations with a Proposed Architecture.
DOI: 10.5220/0006687505670574
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 567-574
ISBN: 978-989-758-283-7

celeo (Haugommard, 2011; Obeo, 2011). This covers both code and document generation. Unfortunately, little has been done in the area of model to text traceability: most solutions essentially focus on code (Olsen and Oldevik, 2007). For documents, the approach is rather unidirectional and requires to correct the model and regenerate the document. Direct integration with text processors is also missing while most users are far more familiar with such software than with a modelling tool.

On the other hand, direct synchronisation between a model and related documents can also be achieved by using a Domain Specific Language (DSL) used to capture the document structure. Popular DSL implementations like XText (Efftinge., 2006) have their own underlying model with a quite direct (i.e. bijective) mapping with the engineering model that makes the model to model synchronisation easier (Hettel et al., 2008). However, again such techniques cannot be applied inside text processors.

As a result, most commonly reported practices are either that people spend a lot of time to locate the model elements they have to correct w.r.t. a problem noticed in a document (e.g. a typo), or they just do not update the model, meaning they manually correct documents again and again, and probably lose some changes. Our motivation to tackle this problem actually comes from requests of industrial users having to face such costly overheads which strongly impact the efficiency of a MDE approach in their domain.

This short paper reports on our work to address the above limitations by proposing a practical architecture that can cope with modelling frameworks on one side, and mainstream (Open Source or commercial) office suites on the other side. We describe a global architecture covering both model to document and document to model interactions. However, our prototyping will focus on the later and less covered topic. We consider the following three scenarios:

- Locate an element in a model from a text document.
- Perform direct edition of a modelling element from a text document.
- Create a new modelling element from a text document.

In order to validate that our approach can be used in practice, we demonstrate its use on two different modelling environments: a requirements engineering tool named Objectiver (Respect-IT, 2011), and the Capella System Engineering platform (Polarsys/CIS, 2015). We also cover different targets: MS Word documents, Open/Libre Office text documents and HTML documents.

This paper is structured as follows. First, Section 2 presents the proposed architecture. Then, Section 3 details our implementation on the mentioned targets. Section 4 discusses it in the light of related works. Finally, section 5 concludes with the current status of our research and the next envisioned steps.

2 REFERENCE ARCHITECTURE

2.1 Overview

The proposed architecture is depicted on Figure 1. The left part of the picture represents the (system) modelling environment while the right part is the Office environment. Although we will restrict to text document for the rest of the paper, the architecture is general and can also be applied to spreadsheets. In the figure, white components are standard components like the modelling environment (e.g. Capella), the modelling repository (e.g. EMF), document generation libraries (GenDoc/M2DOC/FOP) and office suites (MS Word, Writer). The grey components are additions that need to be implemented to ensure the correct support for the envisioned synchronisation scenarios. The rest of this section will describe them.

2.2 Initial Document Generation

This step is done using the model to text library. Model elements that need to be kept synchronised are tagged with a unique model identifier so they can be traced later. This information can be embedded using a custom field of the target format (docx/odt/html).

Once generated the document can be transferred and opened automatically in the target office platform (or browser in case of HTML generation) using the standard integration API of those components.

2.3 Model to Document Update

If the model is changed then the related documents need to be updated. For this purpose, documents must be kept in the model workspace. The update can be :

- an incremental update of traced elements: the referenced are updated through the server API of the target office platform (e.g. OLE for MS Office, UNO for Open/Libre Office). If some model element are removed, their reference is marked as deleted or,
- in case of full update, the whole document is regenerated using the same process as the initial document generation.

Three update scenarios are possible:

- for documents within the workspace and opened, the update should be immediate and the documents should be refreshed.
- for closed documents within the workspace, the update can be delayed until their next opening.
- for documents exported outside the workspace, the update will be delayed until they are imported back into it (assuming identifiers are preserved).

2.4 Document to Model Update

This part is supporting the proposed scenarios to locate, edit or create concepts from the text processor. This is achieved through a channel from the office platform back to the modelling platform. This requires the following extensions :

- the document editor must support those user interface actions on text bound to a model element.
- once triggered the requested action must be communicated to the modelling platform.
- the modelling platform must act as a server and listen to incoming synchronisation requests to perform the required update.

In order to achieve this communication channel, we rely on a more general web service API developed to expose any kind of model that can be modelled using EMF (Michot et al., 2016). The full API is not required but is useful for other purposes. In case it is not available, the minimal services required are :

- *locate*: GET <server_url>/api/concepts/id/locate
- *edit*: PUT <server_url>/api/concepts/id - the JSON input should contain the attributes/values to update.
- *create*: POST <server_url>/api/concepts - the JSON input should contain the attributes to initialise and, upon success, the service will return the identifier of the created concept.

3 IMPLEMENTATION CASES

In this section, we describe two implementations that cover two different system engineering tools and different target document formats. Each case will have the same structure: first a short description of the tool, then some implementation details and, finally, an overview of the prototyped integration.

3.1 Integration with Objectiver

3.1.1 Description

Objectiver is a goal-oriented requirements engineering tool (Respect-IT, 2011). It relies on a strong modelling approach with a rich meta-model (including goals, requirements, obstacles, refinements, operations, entities, relationships, agents, etc.). It is fully documented in (van Lamsweerde, 2009).

The tool is built around the requirements model and supports a rich set of interplay scenarios involving text, diagrams and tables as depicted in Figure 2 and detailed in (Ponsard et al., 2015). For example a diagram can be generated to describe the responsibilities of an agent. This diagram can then be included in a requirements document along with a list of all the requirements under the agent's responsibility.

3.1.2 Implementation

Both Open/Libre Office and MS Word were integrated with the desktop edition of Objectiver. The noteworthy features and issues are :

- a REST API can be used for integrating with external tools and the web version of the product.
- Open/Libre Office has a server mode based on UNO. MS office can be controlled through OLE.
- context menu extensions are available both for Writer and MS Word. Special fields can be used to embed model elements in text.
- MS Word provides a direct edition field. For Writer, a modal window is needed for editing.
- for uniquely referencing concepts, a specific URI protocol is used: objectiver://project_id/concept_id[/attribute-id]

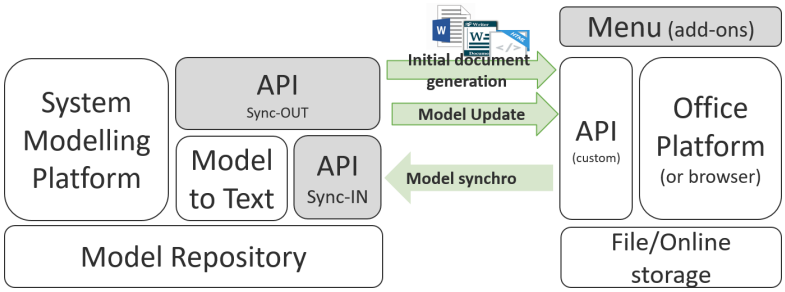

Figure 1: Proposed round-trip architecture.

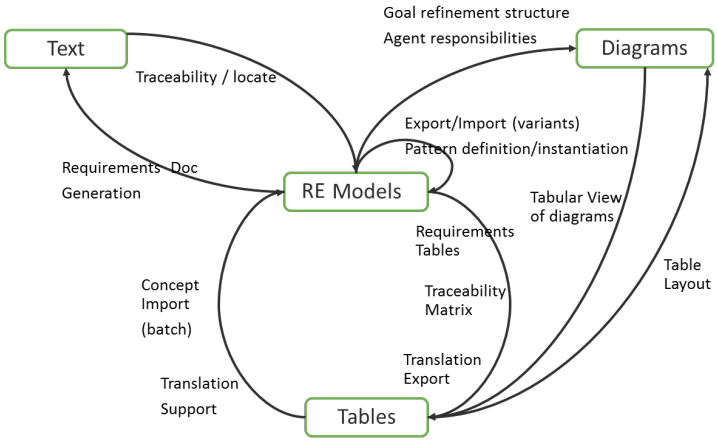

Figure 2: Interactions between text and modelling artefacts.

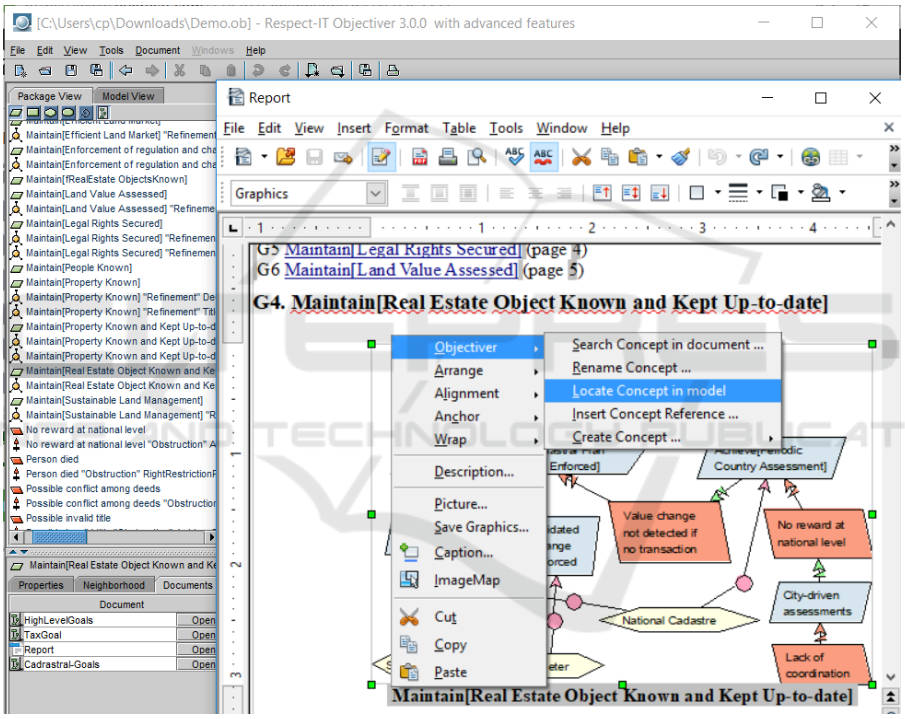

Figure 3: Locate action invoked from Open Office.

3.1.3 Resulting Open/Libre Office Integration

Locate Scenario. The user just has to right click on the concept and select "Locate Concept in Model" from the additional Objectiver menu available at the top of the concept menu. Figure 3 shows this action being invoked on the *Maintain[Real Estate Object Known and Kept Up-to-date]* goal fully described through a diagram just above.

The result is that the concept is located in the main Objectiver window shown in the background. In this window one can see the concept is visible and highlighted in the model navigator area. Note also that the OpenOffice report is among the listed referencing documents in the tab on the bottom right. Clicking on the corresponding "Open" button will trigger the reverse locate from the model to the document. The document will be opened (if not yet) and positioned on the first reference of the concept.

Edit Scenario. The same menu is used. On the concept reference the "Rename Concept" action can be used. Edition is not in place but through a modal box shown in Figure 4. The model and other document references are of course updated accordingly. Alternately, the user can also chose to use the "lo-

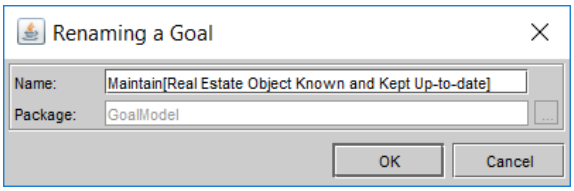

Figure 4: Rename concept action invoked from Open Office.

cate" function and perform the update in the model and it will see the opened document updated directly.

Create Scenario. The user can select some text and decide to tag it as concept of the specified type. The selected text will then be used as concept name. The concept is inserted in the same package as the document and the traceability link is inserted in the document. Such a scenario is more likely from source document than from a report. Figure 5 shows the menu for a create action and also shows the result in the modelling tool (concept listed in navigator and source document referenced in model).

Concept Evolution from Model. Also impacts the document. In addition to the rename operation already discussed, concept deletion is marked using a specific reference as shown in Figure 6. An alternative could be to use the text processor change tracking facilities.

3.2 Integration with EMF

3.2.1 Description

EMF is an Eclipse-based modeling framework and code generation facility for building tools and other applications based on a structured data model (Steinberg et al., 2009). This common standard for data models is used by many technologies and frameworks, e.g. server solutions, persistence frameworks and support for transformations. The Capella system engineering tool is based on it (Polarsys/CIS, 2015).

3.2.2 Implementation

We highlight here a generic implementation based on the core EMF editor. This implementation is available as Open Source (Obeo, 2011) and relies on the following principles. We focus here on the integration with HTML and Word.

- the RAWET REST API provides the required services for interacting with the model from the document (Michot et al., 2016). It was directly plugged on the client tool (local use only) but it can also be used on a EMF repository.
- HTML generation is provided by EGF and can easily be tuned to inject references (e.g. as hyperlinks) in the same format as described previously.

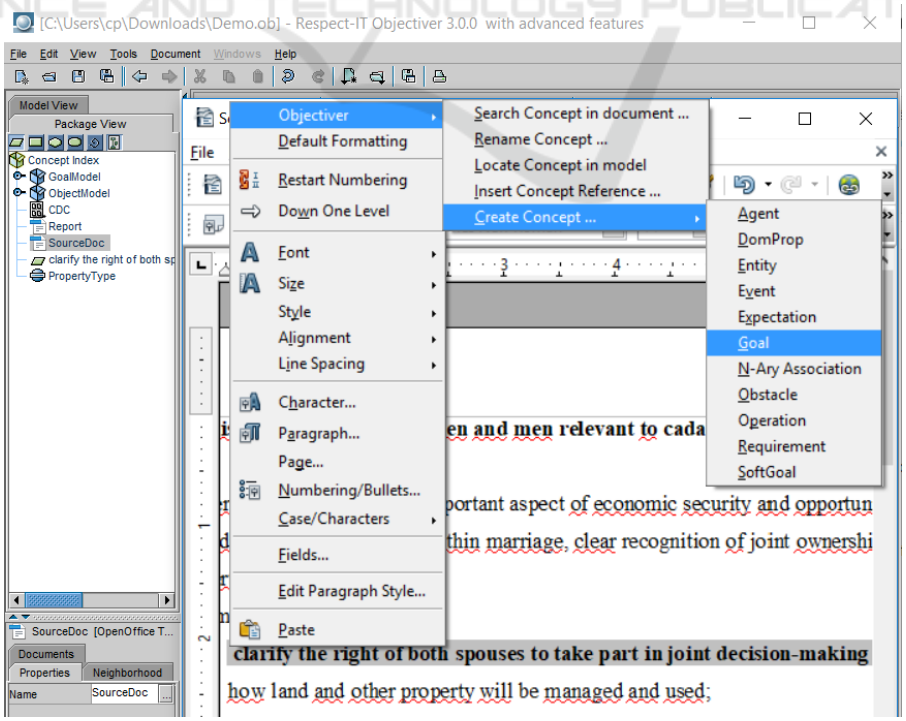

Figure 5: Create action invoked from Open Office.

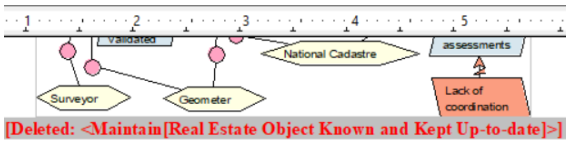

Figure 6: Result of a concept deletion inside a document.

- HTML integration is straightforward using standard JavaScript for event handling and REST calls.
- Word generation was implemented using M2Doc (Obeo, 2011), again with the embedding of references in directly editable custom fields.
- Word menu additions and related commands (e.g. REST calls) are both implemented using Visual Basic. The code is bundled in a template and requires the permission to execute macros.

3.2.3 Resulting HTML/Word Integration

Locate Scenario (HTML). As shown on Figure 7, a click on the "DEMO" concept in the navigator (on the right) triggers the highlighting of the concept in the Eclipse EMF editor (on the left).

Locate Scenario (Word). The locate scenario in Word is quite similar as in OpenOffice. Figure 8 shows the "DOC2M" context menu (on the bottom right) triggered on an mapped field. The effect is similar as for the HTML locate.

Edit Scenario. Edition mechanism is quite similar as for the other tools. The main point is that there is no menu entry as the custom (multi-line) field is directly editable. It can also display the concept type and supports spell checking as shown in Figure 9.

Create Scenario. The create action is visible in Figure 8. Its effect is similar as for the Objectiver tool but within the EMF editor.

Note that edit and create scenarios are less meaningful in HTML except for a web-based editor.

4 RELATED WORK

ModelWriter is an integrated authoring environment providing support for understanding text, guiding the creation of structured models (Erata et al., 2017a). It provides a generic traceability analysis or text-model synchronisation in order to ease documentation maintainability and reduces product defect costs caused by inconsistent or obsolete knowledge. It can consider fragments of text, architecture elements and parts of source code. It also enables reasoning about relation through a proper axiomatisation (Erata et al., 2017b). In comparison, our work is focusing more on a consistent and usable text to model link. Concept traceability in our approach results from the document generator. It is not explicitly modelled as in ModelWriter but could definitely benefit from it.

Synchronisation of Textual Views in the scope of multi-view models is analysed in (Goldschmidt, 2011), especially for partial textual views. There, the author proposes to synchronise transformations from and to the textual view model through an incremental parsing approach. It also defines synchronisation transformations to handle modifications made by editing the text model and update the underlying domain model incrementally. This work can also help in proposing a standard way to define our initial traceability mapping between the model and each document type.

Defining the traceability mapping between the

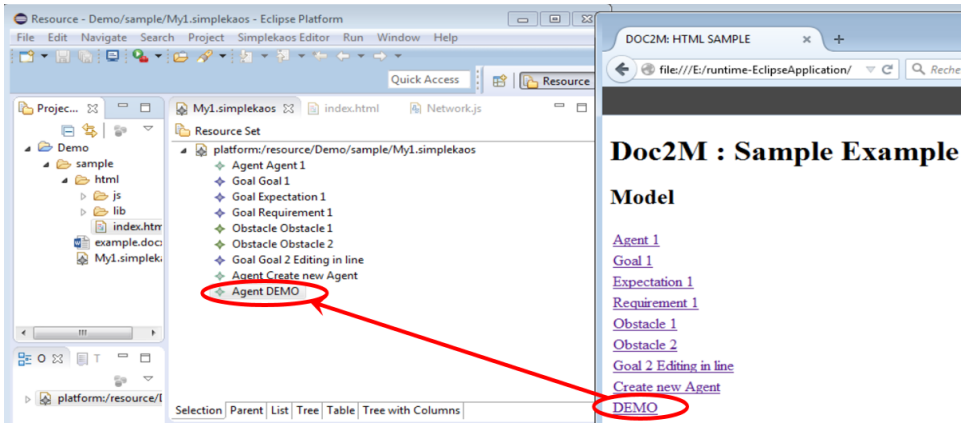

Figure 7: Locate action invoked from a browser.

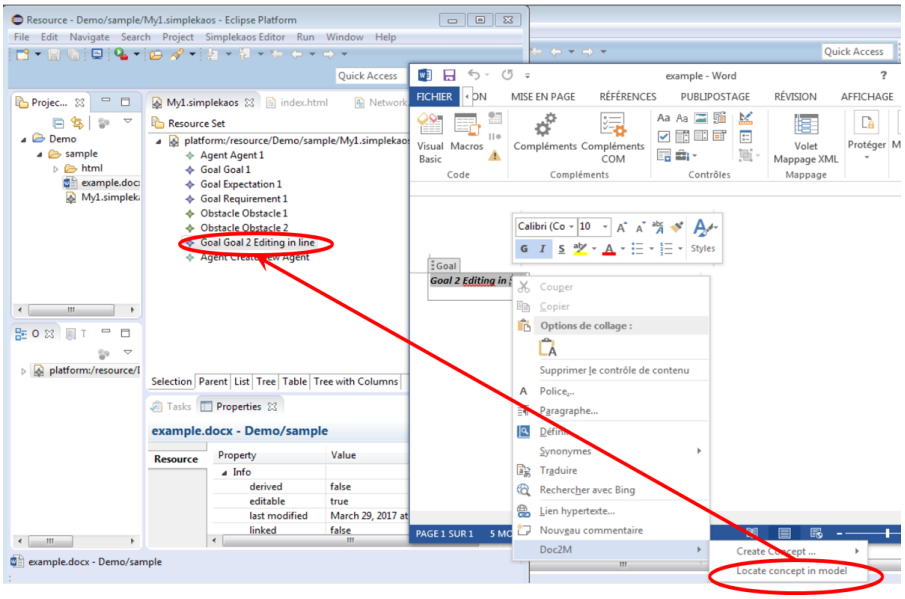

Figure 8: Locate action invoked from Word.

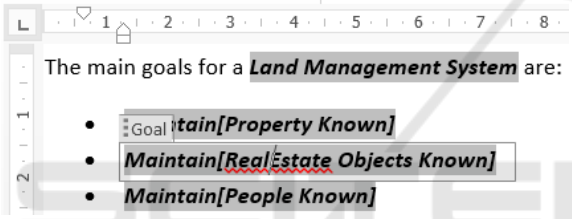

Figure 9: Direct edit of a concept description in Word.

model and the target document can require a lot of tuning. A technique to support the development of M2T transformations by providing automatic corrections from minor changes introduced in the document is proposed by (Guta, 2012). This could ease the evolution of our initial transformation by a non-expert.

5 STATUS AND ROADMAP

This work is motivated by the industrial need for better document to model mappings to complement the existing model to document approaches. After defining a generic architecture, our contribution focused on the less commonly addressed document to model synchronisation. We proposed an URI-based mechanism and prototyped it in both a commercial tool and in the Open Source Eclipse platform. Our work is fully available in Open Source mode (EPL license) on GitHub (Michot and Ponsard, 2017). Based on the proposed architecture and the released code, prototyping a synchronisation can be achieved within a few days of work.

Our work is still partial as our synchronisation is currently limited to concept references (locate, create, rename) and does not yet fully cover the attribute level. We are currently working on extending this mapping and on providing a better integration with widely used Sirius-based editors like Capella (Polarsys/CIS, 2015). Our work is already daily used in the commercial tool implementation but not yet on the Eclipse-based integration. The next step will be to extend our prototyping to support Capella specific views and perform validation with industrial users, mainly from the railway and logistics domains. Looking further ahead, we are also considering the interfacing between SaaS-based modellers and editors (like Google Docs or Open Source alternatives) based on the developed REST API.

ACKNOWLEDGEMENTS

This research was partly funded by the INOGRAMS project of the Walloon Region (grant nr. 7171). We thank Respect-IT for giving us access to their SDK and Obéo for their support about Sirius and M2Doc.

REFERENCES

Efftinge., S. (2006). XText - Language Engineering for Everyone. https://www.eclipse.org/Xtext.

Erata, F. et al. (2017a). ModelWriter Project. https://itea3.org/project/modelwriter.html.

Erata, F. et al. (2017b). ModelWriter: Text and Model-Synchronized Document Engineering Platform. In *32nd IEEE/ACM Int. Conf. on Automated Software*

Engineering (ASE 2017), Urbana-Champaign, IL, United States.

Goldschmidt, T. (2011). *View-based Textual Modelling*. The Karlsruhe series on software design and quality. KIT Scientific Publ.

Guta, G. (2012). Model-to-Text Transformation Modification by Examples. PhD Thesis, Johannes Kepler University, Linz.

Haugommard, A. (2011). GenDoc - From Models to Documents. https://www.eclipse.org/gendoc.

Hettel, T., Lawley, M., and Raymond, K. (2008). Model synchronisation: Definitions for round-trip engineering. In *Proc. of the 1st Int. Conf. on Theory and Practice of Model Transformations*, ICMT '08.

Michot, A. and Ponsard, C. (2017). Doc2M on GitHub. https://github.com/cetic/Doc2M.

Michot, A., Ponsard, C., Zhao, W., and Darimont, R. (2016). RAWET - A generic REST API on top of Eclipse CDO for web-based modelling. EclipseCon France 2016, Toulouse, June 7-9.

Obeo (2011). M2DOc - Generate Word Documents From Your Models. https://www.eclipse.org/gendoc.

Olsen, G. K. and Oldevik, J. (2007). Scenarios of Traceability in Model to Text Transformations. In *Proc. of the 3rd European Conf. on Model Driven Architecture-foundations and Applications*, ECMDA-FA'07.

OMG (1997). Unified modeling language. http://www.omg.org/spec/UML.

OMG (2005). System modeling language. http://www.omg.org/spec/SysML.

OMG (2008). Mof model to text transformation language. http://www.omg. org/spec/MOFM2T.

Polarsys/CIS (2015). Capella - open source solution for model-based systems engineering. https://www.polarsys.org/capella.

Ponsard, C., Darimont, R., and Michot, A. (2015). Combining models, diagrams and tables for efficient requirements engineering : Lessons learned from the industry. In *Proc. INFORSID, Biarritz, France, May*.

Respect-IT (2011). The objectiver tool - version 3. http://www.objectiver.com.

Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):25–31.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2009). *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition.

van Lamsweerde, A. (2009). *Requirements Engineering - From System Goals to UML Models to Software Specifications*. Wiley.