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

D3.2.1 M2M Transformation Framework Architectural Design Document

ModelWriter

Text & Model-Synchronized Document Engineering Platform

Project number: ITEA 2 13028
Work Package: WP3
Task: T3.2 - Specification and design of the M2M Transformation

Edited by:

Ferhat Erata <ferhat.erata@unitbilisim.com> (UNIT)
Moharram Challenger <moharram.challenger@unitbilisim.com> (UNIT)

Date: 07-June-2015
Document version: 1.0.0

Document History

Version	Author(s)	Date	Remarks
0.5.0	Ferhat Erata Moharram Challenger	07-June-2015	Draft
0.6.0	Developer team in UNIT	09-Sep-2015	Modifications

Table of Contents

1. Introduction

Role of the deliverable

This document is the first version of the description of the use cases proposed by the French consortium. It may be up-dated depending on the further details and requirements we get from our industrial use case providers.

The List of Technical Work Packages

UC Code	Requirements derived from
WP2	Semantic Parsing and Generation of Documents and Documents Components
WP3	Model to/from Knowledge Base (synchronization mechanism)
WP4	Knowledge Base Design and Implementation
WP6	Architecture, Integration and Evaluation

Conventions

The requirements are prefixed by “REQ-SR-WPz-xxx”, and are written in a roman typeface, where “REQ” stands for “Requirement”, “SR” indicates “Software Requirements”, “z” stands for the number of work package where the requirement is orginated and “xxx” is the positive integer identifier of the requirement. You can add this id (xxx) which is unique entire ‘requirements’ repository, at the end of https://github.com/ModelWriter/Requirements/issues/ to access the latest version of the requirement.

Structure of the document

This document is organized as follows:

- Chapter 1 introduces the document.
- Chapter 2 describes the specification of the M2M
- Chapter 3 describes the design of the M2M
- Annex 1

Terms, abbreviations and definitions

Abbreviation	Definition
RDF	Resource Description Framework
WP	Work Package
UC	Use Case

This document consists of the specification and design of the M2M Transformation Framework whose main goal is to make the ModelWriter tool able to launch M2M (model-to-model) transformations including the following features:

- To obtain one or several output models from one or several input models.
- To configure the transformations to be able to produce different outputs using the same inputs.
- To compose simple transformations to obtain more complex ones.
- To keep traces between transformed models and its source models.
- To synchronize the output models after its input models or configurations have been modified.
- To synchronize the output models without changing the modifications that users or processes may have made on them after the transformations was finished.

2. Specification of M2M Transformation

1. Text Part

Feature 1.1:

User shall be able to mark any kind of textual documents on Eclipse Editor.

- User shall be able to mark text fragments on a Markdown, Wikitext.. file (Eclipse Wiki Editor).
  - plugin: org.eclipse.mylyn.wikitext.ui
  - editor: org.eclipse.mylyn.internal.wikitext.ui.editor.MarkupEditor
- User shall be able to mark text fragments on a Java file (Eclipse JDT Java Editor).
  - plugin: org.eclipse.jdt.ui
  - editor: org.eclipse.jdt.internal.ui.javaeditor.CompilationUnitEditor
- User shall be able to mark text fragments on an XML file (Eclipse XML Editor).
  - plugin: org.eclipse.wst.xml.ui
  - editor: org.eclipse.wst.xml.ui.internal.tabletree.XMLMultiPageEditorPart
- User shall be able to mark text fragments on a Plain text file (Eclipse Text Editor).
  - plugin: org.eclipse.ui.editors
  - editor: org.eclipse.ui.editors.text.TextEditor
- User shall be able to should mark text fragments on a Textual DSL (Eclipse Xtext Editor).
  - editor: org.eclipse.xtext.xbase.ui.editor.XbaseEditor

Feature 1.2:

User shall be able to see the start and end char positions of markers shifting while editing on the text editor.

- Offset and Length of markers should be updated while editing on the text editor.

Feature 1.3:

The system shall indicate text fragments or a model elements which are already linked by means of a kind of visual indicator.

Feature 1.4:

User shall be able to delete any marker based on a valid text selection on the editor.

- Precondition: the text should be marked before deletion.

Feature 1.5:

User shall be able to mark a text fragment by the 'Mark All' command to indicate that all text fragments with the same syntax should be searched and marked as well as different representations of single entity.

- Although the system should assign unique IDs to all markers, another group ID should be also assigned.

Feature 1.6:

User shall be able to delete a marker which has been already marked by the 'Mark All' command by means of the 'Delete All' command to indicate that all the related markers should be also deleted.

Feature 1.7:

The System shall persist a text marker and its state.

Feature 1.8:

User shall be able to undo/redo markers and their states while working on Text editor.

Model Part

Feature 2.1:

User shall be able to mark an element which inherits `ENamedElement` on an Eclipse Ecore Editor.

- User shall be able to mark an EMF model element on EcoreEditor/EMF Reflective Editor.
  - plugin: org.eclipse.emf.ecore.editor
  - editor: org.eclipse.emf.ecore.presentation.EcoreEditor
- User shall be able to mark a EMF model element on Generic EMF Form Editor.
  - plugin: org.eclipse.emf.generic.editor
  - editor: org.eclipse.emf.editor.EEditor

Feature 2.2:

User shall be able to mark an element which inherits `NamedElement` on an Eclipse UML Editor.

- User shall be able to mark a UML model element on the tree-based UML2 editor of Eclipse.
  - plugin: org.eclipse.uml2.uml.editor
  - editor: org.eclipse.uml2.uml.editor.presentation.UMLEditor
    - ❖ selection: org.eclipse.uml2.uml.NamedElement
- User shall be able to mark a UML model element on the free-form UML editor of Eclipse.
  - plugin: org.eclipse.papyrus.editor
  - editor: org.eclipse.papyrus.editor.PapyrusMultiDiagramEditor
- User shall be able to mark a UML model element on the free form Sirius editor of Eclipse.
  - plugin: org.eclipse.sirius.diagram.ui.ext
  - editor: org.eclipse.sirius.diagram.ui.tools.internal.editor.DDiagramEditorImpl
    - ❖ selection:org.eclipse.emf.ecoretools.design.ui.parts.DNodeListEditPartWithAlpha
    - ❖ selection:org.eclipse.sirius.diagram.ui.internal.edit.parts.DNodeContainerEditPart

- ❖ selection:org.eclipse.sirius.diagram.ui.internal.edit.parts.DNodeListEditPart
- ❖ selection:org.eclipse.sirius.diagram.ui.internal.edit.parts.DEdgeEditPart

Feature 2.3:

User shall be able to mark OMG ReqIF model elements such as `SpecObject` and `SpecHierarchy`.

- plugin: org.eclipse.rmf.reqif10.pror.editor (RMF is still in incubation phase)
- editor: org.eclipse.rmf.reqif10.pror.editor.presentation.SpecificationEditor
  - selection: org.eclipse.rmf.reqif10.Specification
  - selection: org.eclipse.rmf.reqif10.SpecObject
  - selection: org.eclipse.rmf.reqif10.SpecHierarchy

Feature 2.4:

User shall be able to mark OMG BPMN2.x model elements such as Gateways and Activities

Feature 2.5:

Once an instance of EMF model is marked on the `EcoreEditor`, the offset and length of the corresponding XMI statement should be also marked.

Feature 2.6:

User shall be able to delete an `ENamedElement` on the `EcoreEditor`.

- Once an instance of `EClass` is deleted, Markers on its EStructuralFeatures should be deleted.
- Once an instance of `EAttribute` is deleted, its marker should also be deleted.
- Once an instance of `EReference` is deleted, its marker should also be deleted.
- Once an instance of `EPackage` is deleted, all `Subpackages`, `EClassifiers` and `EStructuralFeatures` of those `EPackage` should be deleted recursively.

Feature 2.7:

User shall be able to undo/redo model markers and their states while working on Model editors.

Model <-> Text

Feature 3.1:

Show mapping between arbitrary model and text markings.

Feature 3.2:

Show the filter on the project explorer view on the mapping wizard.

Feature 3.3:

Show filtered relation types according to selected marker type.

Feature 3.4:

Show the markers which have been filtered according to selected relation type and selected marker.

Views (UI parts)

Feature 4.1:

Show ModelWriter Markers which marked current document on Master View.

Feature 4.2:

Show target markers of selected marker on Target View.

Feature 4.3:

Show source markers of selected marker on Source View.

Feature 4.4:

Show details of marker which is selected from Master View on the Properties View.

Feature 4.5:

Navigate and focus to markers through selected marker on Views.

Traceability

Feature 5.1:

The System shall persist a model marker and its state. The persistency formalism must align with set theory and relational calculus.

Feature 5.2:

Virtualize markers and their relations.

Configuration

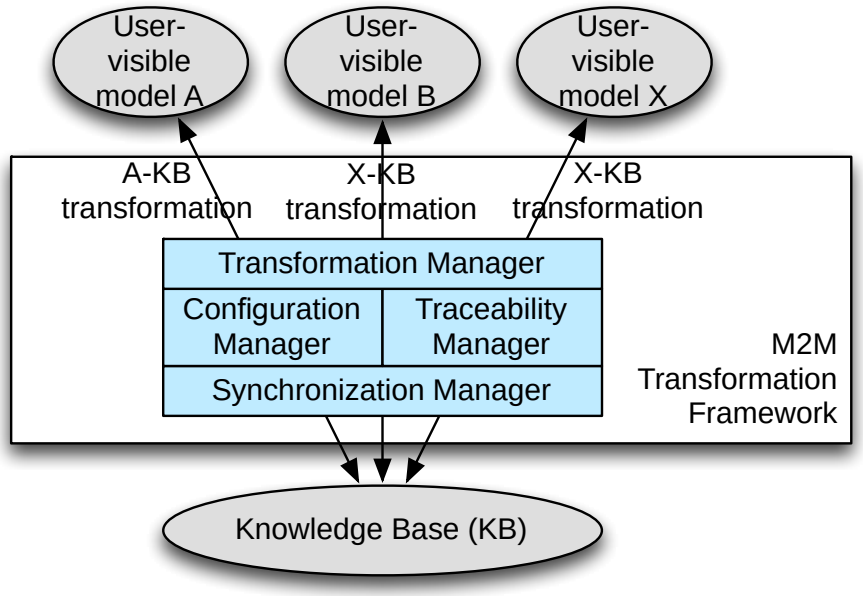
Feature 6.1:

Markers and links (mappings) shall be able to specify by the user in a declarative way.

Feature 6.2:

The specification formalism must support First-order Logic constraints.

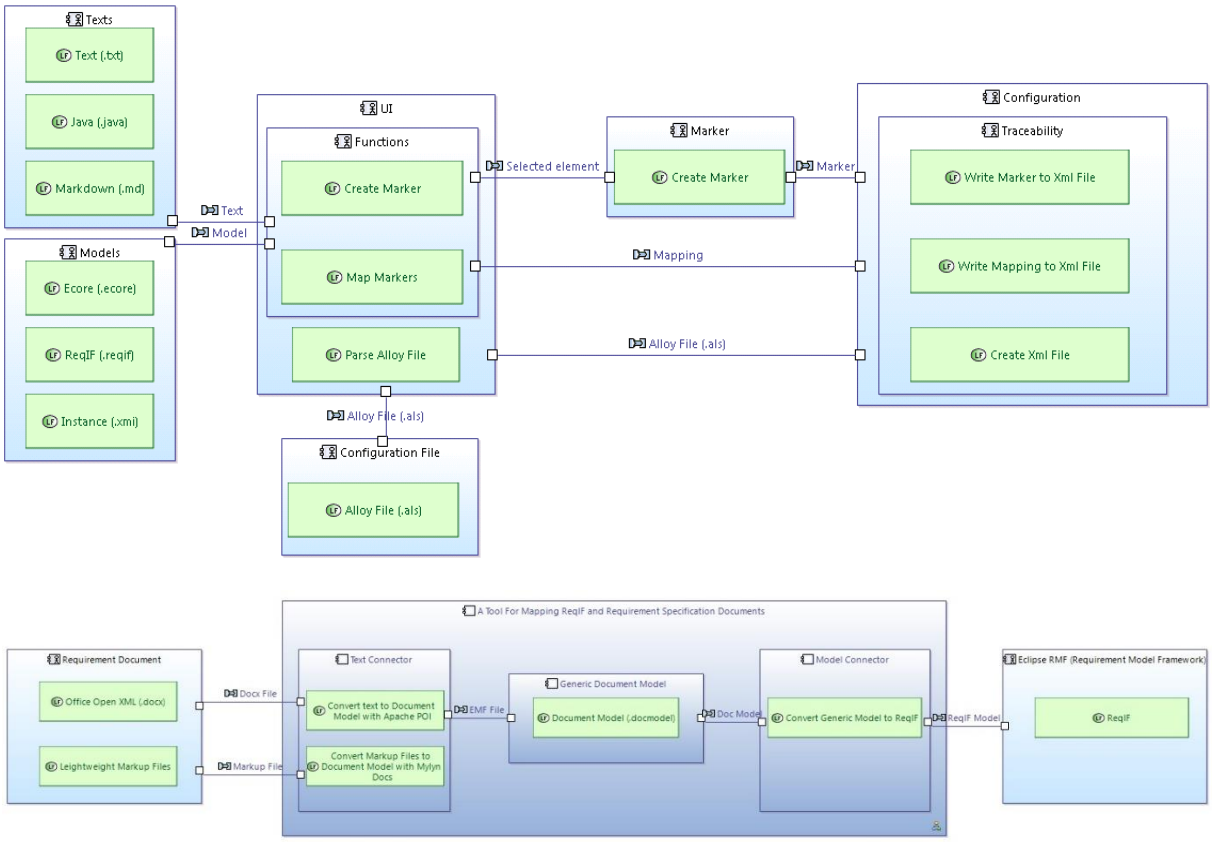
The following image offers an overview of the components of the M2M Transformation Framework.



All the components will be designed as Eclipse plugins in order to make the framework usable and extendable by third-party companies.

https://github.com/ModelWriter/Deliverables/issues/34

3. Design of The M2M Transformation



User give an alloy file (".als") to system via "UI" and "Configuration" parses the file in the background and creates an xml file via "Traceability". So marker types and relation types are determined. Then Users open text and model documents. Select a part of text and mark this area via "UI". "Marker- part" creates marker according to selected type and selected area. And then marker data is written by "Configuration" to the xml file which is already created. The same steps are repeated for the selected model element.

After markers have been created, mapping can now be done. For mapping user selects a marker as source, then selects a relation type and selects markers as target. "Configuration-part" also writes the mapping data to the xml file via "Traceability-part". So user mapped a model element and a text element.

Texts

It includes Text (.txt), Java (.java), Markdown (.md), and etc. type text files. Any kind of these files can be used to marking action.

Models

It includes Ecore (.ecore), ReqIF (.reqif), XML instance (.xmi), and so on. type model files. Any kind of these files can be used to marking action.

Configuration File

There is an Alloy (.als) file to describe sets and relations between sets. Sets correspond to marker types and relations correspond to mapping notion in our system.

UI

This part includes “Create Marker” and “Map Markers” commands.

Create Marker:

Triggers creating marker action in the Marker part according to selected text or model element.

Map Markers:

Connects existing markers with existing relation types.

Parse Alloy File:

Triggers parsing alloy file action in Configuration.

Marker

Creates marker with selection which comes from UI.

Traceability

It is an API that records markers and their relations to xml file for persistence.

Configuration

It uses Traceability API to create xml file and write marker, mapping to xml file.

Write Marker to Xml File:

Writes marker to xml file via Traceability API.

Write Mapping to Xml File:

Writes mapping relation to xml file via Traceability API.

Create Xml File:

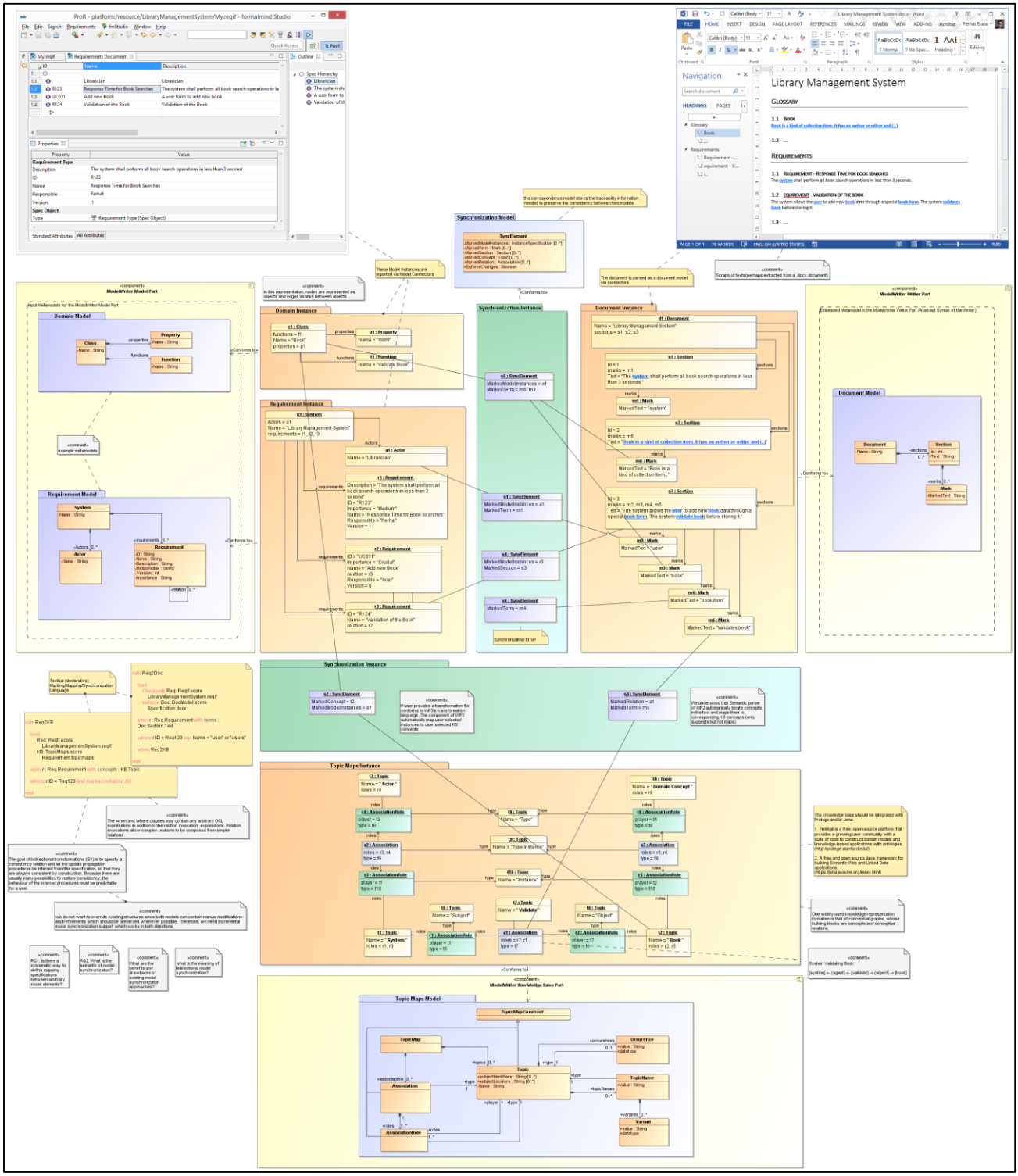
Parses alloy file and then creates template xml file according to result of parsing.

4. Towards implementation

The following figure illustrates the snapshot of the internal representation of ModelWriter Platform between two consequtive user interactions. The platfor has three main modules: Model part (at the left side of the figure), Writer part (at the right side of the figure), and Knowledge base part (at the buttom of the figure). In this big picture, the simplified metamodels are used to ease the understanding of the system. These metamodels are called “Document Model”, “Domain and Requirement Models”, and “Synchronization Model” for Writer, Model, and KB parts respectively. Document Model is used to decompose a text in the document and restore in an structural and meaningful way. Domain Model and Requirement Model are sample models which are used in the Model Part as the User-Visible model. However, ModelWriter will support adding various modeling languages by configuring ModelWriter. The Synchronization metamodel aims to keep the traceability links between texts and models.

The figure below illustrates snapshot of ModelWriter. Using Microsoft Word’s OOXML standard (.docx) and the Text Connectors provided in WP6, a requirement document is imported to the system as an instance of the Document Model. The user also provides two instance model for the Model part of the system. Considering the KB part, “System validates Book” statement is restored.

https://github.com/ModelWriter/Deliverables/issues/34



https://github.com/ModelWriter/Deliverables/issues/34

5. Conclusion and way forward

The initial version of the architecture for WP3 is discussed in this document. The architecture and the implemented features will be updated in the next phases by adding new features while integrating with other components. One of the main components in WP3 is the transformation between the models and knowledge base which will be realized in the second year of the project.

References

N/A