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

BUMBLE Deliverable D3.3 (Version 2)

BUMBLE Methodology

Edited by: BUMBLE Team
Date: June 2022

Project: BUMBLE - Blended Modeling for Enhanced Software and Systems Engineering

Contents

1. Introduction

The purpose of this deliverable is to provide a technology-agnostic overview of the BUMBLE approach to realize blended modeling.

This document contains three major parts. The first part is the BUMBLE topology, which defines the building blocks of our solutions and how they fit together. The topology is not meant as a technical architecture, but as a technology-agnostic overview of the functional parts of the solutions. The second part is the taxonomy. The taxonomy is a definition of the terminology by means of an ontology that depicts the relations between the different terms. Finally, the third part of this deliverable describes a technology-agnostic workflow for the application of the BUMBLE solutions from the viewpoint of a blended modeling language engineer.

2. Topology

BUMBLE is about blended and collaborative modeling. We follow common terminology and approaches used in model-driven practices as far as possible. Since it is common to use modeling for different purposes we distinguish the common *metalevels* of models:

- M1 are the actual models that model non-language aspects like printer cars, or income-tax laws. Here is where the users enjoy the blending and collaboration functionality.
- M2 are the models that define the languages of the M1 models and the way different languages blend together.
- M3 are the models that define the modeling-languages in which the M2 models are written.

Figure 1 shows the topology of an example of the functional parts that blend and synchronize M1 models. Figure 2 shows the topology of an example of the functional parts that define the languages and their transformations. It also shows how these M2 models relate to the M1 topology. It is shown that the functionality of M1 is derived from the M2 specifications, by means of generators.

Figure 1 depicts two modeling client environments that are involved in a collaboration session where their models are immediately synchronized with each other. Furthermore, the modeling languages (and possibly MDSE technologies) are different from each other. There is also blending of different concrete syntaxes happening in both client environments. To make this happen there are several transformations happening, both in the client- as in the server environments. The synchronization between the different environments is done by two model distribution services that exchange the mutations for the synchronized models. Mind that remote synchronization only takes place between models that conform to the same metamodel (are written in the same language).

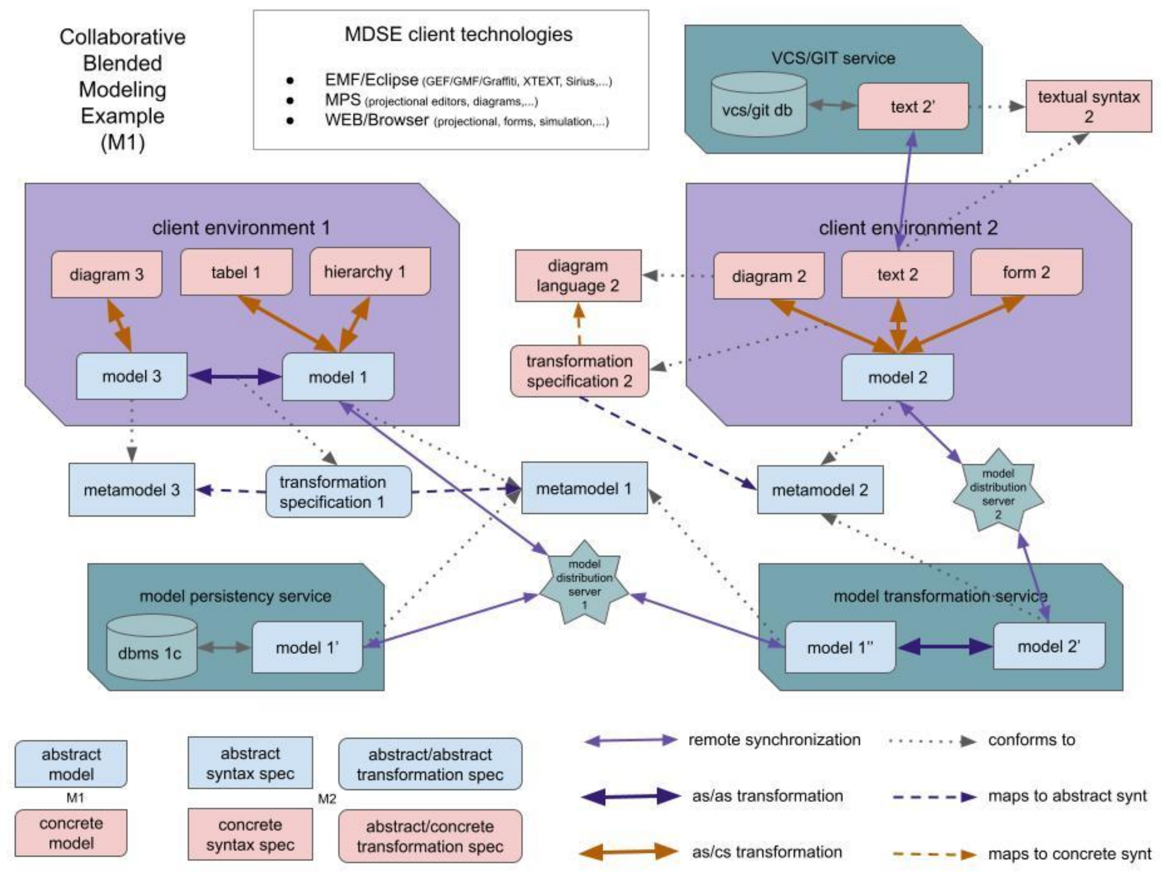

Figure 1. M1 topology

The goal of the BUMBLE project is to develop solutions for blending for different MDSE technologies, and that are usable for all possible modeling languages. MDSE technologies already have generic functionality to define modeling languages. We need to extend this functionality to define transformations between those languages. It very much depends on the MDSE-technology involved if and how this should be realized.

Figure 2 shows that the transformations of the different syntaxes (blending) is realized by an modeling-environment that is (partly) generated based on language and transformation specifications. These M2 level models are also modeled in a modeling environment (the yellow one in figure 2). A generator (the yellow arrow) generates parts of the M1 modeling environment (the purple part). How and which solutions need to be developed in the BUMBLE project for defining and executing the transformations is very much dependent on the involved MDSE technologies (e.g. PMS or EMF).

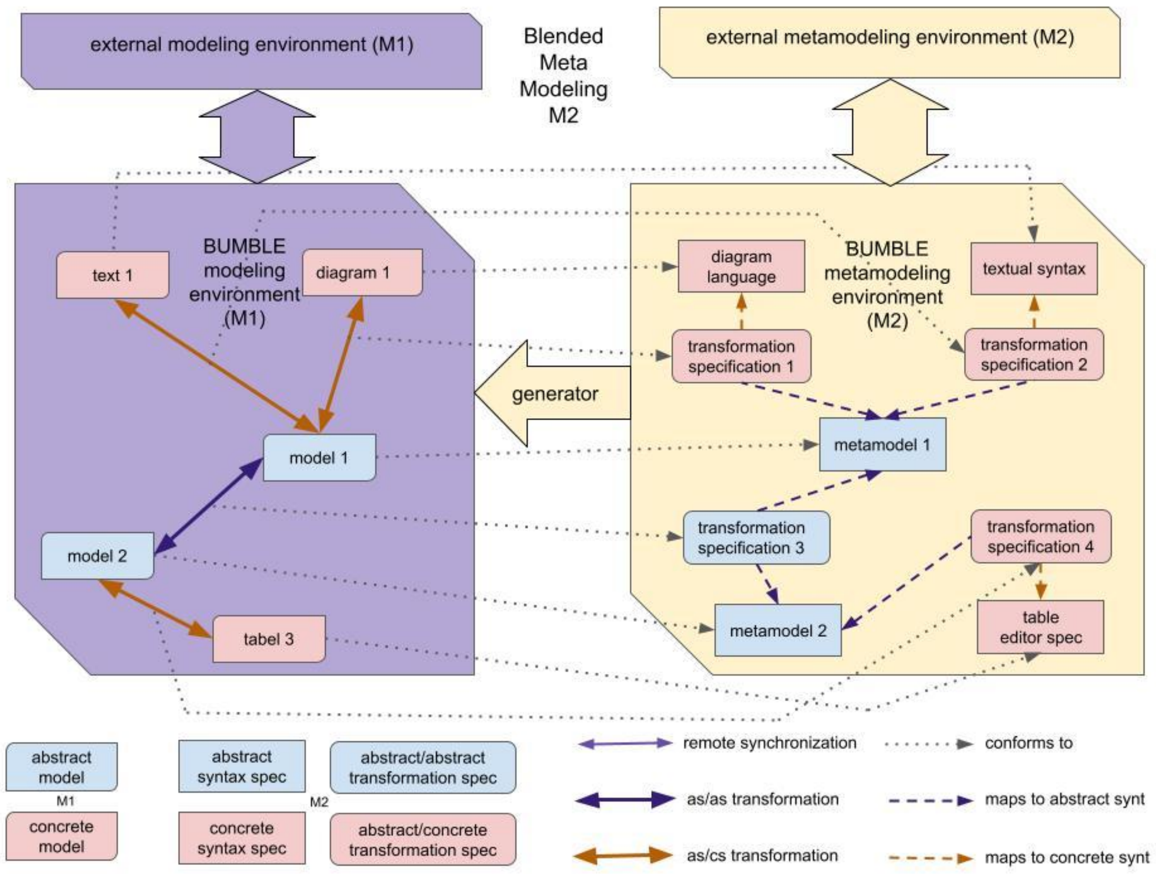
external modeling environment (M1)
Blended Meta Modeling M2
external metamodeling environment (M2)
BUMBLE modeling environment (M1)
text 1
diagram 1
model 1
model 2
tabel 3
generator
BUMBLE metamodeling environment (M2)
diagram language
textual syntax
transformation specification 1
transformation specification 2
metamodel 1
transformation specification 3
transformation specification 4
metamodel 2
table editor spec
abstract model
concrete model
M1
abstract syntax spec
concrete syntax spec
abstract/abstract transformation spec
abstract/concrete transformation spec
M2
remote synchronization
as/as transformation
as/cs transformation
conforms to
maps to abstract synt
maps to concrete synt

Figure 2. M2 and M1 relation topology

3. Taxonomy

Figure 3 shows the terms and the relationship between the terms.

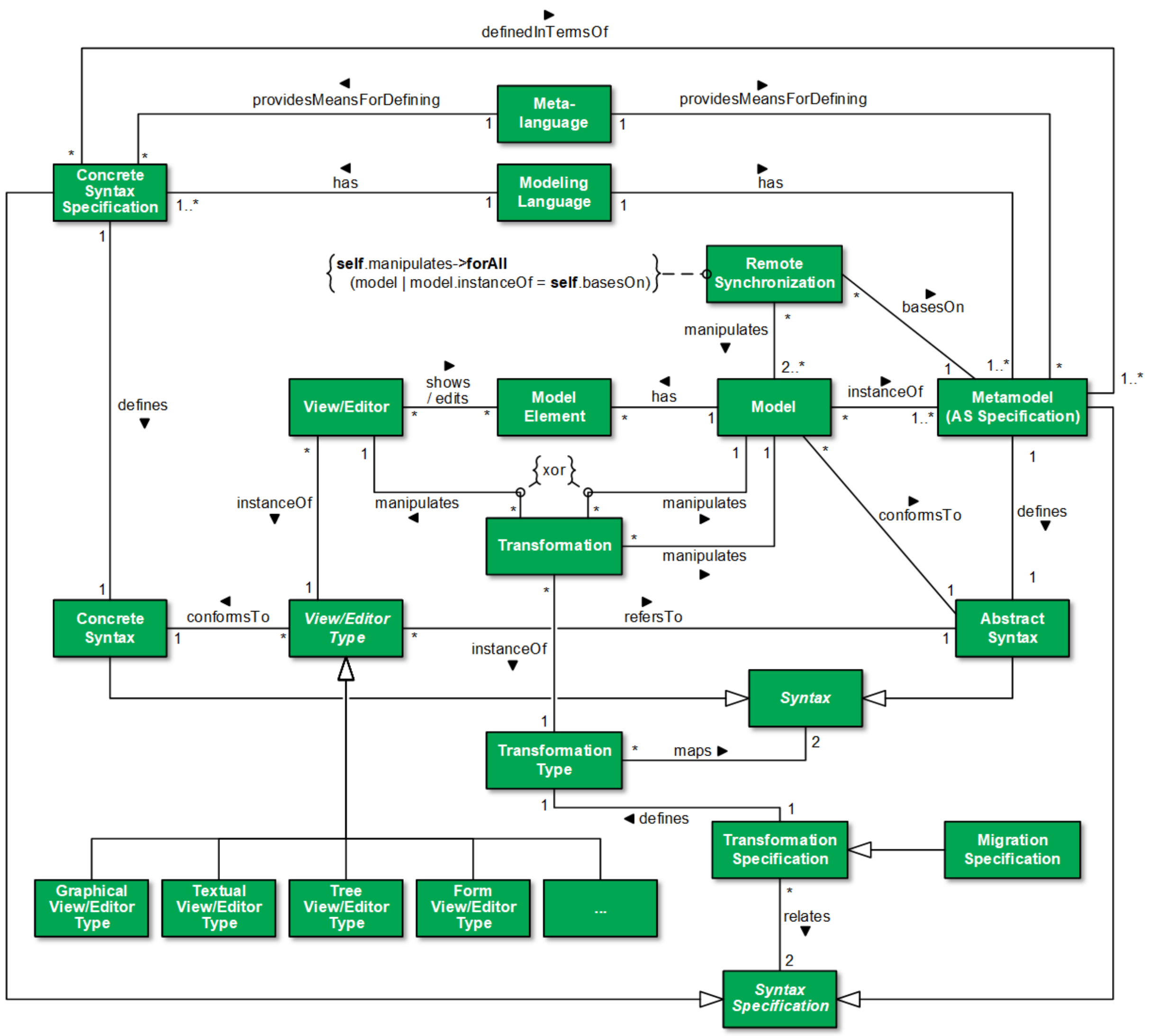

Figure 3. taxonomy

Model
"A model represents an aspect of a system under development captured in a specific instance of a [machine-processable] ***[modeling] language*** that serves a purpose within the development lifecycle."[1]

Modeling Language
A language defines the ***syntax*** and static semantics of ***model***s. The syntax defines the concepts and rules to be used and conformed to, for any ***model*** to be a well-formed instance of that ***modeling language***.

Metamodel
A metamodel defines the ***abstract syntax*** of a ***modeling language***.

Syntax
The syntax defines how to read and write (either by humans or by computers) ***model***s of a specific ***modeling language***. The syntax is defined by and has to conform to a ***syntax specification***, which is part of a ***modeling language***. A syntax can be an ***abstract syntax*** or a ***concrete syntax***.

Syntax Specification
A syntax specification defines a ***syntax*** for a ***modeling language***.

Abstract Syntax
The abstract syntax defines the concepts and rules by which structure models shall be written in a specific ***modeling language***. The abstract syntax is mainly useful for the static semantic aspects of the ***model***s. The abstract syntax is defined by and has to conform to an abstract syntax specification (i.e., a ***metamodel***), which is part of a ***modeling language***.

Concrete Syntax Specification
A concrete syntax specification defines a ***concrete syntax*** for a ***modeling language***.

Concrete Syntax
The concrete syntax is how humans and computers interact with ***model***s of a specific ***modeling language***. The concrete syntax is defined by and has to conform to a ***concrete syntax specification***.

Metalanguage
A metalanguage is a language that provides means for defining an aspect (***metamodel***, ***concrete syntax specification***, static semantics, ...) of a ***modeling language***.

[1] J. Holtmann, J.-P. Steghöfer, M. Rath, D. Schmelter: Cutting through the Jungle: Disambiguating Model-based Traceability Terminology. RE 2020: 8-19

Remote Synchronization
A synchronization mechanism that keeps two or more models of the same language edited by different editors the same across a network. Changes in one model are propagated to equivalent changes in the other model.

Transformation
A transformation is a manipulation of a pair of one ***model*** and another ***model*** or a ***view*** that preserves the relation between them according to a ***Transformation Specification*** for the two syntaxes involved.

Transformation Specification
A transformation specification is a specification of a ***Transformation Type. Transformation Type***s map two syntaxes, therefore transformation specifications relate two ***syntax specifications***.

Transformation Type
A transformation type is a meaning-preserving relation between two ***syntax****es*. A ***syntax*** can be an ***abstract syntax*** or a ***concrete syntax***. The transformation type is defined by and has to conform to a ***transformation specification***.

Migration Specification
A migration specification is a ***Transformation Specification*** that specifies a ***Transformation Type*** that maps a ***Metamodel*** to an evolved ***Metamodel*** that is a new version of the first ***Metamodel***.

View/Editor Type
"A view[/editor] type defines rules according to which ***view***s[*/**editor***s] of the respective type are created"[2] based on the ***concrete syntax*** and thereby its ***concrete syntax specification*** of a ***modeling language***. "It defines the set of metaclasses whose instances a ***view***[***/editor***] can display [and can be edited]."[2] A view/editor type can be of graphical, textual, tree-based, form-based, … nature.

View/Editor
"A view[/editor] is the actual set of objects and their relations [(i.e., the elements of a ***model***)] displayed using a certain representation and layout [and providing the allowed editing commands]. A view[/editor] resembles the application of a view[/editor] type on the [...] ***model***s. A view[/editor] can therefore be considered an instance of a ***view***[/***editor***] ***type***."[2]

[2] T. Goldschmidt, S. Becker, E. Burger: Towards a Tool-Oriented Taxonomy of View-Based Modeling. Modellierung 2012.

4. Methodology

Figure 4 depicts our workflow for creating a blended modeling environment independently of the BUMBLE technology spaces Eclipse and MPS. The process of a DSML engineer is pretty straight-forward. A DSML engineer creates specifications for the abstract-syntax, concrete syntax and transformations. Subsequently, the DSML engineers realize the transformation and the view/editor types, followed by bundling the types in a blended-modeling environment. In this context, "realize" means that the transformation types, view/editor types, and the blended modeling environment can be automatically generated or manually hand-crafted, or a mixture of both approaches can be applied.

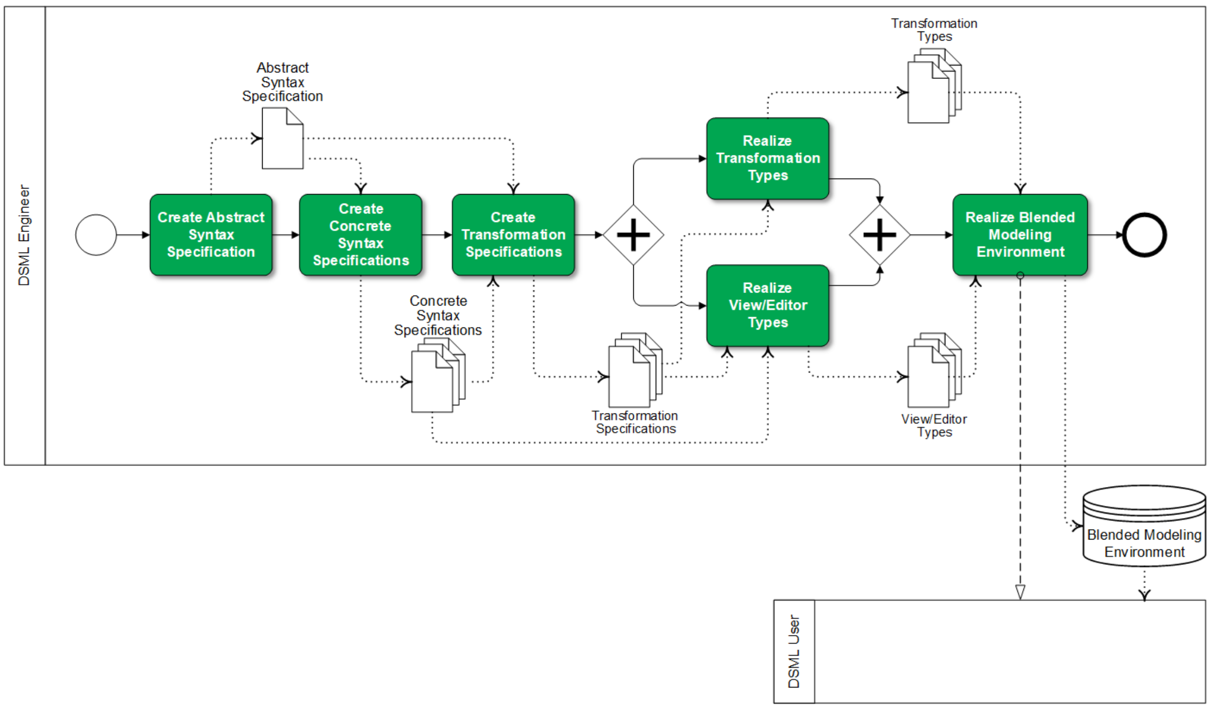
DSML Engineer
Abstract Syntax Specification
Create Abstract Syntax Specification
Create Concrete Syntax Specifications
Create Transformation Specifications
Concrete Syntax Specifications
Transformation Specifications
Transformation Types
Realize Transformation Types
Realize View/Editor Types
View/Editor Types
Realize Blended Modeling Environment
Blended Modeling Environment
DSML User

Figure 4. Workflow for Creating a Blended Modeling Environment

We took the liberty to simplify the process based on the dependencies between the different in- and outputs of the process-steps. The process in practice has a more iterative nature where changes to the specifications can be done in another order where temporal inconsistencies can occur. Hence, as for any model this process model is an abstraction of reality.

The workflow of the DSML user is kept intentionally blank. This is because this process is very much dependent on the particular DSML involved, and therefore cannot be addressed in this technology and DSML independent document.

Figure 5 depicts the workflow of modeling language evolution, which is not much different than the workflow already presented. In fact, it can be argued that an initial DSML development is not different from evolving an existing DSML, where the old DSML is just an empty one. The main difference is that when evolving a DSML the existing models that are conformant to the old DSML can become nonconformant to the new DSML. These models need to be migrated to become conformant to the new version of the DSML. In other words: The models need to be evolved to become conformant with the evolved DSML. That is why the DSML engineer creates a migration specification and realizes a transformation type that migrates the models of the DSML user.

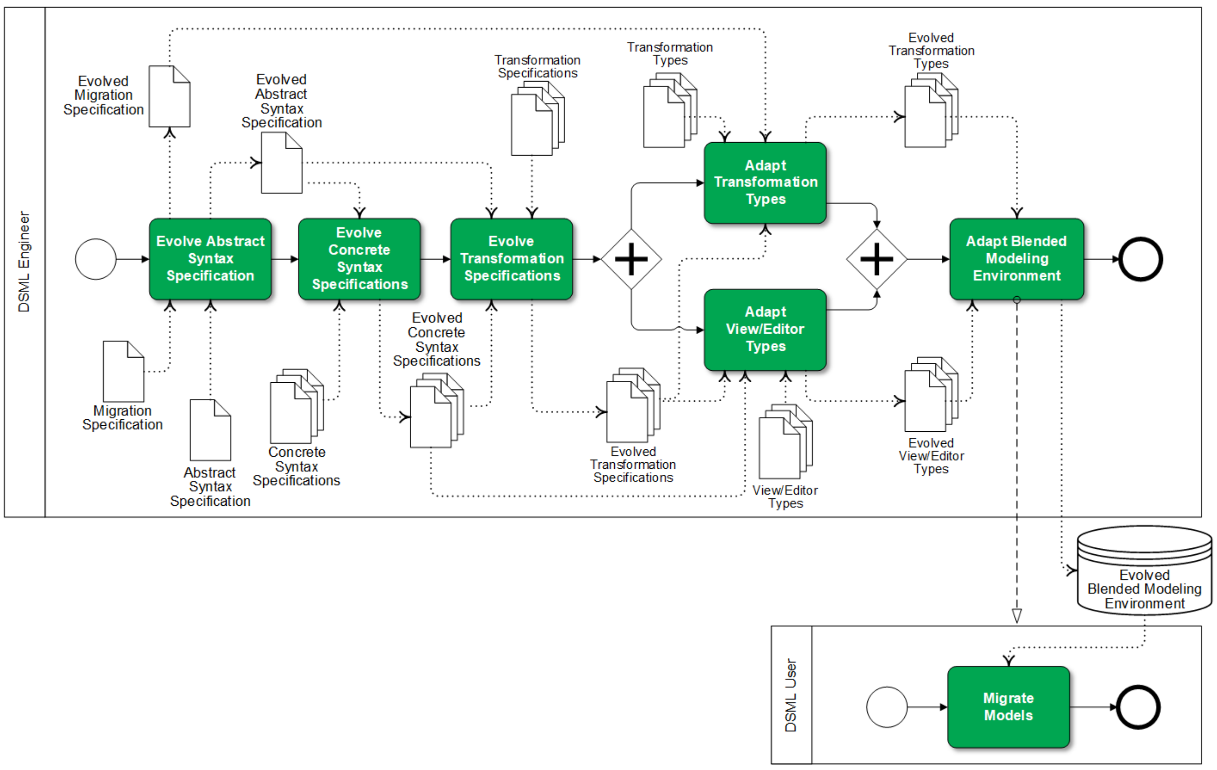
DSML Engineer
Evolved Migration Specification
Evolved Abstract Syntax Specification
Transformation Specifications
Transformation Types
Evolved Transformation Types
Evolve Abstract Syntax Specification
Evolve Concrete Syntax Specifications
Evolve Transformation Specifications
Adapt Transformation Types
Adapt View/Editor Types
Adapt Blended Modeling Environment
Migration Specification
Abstract Syntax Specification
Concrete Syntax Specifications
Evolved Concrete Syntax Specifications
Evolved Transformation Specifications
View/Editor Types
Evolved View/Editor Types
Evolved Blended Modeling Environment
DSML User
Migrate Models

Figure 5. Workflow for Evolving a Blended Modeling Environment

5. Conclusion

We can conclude that it is possible to define the generic BUMBLE concepts in a manner that is independent from the BUMBLE technology spaces Eclipse and MPS. We have also clarified that a complete BUMBLE solution can be created by combining relatively independent ‘partial’ solutions for different aspects.

We also defined the BUMBLE methodology by means of a process/workflow diagram. The process definition addresses mainly the workflow of the DSML Engineer. The workflow of the DSML user is not defined in detail because it is specific to the particular used DSML and modeling tool.