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 D2.2 (Version 1)

## BUMBLE Requirements Specification

Edited by: BUMBLE Team
Date: June 2021

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

# Contents

## Acronyms

| | |
|---|---|
| AD | Microsoft Azure Active Directory |
| API | Application Programming Interface |
| B | Blended Syntaxes & Modeling |
| BPM4DCA | Business Process Management for Debt Collector Advocates |
| C | Collaborative Modeling |
| CAD | Computer Aided Design |
| CAE | Computer Aided Engineering |
| CR | Change Request |
| CRUD | Create, Read, Update, Delete |
| DCA | Debt Collector Advocates |
| DSL | Domain-Specific Language |
| DSML | Domain-Specific Modeling Language |
| E | Evolution |
| ECU | Electronic Control Unit |
| ELK | Eclipse Layout Kernel |
| EMF | Eclipse Modelling Framework |
| EN | European Norm |
| GLSP | Graphical Language Server Platform |
| GUI | Graphical User Interface |
| IDE | Integrated Development Environment |
| JSON | JavaScript Object Notation |
| JVM | Java Virtual Machine |
| LDAP | Lightweight Directory Access Protocol |
| LSP | Language Server Protocol |

| | |
|---|---|
| ME | Modeling Environment |
| MOF | Meta-Object Facility |
| MPS | Meta-Programming System |
| N | Model Non-Conformance |
| OAUTH2 | Open Authentication Protocol Version 2.0 |
| PLM | Product Life-Cycle Management |
| RAfEBM | Reactive Architecture for Editing Blended Models |
| SSSD | System Security Services Daemon |
| T | Traceability |
| UC | Use Case |
| UI | User Interface |
| UML | Unified Modelling Language |
| UML-RT | UML Real-Time |
| VCS | Version Control System |
| VS | Microsoft Visual Studio |
| WP | Work Package |
| XMI | Metadata Interchange |
| XML | Extendable Markup Language |

# 1. Introduction

This document describes requirements for blended collaborative modeling as identified for the use cases of BUMBLE, see Deliverable D2.1. The purpose of these requirements is to clarify what functionality is to be developed as part of the BUMBLE technologies in work packages WP3, WP4 and WP5. Hence, requirements that are specific to a concrete DSML without being relevant to the development of the BUMBLE technologies are not listed in this Deliverable D2.2.

This deliverable D2.2 has three versions at M20, M30 and M36. This first version provides an initial inventory of functional and non-functional requirements based on the use case descriptions in Deliverable D2.1. In addition, an initial identification of requirements common to multiple use cases is made to enable WP3, WP4 and WP5 to focus on during the M20-M30 period.

Recalling from Deliverable D2.1, BUMBLE identifies two kinds of use cases:

- System/software specification (S): use cases about system and software engineering
- Testing (T): use cases concerning automation of test activities

Table 1 gives an overview of the 12 use cases in BUMBLE. Use case UC1 is a public use case by all academic partners in BUMBLE, while the other use cases are by industrial partners.

*Table 1. BUMBLE Use Cases*

| Use Case | Description | Lead Partner |
|---|---|---|
| UC1 (S) | Software Open-Source Blended Modeling | MDH |
| UC2 (S) | Combined Textual and Graphical Modeling of State Machines in HCL RTist | HCL |
| UC3 (S) | Vehicular Architectural Modeling in EAST-ADL | Volvo |
| UC4 (S) | Cross-Disciplinary Coupling of Models | Canon |
| UC5 (S) | Reactive and Incremental Transformations across DSMLs | MVG |
| UC6 (S) | Blended Editing and Consistency Checking of SysML Models and Related Program Code | Saab |
| UC7 (S) | Multi- and Cross-Disciplinary Modeling Workbench | Sioux |
| UC8 (S) | Model-Driven Development of Workflow Models for Debt Collecting Advocacy | Hermes |
| UC9 (S) | Automated Design Rule Verification on Vehicle Models | Ford |
| UC10[1] (S) | Development Process of Low-Level Software | Unibap |

[1] Change Request CR3 describes two use cases of Ford. These have been merged into a single use case (UC9). Identifier UC10 is assigned to a use case of Unibap, which was not yet in CR3.

| Use Case | Description | Lead Partner |
|---|---|---|
| UC11 (T) | Multi-Aspect Modeling for Highly Configurable Automotive Test Beds Ready for Smart Engineering Demands | AVL |
| UC12 (T) | Agile V-model System Architecture | Pictor |

## 1.1. Classification of Requirements

For the complex functionalities that are to be realized by BUMBLE technologies, the requirements reflect a multitude of aspects. It is therefore desirable to enable classifying requirements and to identify for which key technologies a requirement is relevant. Hence, BUMBLE partners concluded to introduce a light-weight classification for requirements to ease such identification. To describe this classification, it is relevant to first identify the two types of users that BUMBLE addresses:

**Definition 1: DSML User** A DSML user or end-user is someone who exploits a DSML to create concrete models for a specific product context. To illustrate this, recall that the BUMBLE use cases cover various DSMLs to describe state machines. A DSML user expresses a specific state machine in one of those DSMLs and by doing so, this DSML user exploits the DSML tooling as realized by (a) DSML developer(s).

**Definition 2: DSML Developer** A DSML developer is someone who applies the BUMBLE technologies to create and implement DSML definitions (including any facilities such as editors, parsers, generations, etc. that come with a defining and implementing a DSML definition). Considering the example of state machines, this covers for instance the approach to capture a Mealy or Moore based state machine language (e.g., grammar) in some data structure (abstract syntax tree) and providing editors (e.g., textual and/or graphical), parsers and generators etc. A DSML developer exploits the BUMBLE technologies as a basis for implementing DSML definitions.

BUMBLE identifies two levels of requirements as follows:

**Definition 3: Core Requirement** A core requirement describes a key principle that is to be realized. This is expressed from the perspective of a user of the BUMBLE technologies. This means that the requirement is (mostly) independent of a solution approach or solution technology. Core requirements can be relevant for a DSML user or a DSML developer (or both) and link directly to realizing added value in the context in which the DSML user and/or DSL developer applies the BUMBLE technologies.

**Definition 4: Technical Requirement** Technical requirements detail core requirements from the perspective of the solution approach or solution technology perspective. Technical requirements primarily address needs of DSML developers, thereby also considering the technical context in which DSML definitions are to be created.
An example core requirement is the ability to support multiple syntaxes (e.g., editable textual and graphical representations of a state machine DSML definition) while the choice for Eclipse or Jetbrains MPS as base solution technology implies different detailing into technical requirements.

Since technical requirements detail core requirements, there should (eventually) be at least one technical requirement referring to each core requirement. We expect this to prevail more explicitly during the BUMBLE project with each updated version of this Deliverable D2.2 at M36 and M42.

Next to the distinction in two levels of details, it was concluded to tag both core and technical requirements with the following 5 key aspects that BUMBLE will address (a shortcut letter is used later on in this deliverable). Note that a requirement can be relevant for multiple key aspects:

- **Blended Syntaxes & Modeling (B)** This includes any aspect related to synchronization / transformation between multiple representations (syntaxes).
- **Collaborative Modeling (C)** This covers any ability to cooperate in using the same DSML model instances or DSML definitions between multiple DSML users and/or DSML developers.
- **Evolution (E)** This can be related to evolution of DSML definitions (User is a DSML Developer) or evolution of instances of a DSML (User is a DSML User) or both.
- **Traceability (T)** This aspect covers both traceability within a (set of) DSML model instances or definitions at a given instance in time but also traceability in the context of their evolution.
- **Model Non-Conformance (N)** This refers to the ability of a DSML user to have syntactical elements that are not (yet) part of a model instance. An example is a partially typed text that is to be parsed to enable a mapping of the text onto the appropriate elements of a DSML.

Chapter 14 presents an initial selection of (core and technical) requirements common to multiple use cases that will be addressed in BUMBLE. For such common requirements, possible links to open-source project(s) may be indicated (if applicable). This gives a wider contextual scope than just BUMBLE in view of potential discussions on further clarifying requirements and/or publishing the BUMBLE technologies that provide a solution to such a requirement.

## 1.2. Deliverable Status

There will be three versions of this deliverable in accordance with the BUMBLE project plan. For this is the first version, a preliminary list of requirements has been identified for most use cases. UC9 (Ford) has not provided requirements yet. Focus has mostly been on identifying core requirements, while technical requirements are gradually emerging from the technological choices that are being made. Hence, identification of selected common requirements in Chapter 14 also concentrates on core requirements with a preliminary list of common technical requirements

# 2. UC1 - Software Open-Source Blended Modeling

This use case covers a public show case for the BUMBLE technologies. Starting from a MOF-based DSML, the BUMBLE framework is expected to provide the possibility to generate at least two model specific notations, one graphical and one textual, and related editors. In addition, the BUMBLE framework will need to support model synchronization mappings between the DSML and the generated notations. Given the DSML, the generated notations, and the model synchronization mappings, the framework is expected to semi-automatically generate synchronization mechanisms between notations and DSML and co-evolution transformations. In addition, the framework should provide an API to access the elements of the abstract syntax tree in order to enable traceability to model elements independent of the concrete notation in which the model is edited.

Given the DSML and its corresponding editor(s), the framework provides a collaboration mechanism that allows multiple users to collaboratively edit the models in real-time. The collaboration mechanism is independent of the number of users collaborating on the models at a given moment in time and supports remotely distributed users. In addition to real-time editing, the collaboration mechanism also keeps track of different versions of the edited models via a set of Git-like diff/merging functionalities.

## 2.1. Core Requirements

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C1.1 | B, T | The framework shall allow to describe mappings between a DSML specification (metamodel) and a notation of choice. |
| C1.2 | B | Given the DSML specification and the mappings to the notation of choice, the framework shall semi-automatically generate notation-specific specification (e.g., grammar) and related editing features. |
| C1.3 | B, C, T | Given the DSML specification and the mappings to the notation of choice and the notation-specific specification (e.g., grammar), the framework shall semi-automatically generate synchronization mechanisms (model transformations) to keep generated notation and DSML in sync. |
| C1.4 | B, C | The framework shall allow change propagation across notations and synchronization both on-demand or on-the-fly, upon user's choice. |
| C1.5 | C | The framework shall allow a model to be viewed and edited in real-time in a collaborative fashion by multiple users. |
| C1.6 | C, T | The framework shall allow to version models and apply diff/merge features, in a GIT-based fashion. |

## 2.2. Technical Requirements

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| T1.1 | B, C, T | The framework shall be implemented in the Eclipse ecosystem. | C1.1, C1.2, C1.3, C1.4, C1.5, C1.6 |
| T1.2 | B, C, T | The framework shall support MOF-based DSMLs. | C1.1, C1.2, C1.3, C1.4, C1.5, C1.6 |

# 3. UC2 - Combined Textual and Graphical Modeling of State Machines in HCL RTist

Users of HCL RTist will be able to use a textual notation for creating, viewing, and editing state machines, as an alternative to the current graphical notation. The textual notation should use a syntax that is easy to learn and use. The Eclipse editor that implements the syntax will support common features such as content assist, navigation etc. These commands will take the semantic context of the state machine into consideration to provide accurate and relevant results. When editing a state machine in one notation, information present in other notations will be preserved to an as large extent possible.

## 3.1. Core Requirements

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C2.1 | B | The textual state machine notation should cover all aspects of UML-RT state machines. That is, it should be possible to fully define a state machine textually without using any other notation or view. |
| C2.2 | B | Changes in the state machine model should update the textual notation without losing non-semantic information it contains, such as comments, indentations, and other white-space characters. The “layout” of the code as chosen by the user when typing the text should hence be preserved. |
| C2.3 | B, E, T | Changing a state machine in the textual notation should update the semantic model in a way that preserves the identity of the model elements. For example, incoming references to the model elements in the state machine should not become broken unless the target element was deleted or renamed. |
| C2.4 | B, T | References in the textual state machine notation that refers to model elements defined outside the state machine should be bound to the correct model element, using the capsule that owns the state machine as the context for reference resolution. |
| C2.5 | B, T | Similar to C2.4 Content Assist (a.k.a. “code completion”) for references should utilize the capsule that owns the state machine as the context for finding valid target objects for the references. |
| C2.6 | B | Semantic checks (a.k.a. validation rules) should be implemented which detects semantically incorrect models which the textual syntax permits creating. An example is creation of an internal transition at state machine level. |
| C2.7 | B | Changes in the state machine model should update the graphical notation without losing non-semantic information it contains, such as colors, symbol sizes, line routing and other layout information. |

## 3.2. Technical Requirements

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| T2.1 | B | The Eclipse editor that implements the new state machine syntax should provide content assist and navigation that utilizes the semantic context of the state machine. | C2.4 |
| T2.2 | B | A textually defined state machine is persisted using the textual syntax, while other parts of the model are persisted as XMI. The textual state machine files should be EMF fragments from a resource loading point of view. | C2.2 |

# 4. UC3 - Vehicular Architectural Modeling in EAST-ADL

Development of automotive embedded systems at Volvo involves large amounts of data from multiple stakeholders. To organize this data efficiently and ensure that syntax and semantics of the content are consistent, a metamodel is required.

Autosar and EAST-ADL are architecture description languages for automotive embedded systems, covering complementary aspects of software, electronics, and mechatronics. Use Case 3 is about providing multi-mode editing and viewing capabilities for such models, as well as metamodel evolution support, with focus on EAST-ADL.

## 4.1. Core Requirements

### 4.1.1. Editors

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C3.1 | B | It should be possible to split the information in one model into different files. The package structure uniquely identifies the elements in an EAST-ADL model. The elements themselves can reside in separate files. The persistence layer the editors are based on resolves these references automatically in the memory representation of the model without exposing the concrete file decomposition to the user. |
| C3.2 | B | Information should be possible to subset according to different model aspects. A particular editor or editor view may address only a subset of the model. According to:<br>• **Package Containment** Edit only elements in the selected package or its subpackages.<br>• **Element Kind** Edit only elements of a certain kind or set of kinds, e.g. related to a package of the metamodel related to, e.g., variability, timing, behavior.<br>• **Element Criteria** Edit only elements that fulfil a selected set of criteria, e.g., allocated to a certain ECU, realizing a certain feature, part of a certain variability configuration, active in a certain mode, etc. In adding elements in such a view, the model will be updated such that the new element complies with the criterion. For example, the new element may be allocated to the ECU, realize the Feature, be part of the variant, etc. |
| C3.3 | B | Shared information relevant only to specific editors (graphical, textual, tree-based) should be stored separately from the model itself:<br>• Graphical information such as colors and positions should be stored in a separate file; the graphical editor aspects shall be separated. This information needs to be updated if the model is edited in a different representation.<br>• Meta information needed by a textual editor shall also be separated. |

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C3.4 | N | It should be possible to create models in the editor that do not fully conform to the meta-model in order to ensure rapid prototyping and evolution of content. |
| C3.5 | N | It should be possible to integrate automated semantic checks into the editors to inform the user about inconsistencies of the model, e.g., with respect to the meta-model or the semantics. |

### 4.1.2. (De-)Serialization

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C3.6 | B | The order of elements in the *eaxml* file should be preserved on deserialization. New elements should be added according to the order in the tree or textual representation on serialization. New elements added in the graphical representation should be added at the end of the list of existing elements in the respective package. The order of existing elements should be maintained in the serialization. |

### 4.1.3. Tree-Based Editing

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C3.7 | B | The tree-based editor shows all elements of a model using the metamodel element hierarchy in packageable elements to structure the information. |
| C3.8 | B | Views shall be possible to define based on information subsetting, i.e. only a subset of model content is exposed according to criteria defined by the user or pre-defined by the editor (e.g., to only show elements in a specific package of the meta-model such as timing or variability). |
| C3.9 | B | The order of elements in the underlying model can be changed by dragging elements into a different order in an unsorted view. |
| C3.10 | B | It should be possible to sort the information in the tree by either the meta-class type, or in alphabetical order of the short name of the element, or in the order in which they are stored in the underlying *eaxml* file. View sorting does not affect the underlying order of elements in the model. |

### 4.1.4. Textual Modeling

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C3.11 | B | A text editor will typically operate on a subset of the model. Declarations in the text are probably required to define which packages are available |

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| | | to the package for anything added. For example, packages with datatypes or other elements may be imported and subsequently visible and part of the scope. |

### 4.1.5. Graphical Modeling

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C3.12 | B | A diagram will concern a subset of the model. This subset will be defined by the user and needs to be stored for later retrieval. The elements shown in the diagram are based on a query. This query can select elements that are in a Parent/child relation (e.g., elements in the same package or function decomposition), in a reference relation (relations implemented as association classes in EAST-ADL, e.g., allocations [e.g., elements that are allocated to a certain ECU], realizations; alternatively relations as references with a role name from a safety case to other elements), or of the same meta-class (e.g., all requirements). |
| C3.13 | B | Diagrams depicting a parent/child relation can be instantiated from any editor by invoking an action on the parent element (e.g., on a package). If no parent element is selected, a dialog allowing to select a package should be shown. |
| C3.14 | B | It is also necessary to define the context for new elements that are added to the model in the graphical view. This context defines where in the package hierarchy new elements are stored and how they are woven with existing elements, e.g., realizing a specific feature or allocating to a specific ECU. The context can be derived from the query that defines the diagram, since that query contains the type of relationship that is being shown in the diagram. |
| C3.15 | B | Deleting anything in a diagram is primarily about deleting from the diagram canvas. If an element shall also be deleted from the model, it must be done explicitly, e.g., by right clicking or ctrl-deleting. This is because a user may want to customize the viewpoint and include/exclude elements depending on the purpose of the diagram. |
| C3.16 | B | It should be possible to model concepts in different ways. Containment could, e.g., be modelled using the black diamond composition relation or direct graphical containment (boxes within boxes). Both ways of modeling should be supported and might need to change the appearance of the elements (e.g., whether attributes are shown or not). It should be possible to switch between these alternatives easily. |
| C3.17 | B | It should be possible to have different diagram types that use a slightly different concrete graphical syntax and different editor capabilities. Timing diagrams can expose event chains, feature diagrams can show the variation points, structural diagrams show allocations, and specialized diagrams for the safety cases are also necessary. |

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C3.18 | B | The editor should support auto-layouting that automatically selects the diagram type and the kind of visualization (e.g., composition or containment), in particular when generating a new diagram from a different editor. Auto-layouting should be based on element types, i.e., keep elements of the same type together. |

### 4.1.6. Diffing and Merging

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C3.19 | C, E | There should be functionality for diffing and merging of EAST-ADL models to support collaborative modeling of different team members. |
| C3.20 | C, E | Diffing and merging should be performed based on the concrete elements of the model, i.e., based on the meta-model rather than on the structure of the file. This means that changes in the order of the underlying *eaxml* file should not be made visible to the user. |
| C3.21 | C, E | Visualizing and managing diff and merge should be possible in a graphical, textual, and tree-based view. It should be possible to see conflicts, added elements, and deleted elements. It should be possible to select the version to keep. |

### 4.1.7. Multi-User Support

Ideally, multi-user editing should be supported, even though these requirements have low priority.

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C3.22 | C | It should be possible to define access and editing rights for different stakeholders that are automatically enforced by the tooling in order to limit users' ability to see certain parts of the model or change certain parts of the model. |
| C3.23 | C, E | Two or more users should be able to concurrently edit the same model without the need for explicit commit and check-out operations. Changes performed by one user should automatically become visible to the other user. Editing conflicts should be dealt with using conflict resolution mechanisms (e.g., first come, first serve). |
| C3.24 | C, E | Even if multi-user concurrent editing is available, it should still be possible to diff and merge a model that has been modified offline with a model that has been concurrently edited in order to support engineers that have been working on the model without access to the concurrent editing environment. |

## 4.2. Technical Requirements

The overall solution shall be implemented by means of applying different forms of the Language Server Protocol (LSP). For each particular editor, a corresponding language server shall be provided. As a side effect, this will enable the application of the tool within browser-based IDEs as VS Code and Eclipse Theia.
The textual editor shall be implemented by applying the Xtext framework and its capability of exporting standalone LSP applications. The graphical editor shall be implemented based on the Eclipse Graphical Language Server Platform (GLSP). For enabling auto-layouting functions in the GLSP editors, the Eclipse Layout Kernel (ELK) shall be applied. For implementing the tree editor, some kind of JSON Forms in combination with LSP will be applied.

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| T3.1 | B | The usage of Xtext requires a grammar as a basis for the textual editors. This grammar shall be inferred from the EAST-ADL metamodel, which is already provided by Xtext out-of-the-box. However, such initially metamodel-inferred grammars are typically not amenable for end users, so that the language engineer in general adjusts the initially inferred grammar. This adjustment procedure must be automated as far as possible, especially since the EAST-ADL metamodel and thereby also the inferred grammar can evolve. | C3.1, C3.2, C3.3, C3.4, C3.5, C3.11 |
| T3.2 | B | The currently favored textual syntax applies whitespace indentation to define model element hierarchies and scopes. However, the typical Xtext-style grammars use brackets to define such hierarchies and scopes. Thus, corresponding adaptations shall be implemented to support a whitespace-aware textual language. This should also be considered in the automation after the initial grammar inference (cf. T3.1). | C3.1, C3.2, C3.3, C3.4, C3.5, C3.11 |
| T3.3 | B | On editing one particular text file, the textual editor shall support referencing and importing contents of other model parts, which requires adaptations on the initially generated Xtext application regarding scoping, linking, etc. | C3.11 |
| T3.4 | B | The GLSP approach requires configuring and implementing a client as well as a server side, distinguishing the notation-specific client rendering and the overall model management on the server side. | C3.1, C3.2, C3.3, C3.4, C3.5, C3.8, C3.12, C3.13, C3.14, C3.15, C3.16, C3.17, C3.18 |

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| | | Particularly, this includes two aspects. First, the model information that is relevant to certain views / diagrams / text editors must be identified and configured for the particular language clients and servers. Second, so-called handlers decide which information is notation-specific and/or relevant to the model. Thus, the particular handlers must be configured and implemented to separate notation-specific and model-relevant information and synchronize if the information is relevant for both domains.<br>This effortful procedure shall be conveniently guided for the language engineer for efficiency purposes (e.g., by means of documentation and/or automation). | |

# 5. UC4 - Cross-Disciplinary Coupling of Models

Canon Production Printing is aiming to increase printer modularity/variability and shortening product development lead time while maintaining high quality software for each configuration of a Product Family. The media handling software component requires tight coupling to information from CAD/CAE models specified in Siemens NX. Mismatches between the CAD/CAE model and the embedded software leads to errors and underperforming products.

Canon Production Printing wants to explore techniques to enable Collaborative Modeling for cross-disciplinary models. It should be possible to access the models easily, and switch between multiple notations (projections in MPS), as well as keeping the models, and their relationships, up to date with (almost) no effort from the modelers.

Currently, the threshold of using JetBrains MPS as a tool for collaborating in models is too high; (1) the default interface is heavily cluttered with tooling for language development, distracting from the model development, (2) keeping the models in sync and up-to-date is too complicated for non-daily users, (3) the tooling, including all DSML plugins requires multiple gigabytes of disk space.

## 5.1. Core Requirements

### 5.1.1. Modeling and Model Management

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C4.1 | C, T | DSML users can authenticate themselves to gain access to the model repository. |
| C4.2 | C, T | DSML users can navigate through (relationships between) the existing models via hyperlinks. |
| C4.3 | C | DSML users can manage (CRUD) a hierarchy/organization of models (folders/packages, as well as model roots), to achieve a maintainable organization of the modeling content. |
| C4.4 | C, T | DSML users can tag model versions, so that they can be used as snapshots for later reference. |
| C4.5 | C | Administrators, Qualified DSML Users/Owners can manage user accounts, user groups, and access levels for the modeling environment and modeling entities. |
| C4.6 | C | DSML users can generate/download/deploy modeling artifacts, so that modeling artifacts can be used outside of the modeling environment. |
| C4.7 | C | DSML users can start/perform analysis on a model, to check for the model for certain properties (correctness, performance, etc.). |
| C4.8 | B, C | DSML users can see errors and feedback (if any) inline in the model editor (when the erroneous model element is visible in the projection), so that they can quickly identify issues in the model. |

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C4.9 | B, C | DSML users can see an overview of errors and feedback (if any) in an overview per model editor (or model package), so that they can quickly identify issues in the project. |
| C4.10 | C, T | DSML users can follow a modeling reference (hyperlink) from the model editor, so that they can easily navigate the relationships between models. |

### 5.1.2. Blended Modeling

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C4.11 | B, C | DSML users can view and edit the models in different projections. |

### 5.1.3. Model Collaboration

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C4.12 | B, C | DSML users can see the current state of the model when they are connected to the modeling environment, so that they are always up to date. |
| C4.13 | B, C | Qualified DSML users can retrieve and export models from external sources to link information between systems. |
| C4.14 | B, C | DSML users can apply (free-form text) reviewing annotations to the model/model elements, so that they can review and track progress. |
| C4.15 | B, C, E | DSML users can use the mutation history of a mode to see the evolution of the model over time. |
| C4.16 | B, C, T | DSML users can use a notebook-style view on the models, so that they can mix the content with the description/documentation. |
| C4.17 | B, C, T | DSML users can perform undo actions inside a model, so that they can undo their own changes while other DSML users are performing non-conflicting changes elsewhere in the DSML model instance. |

### 5.1.4. Language Development

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C4.18 | B, C, E, T | DSML developers can deploy a new language version, so that the model users can make use of the new language features. |

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C4.19 | B, C, E, T | DSML developers can perform (automated) language migrations, so that the models become consistent with the new language. |

### 5.1.5. Integration

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C4.20 | B, C | DSML users can instantiate a template for new (related) models using a web-based wizard, so that creation of new models is low-effort. |
| C4.21 | B, C | DSML developers can create web-based wizards to create templates for models that have a default structure and sets required dependencies to the DSMLs, to enable the modeling user to instantiate new models. |
| C4.22 | C | DSML users can (incrementally) import (i.e., uploaded by users, or retrieved from a server) data from a Git or CAD/CAE repository, so that the external relationships can remain up to date. |
| C4.23 | B | DSML developers can connect an action (button-press, intention called) in the (web-based) front-end to a computation/analysis/transformation on the server, so that the model can be used for analysis/generation purposes. |
| C4.24 | B | DSML users can visualize (interactively, inline, or in an external window) the results of the modeling artifacts, to achieve a smooth integration between the specification and the visualization. |
| C4.25 | C | DSML users can use model editors within a larger application that defines the workflow of the modeling activity, so that it eases the creation/interaction with other components. |
| C4.26 | B, C | DSML developers can integrate model editors with web-based components, so that they can create simplified workflows. |

## 5.2. Technical Requirements

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| T4.1 | C, T | Users can authenticate him/herself by logging into the website through a username/password combination. | C4.1 |
| T4.2 | C, T | Users can authenticate themselves in MPS when connecting to the model repository. | C4.1 |

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| T4.3 | C, T | Users can authenticate themselves through LDAP and OAUTH2 services. | C4.1, C4.5 |
| T4.4 | B, C | Users can access the model repository through a website, and through a connection with JetBrains MPS | C4.2 |
| T4.5 | C | Qualified DSML Users/Owners can set access levels for models and model packages, so that these models and model packages are visible/readable/writable by a particular set of users/groups. | C4.5 |
| T4.6 | C | Qualified DSML Users/Owners can set access levels to models for each defined role. | C4.5 |
| T4.7 | C | Administrators can define roles such that Qualified DSML Users/Owners can assign/remove 'Ordinary DSML Users' to such roles. Administrators can overrule Qualified DSML Users/Owners. | C4.5 |
| T4.8 | C | Qualified DSML Users/Owners can create groups of users. | C4.5 |
| T4.9 | C | Users can navigate and search (by name, tag, package, project, owner) the model repository to find a model. | C4.3 |
| T4.10 | C, T | Users can tag models (textual tags) and model versions (Git tag) for later reference. | C4.4 |
| T4.11 | C, T | Users can generate and download/transport/deploy the artifacts to a defined location (local PC, Git, Windows Share). | C4.6 |
| T4.12 | C | Users can start (predefined) external tools (simulators, visualizations) from the modeling environment. | C4.7 |
| T4.13 | B, C | DSML users can see errors and feedback (if any) inline in the model editor, so that they can quickly identify issues in the model. | C4.8 |
| T4.14 | B, C | DSML users can see an overview of errors and feedback (if any) in an overview per model editor (or model package), so that they can quickly identify issues in the project. | C4.9 |

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| T4.15 | B, C, T | DSML users can follow an error in the error overview to the location where the error is reported. If the project does not show the element, the model is selected. | C4.9, C4.10 |
| T4.16 | C, T | DSML users can follow a modeling reference (hyperlink) from the model editor, so that they can easily navigate the relationships between models. | C4.10 |
| T4.17 | B, C | DSML users can view and edit the model through their individually selected projection, so that they can simplify/extend the information shown in the model based on their needs. | C4.11 |
| T4.18 | B, C | DSML users can see their model in multiple (at least two) views, with different projections, so that they can focus on the structure and particular details at the same time. | C4.11 |
| T4.19 | B | DSML users can use textual syntax (with highlighting, completion, cross-referencing) within a graphical (diagrammatic/tabular) model. | C4.11 |
| T4.20 | B | DSML developers can set the default view of a model (entity) to a particular projection, so that they can simplify/extend the information shown in the model based on their needs (i.e., DSML developers can provide a default projection for DSML users as a starting situation). | C4.11 |
| T4.21 | B, C | DSML users can edit a model in each editable view (text, tables, diagrams, and forms), so that they have the freedom to choose the most effective representation. | C4.11 |
| T4.22 | B, C | Views are automatically updated upon editing by other DSML users. | C4.12 |
| T4.23 | B, C | DSML users can see collaborative feedback, like the current selection or cursor location of other users. | C4.12 |
| T4.24 | B, C | DSML users can see which users have the model open, to improve communication and avoid modeling conflicts. | C4.12 |

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| T4.25 | C, E, T | Qualified DSML users can (incrementally) import from external sources (like Git or CAD/CAE/PLM). | C4.13 |
| T4.26 | C, E, T | DSML Developers can specify incremental import strategies for model types. | C4.13 |
| T4.27 | C, E, T | DSML users can resolve merge conflicts, so that the models remain in a consistent state. | C4.13 |
| T4.28 | B, C | DSML users can apply (free-form text) reviewing annotations to the model/model elements, so that they can review and track progress. | C4.14 |
| T4.29 | B, C, E | DSML users can use the mutation history of a model (by the DSML user him/her-self as well as by other DSML users), so that the differences over time can be viewed. | C4.15 |
| T4.30 | C, E, T | DSML users can select model versions in the mutation history, so that they can compare the current model to the old model in any selected projection (i.e., any of the available syntaxes). | C4.15 |
| T4.31 | B, C, T | DSML users can use a notebook-style view on the models, so that they can mix (references of) the model content with the description/documentation. | C4.16 |
| T4.32 | B, C, T | DSML users can perform undo actions inside a model, so that they can undo their own changes while other DSML users are performing non-conflicting changes elsewhere in the DSML model instance. | C4.17 |

# 6. UC5 - Reactive and Incremental Transformations across DSMLs

The use case combines collaborative and blending modeling of two different state-transition modeling-languages that are transformed and synchronized instantly. The modelers (the users in the uses-case) can change their models in different network locations and can view and edit their models in their own preferred syntax, yet still be able to edit the models together. Changes made by one user are immediately visible by other users. The use case is based on a combination of highly desired functionality by customers of the Modeling Value Group, and essential features relevant for most of the BUMBLE partners.

The two models can both be changed independently and synchronized later-on, or immediately synchronize when either models are changed. Furthermore, the two models are not wired together persistently, the transformation will match the models only when synchronized and only change models when needed.

The use case blends two languages that are both languages for defining state-machines. State-machines are well understood by most of the BUMBLE participants. One of the two languages will have state-transformations that are children of the source-states (referring to the target state), the other language will have state-transformations that are children of the state-machine itself (hence peers from the states, and referring to the source and target states). This use case will therefore contain a non-trivial (bidirectional) language-transformation.

## 6.1. Core Requirements

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C5.1 | B | Bi-directional immediate transformation and synchronization for the DSML user. |
| C5.2 | B | Non-trivial bi-directional transformation of abstract syntaxes. |
| C5.3 | C | Remote synchronization across different modeling clients for the DSML user. |
| C5.4 | E | Combining immediate and deferred synchronization by activating and deactivating immediate synchronization and updating models via a VCS when not synchronizing. |
| C5.5 | B | Easy specification of non-trivial bid-directional transformations by the DSML developer. |

## 6.2. Technical Requirements

All models and meta-models will be viewed and maintained using MPS. The DclareForMPS engine will take care of the immediate (reactive) and incremental synchronization and transformation of the models. The delta's broadcast server (part of Dclare) will be used to exchange mutations across different synchronized MPS clients. The MPS GIT integration will be used to synchronize and or

merge defferedly. The rule definition aspect of DclareForMPS will be used to define the (bi-directional) transformation between the two abstract syntaxes.

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| T5.1 | B | Persistent models in MPS must keep unchanged when they are transformed and already consistent translations according to the transformation definition (incrementality). That implies that the node-identities within MPS are also unchanged so that external references are kept valid. | C5.1 |
| T5.2 | C | The part (MPS models) that is remotely and immediately synchronized is chosen using a dialogue integrated within MPS. | C5.3 |
| T5.3 | C | The combination of immediate and deferred communication will be done consistently with (and therefore using) the GIT integration of MPS. | C5.4 |
| T5.4 | B | Since the MPS is used for the editing of the models, also the definitions of the (bi-directional) transformations need to be done in a language that fits the ecosystem of MPS. Preferably by using the same syntax (base-language) for querying and manipulating models. | C5.5 |

# 7. UC6 - Blended Editing and Consistency Checking of SysML Models and Related Program Code

The development of large complex embedded systems at Saab involves many different models of different notation, such as code, SysML, matlab, unstructured data, etc. To handle this data efficiently and ensure that syntax and semantics of the content are consistent, a metamodel is required. The use case is about providing multi-mode editing and viewing capabilities for such models, as well as metamodel evolution support.

## 7.1. Core Requirements

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C6.1 | B, T | Bi-directional transformation and synchronization of models, including graphical traceability. |
| C6.2 | E, N | Model consistency validation with graphical notification of violations between code and related models. |
| C6.3 | C, E | Feedback changes between code and models, especially in the case of model validation violations. |

## 7.2. Technical Requirements

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| T6.1 | B, T, N | Possible to view several different models related to code in the same IDE, chosen from CLion or Eclipse. | C6.1, C6.2, C6.3 |
| T6.2 | N | Architectural model violations visible in the IDE. | C6.2 |
| T6.3 | B, E, T | Bidirectional Code to model traceability by graphical or textual links to related models. | C6.1, C6.2 |
| T6.4 | B, E | Collaborative feedback is visible in the IDE. | C6.3 |

# 8. UC7 - Multi- and Cross-Disciplinary Modeling Workbench

At Sioux, we intend to blend different but interconnected aspects of a system specification, some of which are expressed in graphical DSMLs of Supermodels and others in multi-notation DSMLs of MPS and hence, facilitating multi- and cross-disciplinary modeling. Within BUMBLE we aim at creating a blended modeling environment (ME) that combines the strengths of Supermodels and MPS. Here, blended refers to mixing different (but potentially interconnected) language instances on the same model. Live synchronization is expected between Supermodels and MPS views on the multi aspect system specification. Support version control (git, svn) collaboration is expected between multiple DSML users working on the same model. First iteration prototype is expected to visualize differences well enough between models using graphical DSMLs and resolve conflicting changes on DSML level.

## 8.1. Core Requirements

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C7.1 | B | The blended ME should consist of MPS and of Supermodels as optional front-end (both run on the same machine). DSML user perceives the blended ME as one environment. |
| C7.2 | B | DSML user can use DSMLs in Supermodels to edit (parts of) a model and/or can use other DSMLs in MPS to edit (other parts of) a model. |
| C7.3 | B | Supermodels and MPS editors should be synchronized (on-the-fly). |
| C7.4 | B | The blended ME should allow model checks to be triggered from both Supermodels and MPS (engages model checkers in both Supermodels and MPS). |
| C7.5 | B | The blended ME should allow generation to be triggered from both Supermodels and MPS (engages generators in both Supermodels and MPS). |
| C7.6 | C, E | The blended ME should allow for collaboration via file-based version control. |
| C7.7 | C, E | The blended ME should provide diff and merge functionality on DSML level from MPS (and optionally from Supermodels). |
| C7.8 | B, E | DSML developer can further develop the existing Supermodels DSMLs. |
| C7.9 | B, E | DSML developer can implement (new) DSMLs in MPS for which it can implement diagrammatic editors in Supermodels. |
| C7.10 | B | DSML user can open (or create new) and save (persist) a model from both Supermodels and MPS. |

## 8.2. Technical Requirements

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| T7.1 | B | The blended ME should provide interfacing technology suitable to bridge MPS (JVM) and Supermodels (.NET). | C7.1 |
| T7.2 | B | The blended ME should be started either by starting Supermodels or by starting MPS. Starting Supermodels should start the blended ME if it was not already started. | C7.1 |
| T7.3 | B | The blended ME should allow for flexible deployments depending on DSML user needs:<br>• only MPS (Supermodels is not deployed/started).<br>• only Supermodels (MPS running headless on the background).<br>• both Supermodels and MPS. | C7.1, C7.2 |
| T7.4 | B | Usability: Synchronization between MPS and Supermodels should happen fast enough to be perceived by the user as live updates (probably less than 0.5s). | C7.3 |
| T7.5 | C, E | The blended ME should provide models persistence mechanisms. At least file based should be supported among others. (To allow collaboration via file-based version control). | C7.1, C7.6 |
| T7.6 | B | The blended ME should provide interface between MPS and Supermodels for model checks and generation. | C7.4, C7.5 |
| T7.7 | B | Scalability: The blended ME should handle big models (50+K elements) while keeping the UI responsive enough. | C7.3, C7.4, C7.5 |
| T7.8 | C, E | Usability: visualize differences/conflicts between models of graphical DSMLs in a concise, readable, and clear way. | C7.1, C7.7 |
| T7.9 | C, E | The interfacing technology and interface architecture provided by the blended ME should be flexible enough to accommodate developing existing and new DSMLs. | C7.8, C7.9 |

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| T7.10 | B | The blended ME should provide interface from MPS to Supermodels for model files creation, opening, closing. | C7.10 |

# 9. UC8 - Model-Driven Development of Workflow Models for Debt Collecting Advocacy

HERMES İletisim's main job is creating digital solutions especially in Information Communication Technologies and Business Process Management area. HERMES provides Model Driven Engineering Solution for the development of Rule Based Workflow and Business Process Management Systems for various domains. Our aim is to design and implement a model-driven engineering platform to ensure Business Process Management for Debt-Collector Advocates, shortly called BPM4DCA. Debt Collector Advocates (DCA) usually cannot reach their customers/debtors by using a single way of communication like Phone Call, SMS, Voice Message or National ID SMS. Reaching a debtor, in fact, needs mostly reaching his/her guarantor, mother, father or other relatives in many different ways. Moreover, these debt collectors should deal with more than 10K case files in average which must be handled only in one month. Modeling with BPM4DCA consists of both various modeling viewpoints and the construction of relations required for managing the desired workflows. The blended modeling approach brought by the BUMBLE project will facilitate modeling and implementation of both choreography and orchestration of complex business services inside BPM4DCA.

## 9.1. Core Requirements

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C8.1 | C, T | Users can be authenticated to login to get access to a graphical and textual modeling environment. |
| C8.2 | B | Users can design their workflow by drag-and-dropping the elements in an editing environment. |
| C8.3 | C, T | Users can access their previously accessed models on a system. |
| C8.4 | B | Users can view and draw graphically their workflow's rules. |
| C8.5 | C, T | Administrator users can give different priorities to manage and control their access to the existing models. |
| C8.6 | T | Users can perform live tests for their workflows. |
| C8.7 | B | Users can be informed about the notifications and errors in the modeling environment. |
| C8.8 | B, C | Users can edit their defined rules to represent them in a graphical view simultaneously. |
| C8.9 | T | Users can store the models in a database to ease accessing them. |
| C8.10 | B, C, T | Users can fork new tasks by using the attributes of the current task and visualize relations of their workflows in a single diagram. |

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C8.11 | B | Users can modify the workflow in a textual and graphical editing environment with low code or no code. |

## 9.2. Technical Requirements

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| T8.1 | B | The workflow rules will be written in JsonLogic format. | C8.4 |
| T8.2 | B | The model will be generated in XML format to store in the database. | C8.9 |
| T8.3 | B | Design of the workflow will be performed inside BPMDCA's graphical modeling environment based on the MxGraph. | C8.2 |

# 10. UC9 - Automated Design Rule Verification on Vehicle Models

No input to this deliverable has yet been provided for this use case lead by Ford.

# 11. UC10 - Development Process of Low-Level Software

Unibap is a young tech company, with a high level of innovation and variation in our portfolio, and a wide range of skills and projects distributed among a relatively small number of employees. Our projects flow along a chain where each step involves different people, skills, and tools. This diversity introduces problems such as risk of miscommunication, difficulties in resource distribution, complicated documentation, and more. The possibility to collaborate on and automatically switch between representations of a model would both streamline our processes and eliminate several of the risk factors, which is of great importance in our development of safety critical components. In short, Bumble technology would support companies like Unibap in efficient utilization of valuable resources, as well as in ensuring high quality in our products.

## 11.1. Core Requirements

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C10.1 | B | The tool shall support creating a state machine as UML, and as XML. |
| C10.2 | B | The tool shall support display of a state machine at diagram level as UML, as XML, and as a state-event table. |
| C10.3 | B | The tool shall support editing of a state machine at diagram level as UML, as XML, and as a state event table. |
| C10.4 | B | The tool shall support addition and display of snippets of full C17 to a state machine state. |
| C10.5 | B | The tool shall support dependencies on internal and external libraries for the C17 snippets in a state machine state. |
| C10.6 | B | The tool should support nested includes for the C17 snippets in a state machine state. |
| C10.7 | B, N | The tool shall automatically convert between UML, XML, and state-event table representations of a state machine diagram on request, without loss of information that cannot be displayed in the current representation. |
| C10.8 | B, N | The GUI of the tool shall indicate when there is information that is not visible in the current representation. |
| C10.9 | T | The tool shall provide statistics of what states and branches have run during a test, and what C functions have been called from these. |
| C10.10 | B, C, T | The tool shall support version control, and diff and merge operations, on a state machine, with the possibility to see a graphical representation of the differences between versions. |
| C10.11 | B | The tool shall be able to generate a fully functional C code representation of a state machine for export on request. |

## 11.2. Technical Requirements

We are currently investigating HCL RTist as a potential match for our requirements. Depending on the result, we may present additional technical requirements in future versions of this document.

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| T10.1 | - | It must be possible to use the tool in a Linux environment. | - |

# 12. UC11 - Multi-Aspect Modeling for Highly Configurable Automotive Test Beds Ready for Smart Engineering Demands

At BUMBLE (and HybriDLUX), AVL wants to extend existing and new DSLs with the ideas of blended and collaborative modeling. With regard to collaborative modeling, two dimensions are of interest: One is about enhancing existing/new DSLs in terms of collaborative modeling within a dedicated user group/department (e.g. graphical model diff), while the second dimension is about collaborative modeling across departments. In order to somewhat concretize the DSLs applied in this context, the following three DSLs will be considered here:

- DSL A for measurement device specification (textual and graphical aspects) with database integration and code generation - related to department X.
- DSL B for measurement device integration test definition (textual and graphical aspects) - related to department Y. This DSL has links to DSL A regarding the reuse of the data sets there. Furthermore, DSL B is considered for test case generation.
- DSL C for the definition of step-by-step instructions (textual, graphical and 3D CAD aspects), applied in department Z. This DSL also has direct links to DSL A. Generated results of this DSL are interactive documentations (e.g. web-based) up to virtual and augmented reality applications.

Intra-departmental collaboration is most relevant for DSL A, while inter-departmental collaboration is relevant for DSL B and DSL C. Note that there is not a single physical source or data model for all DSLs. Instead, the DSLs are developed independently, but are actively linked for reuse of data and notification of changes (subject of improvements).

## 12.1. Core Requirements

AVL use cases (based on three DSLs) are usually based on a so-called driving DSL representation. As a DSL can have different representations (views, sometimes only read-only), the driving representation is the one the DSML user is mostly working with. It has thus higher demands regarding features or requirements like data consistency (e.g. to other data sources). A non-driving representation may be outdated for a while, e.g. if changes are done in the driving DSL. Related, a driving representation always must be available to the DSML user, even in invalid status (temporary violation of metamodel), while secondary one may be removed or is not accessible as long as an invalid status is the case.

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C11.1 | B | DSML User has one driving DSL representation (textual), but has one or more graphical representations, which shows aspect of the model (reduced information), this is not sufficiently expressed by the driving one (without addition further information). |
| C11.2 | B | Changes to an instance of a driving DSL are forwarded to the alternative representation either immediately or after some trigger event (e.g. model is in a valid state, user event). |

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C11.3 | B | Changes on the alternative representation (if write enabled) are forwarded to the driving DSL immediately. |
| C11.4 | B | Differences based on changes on the driving DSL is illustrated on the alternative representation (e.g. before-after-comparison). |
| C11.5 | B, T | DSL must have the possibility to reference to external elements either from other data sources or to related DSLs. |
| C11.6 | B, T | Alternative data sources should be visualized alongside DSLs (e.g. CAD models) to provide user-friendly input methods for the DSL (e.g. selecting a part referenced by the DSL). |
| C11.7 | C | Multiple users should be able to work on the same model (offline). Model merge/diff techniques should be applied to synchronize the content again (including graphical representations). |
| C11.8 | C | Cross department collaboration: Different DSLs should be loosely coupled by using references. Inconsistency should be indicated, if the linked element has changed and should cause a certain action (e.g. notification) by the user to ensure consistency again. |
| C11.9 | C | Cross department collaboration - extension to C11.8. Fully automatic consistency assurance is not required, but user support (e.g. quick fixes based on the change) are favored. |
| C11.10 | T | Traceability links between different DSL representations should support editing navigation (e.g. marking one element in one representation should highlight the related elements in the other representation). |
| C11.11 | T | Traceability links between different but related DSLs should be established to enable C11.8 and C11.9. |
| C11.12 | T | Traceability links between model and generated artefacts should be established to support backtracking. |
| C11.13 | T | If generated artefacts are executable, traceability links should enable life debugging (if useful including breakpoints). |
| C11.14 | T | If generated artefacts are executable and create a particular outcome, the outcome should be related to model elements (e.g. test execution and test reports). |
| C11.15 | E | If DSL definition evolves, users should automatically get an updated version of the models. |
| C11.16 | E | If models are updated during DSL evolution, changes should be indicated to the user. |

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C11.17 | E | If models cannot be updated automatically, invalid model elements should be indicated, and user should be supported in decision making (e.g. quick fixes). |
| C11.18 | C, E, T | If model editing or model evolution between different but related DSLs lead to inconsistencies, these inconsistencies should be visualized to support decision making in fixing the inconsistencies. |

## 12.2. Technical Requirements

This use case will be realized using the Reactive Architecture for Editing Blended Models (RAfEBM), see https://drive.google.com/drive/folders/16tNZeh9hgegYlp3g4mJdN3ghgMaSGGdt).

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| T11.1 | B | RAfEBM: Ensure Single-Source of truth (non-redundant model source). | C11.1, C11.2, C11.3, C11.4 |
| T11.2 | B | RAfEBM: Editable goal-oriented views with view-specific languages (metamodels). | C11.1, C11.2, C11.3, C11.4 |
| T11.3 | B | RAfEBM: Decouple model source from view-specific languages / representations. | C11.1, C11.2, C11.3, C11.4 |
| T11.4 | B | RAfEBM: Avoid bi-directional transformation and synchronization. | C11.1, C11.2, C11.3, C11.4 |
| T11.5 | B | RAfEBM: Implement only required edit operations (clear definition what need to be changed in model sources for a specific operation). | C11.1, C11.2, C11.3, C11.4 |

# 13. UC12 - Agile V-model System Architecture

The evolved modeling of physical in a digital can be called a digital twin. The digital twin concept has been introduced in several applications domains. In UC12, the application is structural engineering for infrastructure. The digital twin must be supported by blended modeling to support the user in the design and maintenance process of the target system. This use case of Pictor considers table-based specifications of geometry and characteristics of the members and hinges. The target system is a model of building structure with thousands of elements and hinges in different materials. For example, a bridge in concrete with steel cables. The targets have requirements according to standards with high safety for public transportation according to European standards. The European norms EN 1992: for concrete structures are updated with more advanced practices in civil engineering for building bridges in concrete and steel. The bridges are for roads and railway tracks. Civil engineering has high safety requirements for public safety norms. New civil engineering imposes new requirements on the software package and the same as the software package is to meet the requirement on correctness and safety of the calculation for structural analysis.

## 13.1. Core Requirements

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C12.1 | B, T | The framework shall allow to describe mappings between a DSML specification (metamodel) and a notation of choice. |
| C12.2 | B | Given the DSML specification and the mappings to the notation of choice, the framework shall semi-automatically generate notation-specific specification (e.g., grammar) and related editing features. |
| C12.3 | B, C, T | Given the DSML specification and the mappings to the notation of choice and the notation-specific specification (e.g., grammar), the framework shall semi-automatically generate synchronization mechanisms (model transformations) to keep generated notation and DSML in sync. |
| C12.4 | B, C | The framework shall allow change propagation across notations and synchronization both on-demand or on-the-fly, upon user's choice. |
| C12.5 | C | The framework shall allow a model to be viewed and edited in real-time in a collaborative fashion by multiple users. |
| C12.6 | C, T | The framework shall allow to version models and apply diff/merge features, in a GIT-based fashion. |
| C12.7 | C | DSML users can authenticate through a login on a website. |
| C12.8 | C | DSML users can navigate the existing models on a website, to find models with a low threshold. |
| C12.9 | C | DSML users can manage (CRUD) a hierarchy/organization of models (folders/packages, as well as model roots), to achieve a maintainable organization of the modeling content. |

| ID | Classification (B, C, E, T, N) | Description of Requirement |
|---|---|---|
| C12.10 | C | DSML users can tag model versions, so that they can be used as snapshots for later reference. |

## 13.2. Technical Requirements

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| T12.1 | B, C, T | The framework shall be implemented in the Eclipse ecosystem. | C12.1, C12.2, C12.3, C12.4, C12.5, C12.6 |
| T12.2 | B, C, T | The framework shall support MOF-based DSMLs. | C12.1, C12.2, C12.3, C12.4, C12.5, C12.6 |
| T12.3 | B, T | The graphical view must support viewing nodes, hinges, and members configurable for visible or hidden. | C12.1 |
| T12.4 | B, T | The graphical view must show lines for center of gravity, cables, and members configurable for visible or hidden. | C12.1 |
| T12.5 | B, T | The graphical view must show x, y, z coordinates at the point of interest for configurable visible or hidden. The x, y, z coordinates at the point or at the bottom of the window. | C12.1 |
| T12.6 | B, T | The graphical view must show loads for configurable visible or hidden. | C12.1 |

# 14. Selected Common Requirements

This chapter summarizes a selection of requirements that are common to multiple use cases and are therefore considered to be addressed by the BUMBLE technologies in a generic way. This approach allows reusing the BUMBLE technologies and hence, showing their generic applicability.

The goal of this chapter is to provide focus points to work packages WP3, WP4 and WP5 on developing BUMBLE technologies in terms of satisfying requirements common to multiple use cases. The BUMBLE project does not intend to develop one single set of coherent technological solutions that covers all listed requirements. This originates not only from supporting both the Eclipse and MPS based technology platforms, but to some extent also from conflicting contextual details of the use cases. Instead, the BUMBLE project will develop multiple coherent sets of technological solutions which each will address a (major) subset of the presented requirements. This also means that multiple BUMBLE technologies may exist to satisfy the same requirement. Hence, concrete usage of BUMBLE results still allows a choice to which collection of BUMBLE technologies suits a use case best.

Since contributions for most individual use cases have focused on core requirements, while technical requirements need further progress in making technological choices to enable clarifying them further, this chapter also focuses on core requirements in Section 14.1. It is expected that further core requirements may arise during the BUMBLE project and that the list of selected common technical requirements in Section 14.2 will therefore increase as well. Updates and extensions will be documented in subsequent versions of this deliverable in accordance with the BUMBLE project plan.

## 14.1. Core Requirements

Based on the core requirements for the use cases in the previous chapters, we recognize a number of subjects that allow further structuring requirements next to the classification in terms of Blended (B), Collaboration (C), Evolution (E), Traceability (T) and Non-Conformance (N). We therefore use a further structuring, among others deducted from those for use cases UC3 and UC4:

- Blended Modeling
- Real-Time Collaboration
- Model Non-Conformance
- Contextual Integration
- Model Life-Cycle Management

### 14.1.1. Blended Modeling

At the core of BUMBLE is the ability to exploit multiple syntaxes for a DSML. This introduces several requirements compared to what is supported by existing DSML technologies. Requirements for typical facilities provided by DSML technologies such as syntax highlighting, content assist, auto-completion, (while-you-type) model validation, warning/error notifications, and artefact generation are basically the same for the BUMBLE technologies and are therefore not listed here. Nevertheless, it is explicitly stated by all use cases that such traditional facilities should basically be agnostic to the specific set of supported concrete syntaxes for a DSML model definition. Here, we focus on requirements introduced by the novelty of requiring support for multiple syntaxes as identified by almost all use cases. Blended modeling particularly extends the facility of structuring models in multiple (possibly configurable or predefined) editors/views to enable supporting multiple

syntaxes for the same elements of a DSML model definition. For this, the following common core requirements are selected.

| ID | Classification (B, C, E, T, N) | Description of Requirement | Cover UC-Specific Core Requirement(s) |
|---|---|---|---|
| BC1 | B | It must be possible to define multiple concrete syntaxes / representations for a single DSML model definition, including relevant views or editors conforming to the concrete syntaxes / representations. | C1.1, C1.2, C2.1, C3.7, C3.11, C3.12,C3.16, C3.17, C4.11, C4.16, C7.1, C7.2, C8.11, C10.2, C10.3, C11.1, C12.1, C12.2 |
| BC2 | B, C | A DSML user must be able to select a preferred concrete syntax / representation for a DSML model instance. A DSML developer must define a default concrete syntax / representation. | C1.1, C2.1, C3.7, C4.11, C7.10, C8.11, C10.2, C11.1, C12.1, C12.2 |
| BC3 | B, C, E, T | In case multiple syntaxes exist for a (single element of a) DSML model definition, all concrete syntaxes / representations must be updated in accordance with any changes that have been performed by means of using one of those syntaxes. | C1.3, C1.4, C1.5, C2.2, C2.3, C2.7, C4.16, C4.17, C5.1, C5.2, C5.5, C6.1, C7.3, C10.7, C11.2, C11.3, C12.3, C12.4 |
| BC4 | B, C, E, T | In case multiple syntaxes exist for a (single element of a) DSML model definition, it must be possible that certain elements may not be relevant or visible in one or more specific abstract and concrete syntaxes. Semantics of (an element of) a DSML model definition that is considered in multiple abstract and concrete syntaxes must (be enforced to) be/remain the same. | C1.3, C1.2, C1.4, C1.5, C2.2, C2.4, C2.6, C2.7, C3.3, C3.8, C3.11, C3.12, C3.14, C4.11, C4.17, C5.1, C5.2, C5.3, C5.4, C5.5, C6.1, C6.2, C6.3, C7.2, C10.2, C10.3, C10.4, C10.7, C10.8, C11.1, C11.3, C12.3, C12.4 |

### 14.1.2. Real-Time Collaboration

Another novelty of BUMBLE is the ability to support real-time collaboration between multi DSML users that access the same (collection of) DSML model instance(s), although this is not a requirement for all use cases. Various use cases that do require real-time collaboration, refer to a web-based approach although not all use cases require or specify this. Collaboration at DSML development is not explicitly expressed for any of the use cases and therefore not considered in BUMBLE nor are there shared requirements in terms of real-time collaboration-specific facilities such as a chat capability, the possibility of free-form textual reviewing annotations or feedback on at which model element(s) other DSML users are editing/viewing at the same moment in time. These aspects of real-time collaboration may however receive more attention in subsequent versions of this deliverable when prototype implementations of the BUMBLE technologies allow for early feedback from actual DSML users.

| ID | Classification (B, C, E, T, N) | Description of Requirement | Cover UC-Specific Core Requirement(s) |
|---|---|---|---|
| BC5 | B, C, E | It should be possible to support real-time collaboration between multiple DSML users. This means that - independent of which concrete syntax the DSML users have chosen - changes by an individual DSML user are instantly visible to all other DSML users that have viewing/reading and/or editing/writing rights to the considered (collection of) DSML model instance(s). | C1.5, C3.23, C3.24, C4.12, C5.3, C7.3, C12.5 |

### 14.1.3. Model Non-Conformance

Only use case UC3 explicitly specifies requirements on the ability to have support for model non-conformance that do not relate to support for intermediate states of modifying a DSML model instance that does not conform to its DSML model definition (which is generally needed when relying on a parser-based approach). Since no other use case expresses requirements on model non-conformance, no shared common requirements on model non-conformance are identified.

### 14.1.4. Contextual Integration

Although DSML technologies allow for developing the next generation of modeling environments, these are generally to be integrated as part of a bigger context (which may not rely on any DSML technology). Since the BUMBLE technologies should not disable such capability, a common requirement is identified to capture this aspect even though only use cases UC4 and UC7 explicitly express a clear need for being integrable in a larger context. We do note that both UC4 and UC7 will rely on the MPS technology as a basis for realizing their BUMBLE-based DSML solutions.

| ID | Classification (B, C, E, T, N) | Description of Requirement | Cover UC-Specific Core Requirement(s) |
|---|---|---|---|
| BC6 | B, C | It should not be impossible to integrate BUMBLE-based DSML environments in larger (non DSML technology based) applications that enable (real-time or non real-time) collaboration between users of that larger application context. | C4.23, C4.26, C7.1 |

### 14.1.5. Model Life-Cycle Management

Model (life-cycle) management is a generic need in the context of model-based development tools. It plays a role as soon as one starts using modeling tools and models. It covers many aspects in itself and for BUMBLE, these are relevant for both DSML model definitions (different levels of meta models), including the definition of (multiple) concrete syntaxes and DSML model instances.

Core requirements identified for the different BUMBLE use cases are limited to the model (life-cycle) management aspects of version control, persistence, (automated) (co-)evolution, access control, and traceability. Such traceability covers both

- Referencing across elements of a single DSML or of multiple DSMLs at a specific moment in time, i.e., for a given specific status or version of (the collection of) DSML(s). This may also cover references between DSML elements and information related to the execution of underlying tooling (e.g., in the context of debuggers, analysis tools or generated code).
- Referencing or comparing DSML elements that have evolved over time, i.e., traceability in the context of version control.

**Version Control (Non Real-Time Collaboration)**
The majority of use cases describe a need for having support for file-based version control, which is a means to support non real-time collaboration between multiple DSML users of the same (collection of) DSML model instance(s) and between multiple DSML developers of the same (collection of) DSML model definition(s). Although GIT is mentioned in all these use cases as a concrete version control system that is to be supported, also SVN is mentioned as relevant, while one use case mentions PLM as well in this context.

| ID | Classification (B, C, E, T, N) | Description of Requirement | Cover UC-Specific Core Requirement(s) |
|---|---|---|---|
| BC7 | C, E, T | It must be possible to exploit file-based version control, including diff/merge and tagging functionalities for both DSML model definitions and instances. | C1.6, C3.19, C3.24, C4.4, C4.15, C4.22, C5.4, C7.6, C7.7, C10.10, C11.7, C12.6, C12.10 |
| BC8 | B, C, E | Diff/merge functionality should be available at the model level instead of the underlying persistence format, where a DSML user can perform a diff/merge in a concrete syntax of choice (e.g., textual or graphical). | C3.20, C3.21, C7.7, C10.10 |

**Persistence**
File-based persistence of DSML model definitions and instances is not only in view of the need to support file-based version control, but also to allow interaction with tools that are outside of the DSML context. In general, DSML technologies allow persistence or (de-)serialization by means of generators and parsers. While this is generally independent of the way a DSML model is dealt with in a DSML tool, there are use cases for which more or less a one-to-one relation is required between the structure of DSML model definitions and files persisting instances of those DSML model definitions. Other use cases do not require nor wish such a one-to-one relation but some different mapping. This aspect is therefore considered to be specific for the DSML context of such use case and hence, there are no common requirements selected. Nevertheless, the BUMBLE technologies must support these different approaches and can rely on the capability of existing DSML technologies to do so.

**(Automated) (Co-)Evolution**
Although not many use cases in the previous chapters explicitly address details of (co-)evolution of (collections of interrelated) DSML model definitions and instances, taking the ability to support such capabilities has an important impact on primary facilities to be realized by the BUMBLE technologies.

| ID | Classification (B, C, E, T, N) | Description of Requirement | Cover UC-Specific Core Requirement(s) |
|---|---|---|---|
| BC9 | E, T | It should be possible to deploy a new version of a DSML model definition by means of automatically migrating existing instances of that DSML model definition. In conjunction with that, cross-references to other DSML model definitions and instances must be migrated automatically. | C2.3, C2.4, C2.5, C4.19, C5.4, C11.15 |

**Access Control**

Several use cases require the ability to use access control, which is clearly related to the aspect of collaboration albeit with the additional need to limit freedom. Such limitations may refer to individual elements of a DSML or to complete DSMLs. Other use cases have not specified such need. The BUMBLE technologies should support both cases, i.e., with and without access control.

| ID | Classification (B, C, E, T, N) | Description of Requirement | Cover UC-Specific Core Requirement(s) |
|---|---|---|---|
| BC10 | C, E, T | In view of various CRUD functionalities, in particular related to collaboration and (traceability in the context of) evolution, it must be possible that DSML users can be identified by means of a(n) (single) authentication step (e.g., with a login) when accessing the modeling environment. | C4.1, C8.1, C12.7 |
| BC11 | C, E, T | It must be possible to define different levels of (CRUD) access for DSML users. In case multiple access levels exist, the BUMBLE framework should enforce conformance to such access levels based on the authentication step executed when DSML users access the modeling environment. | C4.1, C4.5, C3.22, C8.3 |
| BC12 | C, E, T | In case access control is used, then based on the level of (CRUD) access a DSML user has (possibly including different levels of administrator roles), (s)he must be able to modify the level of (CRUD) access for him/her-self or other DSML users. | C4.5, C8.5 |
| BC13 | C, E, T | By default, a DSML user must at least have full access rights to model elements that (s)he modified. In particular, while editing a DSML instance, a DSML user must at least be able to perform undo actions for modification that (s)he made and is (by default) not able to undo modifications performed by other DSML users. | C4.17 |

## 14.2. Technical Requirements

BUMBLE focusses on two DSML technology platforms as a starting point: Eclipse and MPS. We also consider interaction across these DSML technology platforms (including others given the concrete context in certain use cases). For each of these DSML technology platforms, different architectural and design choices can be made on how to realize the required BUMBLE functionalities. The various approaches are primarily to be documented in Deliverables D3.1, D3.2, and D3.3. In this section, we consider selected common technical requirements taking into account (and hence referring to) relevant DSML technology platform contexts.

### 14.2.1. Blended Modeling

At the moment of writing this version of this deliverable, the requirements for blended modeling are worked out more elaborately compared to the other requirements. Many of these requirements hold for both DSML technology platforms, while few are specific to a DSML technology platform choice. It is recognized that some of the technical requirements may individually already be satisfied by existing DSML technologies available for one or both DSML technology platforms. This allows BUMBLE to reuse and extend such existing technologies. It also helps in connecting to existing open-source communities for exploitation of the BUMBLE technological solutions. Notice that the combination of all requirements together is the core novelty of the solutions to be realized by the BUMBLE project.

**DSML Platform Independent Requirements**

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| BT1 | B | At least one editor/view (i.e., concrete syntax) must be generated automatically (on-the-fly or on demand) for a DSML model definition. | BC1, BC2 |
| BT2 | B | It must be possible to define (a) textual editor(s)/view(s) for (elements of) a DSML model definition. | BC1 |
| BT3 | B | It must be possible to define (a) graphical editor(s)/view(s) for (elements of) a DSML model definition. | BC1 |
| BT4 | B | It must be possible to define (a) form-based editor(s) (including tabular-like layouted forms) for a DSML model definition. | BC1 |
| BT5 | B, C | DSML users must be able to choose the editor/view (i.e., concrete syntax) to be used to edit/view (elements of) a DSML model instance. | BC2 |
| BT6 | B, C | DSML developers must be able to specify a default editor/view (i.e., concrete syntax) for (elements of) a DSML model instance that | BC2 |

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| | | is presented to a DSML user if (s)he has not (yet) made a choose for a preferred alternative editor/view (i.e., concrete syntax) (if alternative(s) would be available). | |
| BT7 | B, C, T | Cross-referencing between elements of the same or different DSML model instances must be agnostic to any specific syntax that a DSML user may have selected to edit/view such DSML model instance(s). | BC1, BC2 |
| BT8 | B, C, E, T | In case multiple syntaxes exist for a (single element of a) DSML model definition, DSML developers must be able to exploit an (easy-to-use) semi-automatic approach to generate synchronization and/or transformation mechanisms that operate at the level of the elements of the relevant DSML model definitions to update all concrete syntaxes / representations in accordance with any changes that may have been performed by using one of those syntaxes. This must enable at least one of the following capabilities:<br>• automated real-time (on-the-fly) synchronization/transformation.<br>• on-demand (i.e., based on an explicit request of a DSML user) synchronization/transformation.<br>next to also supporting:<br>• synchronization/transformation via file-based version control. | BC1, BC3, BC4, BC6, BC7, BC8 |
| BT9 | B, C, E | It must be possible to view errors/notifications on the results of DSML model instance validation in the editor/view for any concrete syntax that represents (elements of) the corresponding DSML model definition. Model validation is therefore to be realized at the level of (elements of) the relevant DSML model definitions while the interaction with the DSML user is to be performed via all of the available concrete syntaxes. | BC4 |
| BT10 | B, C, T | Errors/notifications on the results of DSML model instance validation must be provided with a reference to the relevant element(s) represented by any concrete syntax of that | BC4 |

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| | | DSML model instance and/or of other relevant DSML model instances causing the error/notification to be present. | |

**Eclipse Platform Specific Requirements**

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| BT11 | B | It must be possible to create/use MOF-based DSML model definitions. | BC1 |
| BT12 | B | It must be possible to define (a) Xtext-based textual editor(s)/view(s) for a DSML. | BC1 |
| BT13 | B | It must be possible to define (a) tree-based editor(s)/view(s) for (elements of) a DSML model definition. | BC1 |

### 14.2.2. Real-Time Collaboration

Real-time collaboration as considered by some use cases is assumed to be based on a server-client approach, where the server and the different clients may or may not exist at different computers. This allows for real-time collaboration between more traditional desktop application clients and a centralized server, as well as for real-time collaboration exploiting web-clients using traditional internet-browser technology and a centralized server. Both options are considered relevant in the BUMBLE context, although some use cases are specifically referring to the web-based approach.

**DSML Platform Independent Requirements**

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| BT14 | B, C, E, T | Changes of (elements of) DSML model instances by one DSML user must automatically become visible to all DSML users editing/viewing those (elements of) the DSML model instances in (near) real-time. | BC1, BC3, BC4, BC5 |

**Eclipse Platform Dependent Requirements**

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| BT15 | B, C, E, T, N | DSML developers should be able to realize real-time collaboration based on a LSP/GLSP-based approach using an | BC4, BC5 |

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| | | extension (i.e., by means of plugins) of the existing Eclipse IDE as desktop-client application.<br>Note: DSML developers may also use a different approach | |

**MPS Platform Dependent Requirements**

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| BT16 | B, C, E, T | DSML developers should be able to realize real-time collaboration based on using an MPS Model Server, where the options of using an extension (i.e., by means of plugins) of the existing MPS IDE as desktop-client application and/or a web-client must be supported.<br>Note: DSML developers may also use a different approach. | BC4, BC5 |

### 14.2.3. Contextual Integration

As explained in Section 14.1.4, only two use cases have expressed a need for integrating their BUMBLE-based DSML solutions as part of a larger application context. At the moment of writing this version of this deliverable, insufficient common technical requirements have been identified. Hence, we have currently only two technical requirements to consider in WP3, WP4 and WP5 for this aspect.

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| BT17 | B, C, E, T | It should not be impossible to integrate BUMBLE technological solutions as 'DSML components' in a bigger non DSML-technology based application. | BC1, BC2, BC3, BC4, BC5, BC6, BC10, BC11, BC12, BC13 |
| BT18 | B, C, E, T | In the case of integrating BUMBLE technological solutions as 'DSML components' in a bigger (non DSML-technology based) modeling environments, it should not be impossible to use traditional GUI widgets to represent certain elements of DSML model instances, i.e., traditional GUI widgets such as a checkbox or radio button being a (default) concrete syntax. | BC1, BC2, BC3, BC4, BC6 |

### 14.2.4. Model Life-Cycle Management

**Non Real-Time Collaboration**

Section 14.1.5 highlights a few specific version control systems mentioned in the context of different use cases that are to be supported. We have selected this ability as a common technical requirement.

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| BT19 | C, E, T | File-based version control must at least be possible based on the traditional GIT and SVN approaches (for both DSML model definitions and DSML model instances). | BC7, BC8 |
| BT20 | B, C, E, T | Version control functionality (e.g., diff/merge/tagging) should be accessible by a DSML user at any available concrete syntax for the considered (collection of) (elements of) (a) DSML model instance. This requires diff/merge functionality at persistence level to be (bi-directionally) linked to diff/merge views at DSML model instance level. | BC1, BC2, BC7, BC8 |
| BT21 | E, T | Version control of DSML model definitions must not break concurrent use of instances of such DSML model definitions. Any conflicts that may arise must be either taken care of automatically or resolved by DSML users. | BC7, BC8, BC9 |

**(Automated) (Co-)Evolution**

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| BT22 | E, T | DSML users who are editing/viewing instances of DSML model definitions that are to be updated with a new version by a DSML developer should be informed about an (upcoming) migration of these instances. | BC9 |
| BT23 | E, T | DSML users who are editing/viewing instances of DSML model definitions that are updated with a new version by a DSML developer should be able to view the differences with the previous version to be able to understand the impact of automatic migrations of these instances. | BC9 |
| BT24 | E, T | Migration of instances of DSML model definitions to a new version should come | BC1, BC9 |

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| | | with migration of relevant editors/views for all existing concrete syntaxes. | |

**Access Control**

Although a number of common core requirements have been selected in Section 14.1.5, only very few technical requirements have been identified at the moment of writing this version of this deliverable. Hence, for now, we have identified one common technical requirement for access control. Further detailing is expected to emerge during the next period of the BUMBLE project when more practical experience is obtained in deploying the BUMBLE technologies with respect to collaboration.

| ID | Classification (B, C, E, T, N) | Description of Requirement | Details Core Requirement(s) |
|---|---|---|---|
| BT25 | C, E, T | DSML users can authenticate via standard external infrastructural authentication services, including LDAP and OAUTH2[2]. | BC10, BC11, BC12, BC13 |

[2] Although not explicitly mentioned by use cases, support for authentication via SSSD and/or Microsoft AD services may also be required.