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 3)

BUMBLE Requirements Specification

Edited by: BUMBLE Team
Date: November 2022

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

Contents

Acronyms

AD	Microsoft Azure Active Directory
ADL	Architecture Description Language
API	Application Programming Interface
B	Blended Syntaxes & Modelling
BPM4DCA	Business Process Management for Debt Collector Advocates
C	Collaborative Modelling
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 Modelling Language
E	Evolution
EAXML	East-ADL XML
ECU	Electronic Control Unit
ELK	Eclipse Layout Kernel
EMF	Eclipse Modelling Framework
EMOF	Essential MOF
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	Modelling 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

Versions

Release	Date	Reason of change	Status	Distribution
V1.0	31/07/2021	First release of D2.2	Final	Uploaded to ITEA portal
V2.0	11/05/2022	Second release of D2.2	Final	Uploaded to ITEA portal
V3.0	29/9/2022	Third release of D2.2	Final	Uploaded to ITEA portal

Contributors

Federico Ciccozzi	MDU
Martin Axelsson, Mattias Mohlin	HCL
Henrik Lönn	Volvo
Bart Theelen, Joost van Pinxten, Roelof Hamberg	Canon Production Printing
Wim Bast	MVG
Johan Fredriksson	Saab
Alexander Darmonski, Duncan Stiphout	Sioux
Katira Soleymanzadeh, Geylani Kardas	Hermes
Ferhat Erata	Ford Otosan / UNIT
Elisabeth Schold Linnér, Johan Ersson	Unibap
Gerald Stieglbauer	AVL
Staffan Skogby, Mikael Tillman	Pictor Consulting
Pelin Latifoglu, Ilkay Yelmen	Turkcell

Reviewers

Jan-Philipp Steghöfer	University of Gothenburg
Federico Ciccozzi	Mälardalen University
Detlef Scholle	Pictor Consulting

1 Introduction

This document describes requirements for blended collaborative modelling as identified for the use cases of BUMBLE, see Deliverable D2.1 (updated version 1.1). The purpose of these requirements is to clarify what functionality must be supported by 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.

Deliverable D2.2 was planned to have three versions, see Section 1.2. This delivery document is the third release (version 3). The first version has provided an extraction of requirements from all the use cases to come to a set of common requirements that were instrumental in starting the work packages WP3, WP4, and WP5. The second version provides a refinement of the requirements where needed, arising from new insights, a reflection of the common requirements on the individual use cases, and processed feedback from the work packages WP3, WP4, and WP5. The most relevant update to the second version is the organisation of common requirements along the lines of Technology Bricks. This third version includes a consolidated list of requirements together with an update of their status. The third version is also the final version.

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 13 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 (kind)	Description	Lead Partner
UC1 (S)	Software Open-Source Blended Modelling	MDU
UC2 (S)	Combined Textual and Graphical Modelling of State Machines in HCL RTist	HCL
UC3 (S)	Vehicular Architectural Modelling 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 Modelling 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
UC11 (T)	Multi-Aspect Modelling for Highly Configurable Automotive Test Beds Ready for Smart Engineering Demands	AVL
UC12 (T)	Agile V-model System Architecture	Pictor
UC13 (T)	Automatic CFP (Cosmic Function Point) Value Generation for Software Analysis Documents	Turkcell

1.1 Classification of Requirements

For the complex functionalities that are to be realised 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 context, i.e., a specific domain. 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 realised 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, generators, etc. that come with 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 Stakeholder Requirement - A core stakeholder requirement (also shortened to core requirement) describes a key principle that is to be supported. This is expressed from the perspective of the stakeholders 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 realising added value (in the form of features) in the context in which the DSML user and/or DSL developer applies the BUMBLE technologies.

Definition 4: Technical Solution Requirement - Technical solution requirements (also shortened to technical requirements) detail core stakeholder requirements from the perspective of the solution approach or solution technology. Technical requirements primarily address the needs of DSML

[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. This updated list of use cases has been implemented and described in CR4.

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. In this third version, the lists of technical requirements have been consolidated with respect to the second version of this document.

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 in this deliverable). Note that a requirement can be relevant for multiple key aspects:

- **Blended Syntaxes & Modelling (B).** This includes any aspect related to synchronisation / transformation between multiple representations (syntaxes).
- **Collaborative Modelling (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 2 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

Three versions of this deliverable will be delivered in accordance with the BUMBLE project plan. For this is the third version, the lists of requirements from the first two versions have been consolidated. The numbering of requirements has been kept intact to not break existing references to requirements. Requirements of UC13 (Turkcell) have been included in this final version.
Focus has been on elaborating on the technical requirements, which are gradually emerging from the technological choices made. In Chapter 2, the selected common requirements are also related to the BUMBLE technology bricks, which were introduced in the project to organise the partial solution concepts provided by BUMBLE.

1.3 Status of Requirements

In this third version of Deliverable D2.2, we also indicate to what extent the identified requirements have already been satisfied by the BUMBLE technologies. We do so by using a colour coding along with the identifier of a requirement as follows:

- Grey indicates the initial requirement is defined.

- Red denotes that the requirement has been defined and refined, and as such has been addressed, whereas any activities to realise the BUMBLE technologies to satisfy the requirement are still to be started.
- Yellow reflects that the requirement has not yet been fully satisfied and work to realise the required BUMBLE technologies is still in progress. A repeatable validation test may or may not exist already and a partial demonstration may be feasible, where error/warnings may occur that are still to be resolved as part of finalising the realisation of the BUMBLE technologies.
- Green indicates that a requirement is fully satisfied by the BUMBLE technologies. This has been confirmed by a validation test that can be repeated and shown as part of a demonstrator.

2 BUMBLE Selection of Common Requirements

This chapter summarises 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 shows their generic applicability, notwithstanding customizations that may have to be done to make them fit specific contexts.

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 2.2. It still may happen that further core requirements may arise during the BUMBLE project and/or that the list of selected common technical requirements in Section 2.3 will increase. When significant and relevant, updates and extensions will be documented by just updating the final version of this deliverable.

Note that the referenced UC-specific core and technical requirements can be found in Chapter 3.

2.1 Structuring Requirements along BUMBLE Technology Bricks

In the first version of this document, a further structuring of requirements next to the BUMBLE features Blended (B), Collaboration (C), Evolution (E), Traceability (T) and Non-Conformance (N) was introduced. This additional structuring was comprised of the following categories:

- Blended Modelling
- Real-Time Collaboration
- Model Non-Conformance
- Contextual Integration
- Model Life-Cycle Management, with sub-categories:

- Version control (Non-Real-Time Collaboration)
- Persistence
- (Automated) (Co-)Evolution
- Access control

In the second version, progressed insights in the BUMBLE project led to a slightly different categorization that especially helps in further structuring the common technical requirements. These new categories are called BUMBLE Technology Bricks, and – as the term suggests – they span the space of partial solutions that are being developed in the BUMBLE project. This categorization along these bricks has been kept in this version as well.
The Technology Bricks, according to which the common requirements in this section are organised, are as follows:

- **Editor Generators**: Derive blended editors from meta-model definitions. Either create fully functional editors out of the box or create extensible templates for blended editors.
- **Blended Editors**: The editors themselves, regardless of concrete syntax.
- **Blended Model Access**: Provide access to the models that are edited in a blended way. This technology brick contains all functionality that is not part of the editors themselves, e.g., functionality for access control to the models.
- **Collaboration Engine**: Ensure that concurrent changes are synchronised between users and that conflicts are resolved.
- **Diff and Merge**: Provide conflict resolution mechanisms in case collaboration happens on a file level, e.g., via version control.
- **(Meta-)Model Co-Evolution**: Ensure that models are updated regardless of their concrete syntax when the meta-model changes.
- **Platform Integration**: Address all aspects that are related to embedding the editor generators and the blended editors into the respective platform (e.g., Eclipse, MPS, or VS.code).

Both the Blended Editors and the Platform Integration technology brick have no associated core requirements. Instead, they address technical requirements only. When designing the technology bricks, it became clear that much of the editor functionality is independent of the concrete syntax and has therefore been moved to other technology bricks such as blended model access. The requirements for platform integration are likewise not at the core of the project but result from the necessity to embed the project results into existing IDEs.

2.2 BUMBLE Core Stakeholder Requirements

2.2.1 Editor Generators

At the core of BUMBLE is the ability to exploit multiple syntaxes for a DSML in a flexible way. 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 modelling 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. The need to enable this in a cost-effective way at high quality is reflected in the BUMBLE technology brick Editor Generators. 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	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

2.2.2 Blended Model Access

The requirements for having different users applying the blended editors are collected in this category. Blended editors require different users to specify which editing syntax is applied, whereas for the model content it should be indifferent which variant they apply.
Further, 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)
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 visible in one or more specific concrete syntaxes.	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
BC10	C, E, T	In view of various CRUD functionalities, 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 modelling 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 modelling 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

2.2.3 Collaboration Engine

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 explicitly in BUMBLE. There are no 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 which model element(s) other DSML users are editing/viewing at the same moment in time.

Although DSML technologies allow for developing the next generation of modelling environments, these are generally to be integrated as part of a bigger context (which may not rely on any DSML technology) for collaboration purposes. Since BUMBLE technologies should facilitate 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.

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
BC6	B, C	It must be possible to integrate BUMBLE-based DSML environments in larger non-DSML-technology-based applications, the latter enabling real-time or non-real-time	C4.23, C4.26, C7.1

		collaboration between users in a larger context (i.e., including users who do not apply DSML technology).	

2.2.4 Diff and Merge

Most 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

2.2.5 (Meta-)Model 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 realised 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

2.2.6 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.

2.2.7 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-)serialisation 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 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 cases 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.

2.3 BUMBLE Technical Solution Requirements

BUMBLE focuses 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 realise 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 considering (and hence referring to) relevant DSML technology platform contexts.

2.3.1 Blended Editors

Many of the blended editor 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.

ID	Classification (B, C, E, T, N)	Description of Requirement	Cover UC-Specific Core Requirement(s)
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

2.3.2 Editor Generators

The availability of blended editors from the previous section is facilitated in a cost-effective way at higher quality when they can be generated from the DSML model definition. The common requirements are overlapping with the requirements for blended editors, although they pertain to the generation facilities in this case.

ID	Classification (B, C, E, T, N)	Description of Requirement	Cover UC-Specific Core Requirement(s)
BT1	B	At least one editor/view (i.e., concrete syntax) should 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

2.3.3 Blended Model Access

The requirements for having different users applying the blended editors are collected in this category. Blended editors require different users to specify which editing syntax is applied, whereas for the model content it should be indifferent which variant they apply.
Additionally, there is a common technical requirement for access control to identify users and their roles/authorization levels. Further detailing can emerge during 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	Cover UC-Specific Core Requirement(s)
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 semi-automatic approach to generate synchronisation 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: ● automated real-time (on-the-fly) synchronisation/transformation. ● on-demand (i.e., based on an explicit request of a DSML user)	BC1, BC3, BC4, BC6, BC7, BC8

		synchronisation/transformation. Next to this, it should also enable supporting: ● synchronisation/transformation via file-based version control.	
BT9	B, 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 realised 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, 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 DSML model instance and/or of other relevant DSML model instances causing the error/notification to be present.	BC4
BT11 Eclipse only	B	It must be possible to create/use EMOF-based DSML model definitions.	BC1
BT12 Eclipse only	B	It must be possible to define (a) Xtext-based textual editor(s)/view(s) for a DSML.	BC1
BT13 Eclipse only	B	It must be possible to define (a) tree-based editor(s)/view(s) for (elements of) a DSML model definition.	BC1
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
BT21	C, 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
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.

2.3.4 Collaboration Engine

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 centralised server, as well as for real-time collaboration exploiting web-clients using traditional internet-browser technology and a centralised server. Both options are considered relevant in the BUMBLE context, although some use cases are specifically referring to the web-based approach.

ID	Classification (B, C, E, T, N)	Description of Requirement	Cover UC-Specific 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
BT15 Eclipse only	B, C, E, T, N	DSML developers should be able to realise real-time collaboration (e.g., based on an LSP/GLSP-based approach) using an extension (i.e., by means of plugins) of the existing Eclipse IDE as desktop-client application. Note: DSML developers may also use a different approach	BC4, BC5
BT16 MPS only	B, C, E, T	DSML developers should be able to realise 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. Note: DSML developers may also use a different approach.	BC4, BC5

2.3.5 Diff and Merge

A few specific version control systems, mentioned in the context of different use cases, are to be supported. This leads to a few common technical requirements.

ID	Classification (B, C, E, T, N)	Description of Requirement	Cover UC-Specific Core Requirement(s)
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
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	BC9

		to view the differences with the previous version to be able to understand the impact of automatic migrations of these instances.	

2.3.6 (Meta-)Model Co-Evolution

The (co-)evolution of (collections of interrelated) DSML model definitions and instances leads to a few relevant requirements on the primary facilities to be realised by the BUMBLE technologies.

ID	Classification (B, C, E, T, N)	Description of Requirement	Cover UC-Specific 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
BT24	E, T	Migration of instances of DSML model definitions to a new version should come with migration of relevant editors/views for all existing concrete syntaxes.	BC1, BC9

2.3.7 Platform Integration

Platform integration requirements stem from the internal requirements to be able to integrate solutions in larger contexts. Additionally, two use cases explicitly expressed a need for integrating their BUMBLE-based DSML solutions as part of a larger application context.

ID	Classification (B, C, E, T, N)	Description of Requirement	Cover UC-Specific Core Requirement(s)
BT5	B	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	DSML developers must be able to specify a default editor/view (i.e., concrete syntax) for (elements of) a DSML model instance that 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).	BC2
BT9	B, 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 realised 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, 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 DSML model instance and/or of other relevant DSML model instances causing the error/notification to be present.	BC4
BT17	B, C, E, T	It must be possible to integrate BUMBLE technological solutions as 'DSML components' in a larger 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) modelling environments, it must be possible 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

3 Use Case Requirements

3.1 UC1 - Software Open-Source Blended Modelling

This use case covers a public showcase for the BUMBLE technologies. Starting from a EMOF-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 synchronisation mappings between the DSML and the generated notations. Given the DSML, the generated notations, and the model synchronisation mappings, the framework is expected to semi-automatically generate synchronisation 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 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 should allow to keep track of different versions of the edited models via a set of Git-like diff/merging functionalities.

3.1.1 Core Stakeholder 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 synchronisation mechanisms (model transformations) to keep generated notation and DSML in sync.
C1.4	B, C	The framework shall allow change propagation across notations and synchronisation 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.

3.1.2 Technical Solution 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 EMOF-based DSMLs.	C1.1, C1.2, C1.3, C1.4, C1.5, C1.6

3.2 UC2 - Combined Textual and Graphical Modelling 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.

3.2.1 Core Stakeholder 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 refer 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	Like C2.4 Content Assist (a.k.a. "code completion") for references should utilise 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 colours, symbol sizes, line routing and other layout information.
C2.8	C	Textual state machines should work well with the existing Compare/Merge tooling in RTist. Running a compare or merge session where some state machines are defined with the textual notation and others with the graphical notation should be possible.

3.2.2 Technical Solution 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 utilises 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.	C2.1
T2.3	B	A textually defined state machine should be persisted in a file with the file extension .srt ("state machine real-time")	C2.1
T2.4	B	When an .srt file is created from an already existing state machine, the .srt file should be automatically populated with textual syntax corresponding to the existing state machine. The syntax should be formatted so it is readable.	C2.2

T2.5	B, T	Items shown in the "Content Assist" popup should use the same icons and labels as elsewhere in RTist.	C2.5
T2.6	B	Semantic validation of a textual state machine should be performed as soon as the text changes, and errors should be shown both in the text editor and in the Problems view (with a possibility to navigate to the correct location in the editor)	C2.6
T2.7	B	When creating a state machine diagram for a textually defined state machine, it should be automatically populated with symbols and lines with a nice, automated layout.	C2.7
T2.8	B	To be able to persist graphical changes made by the user in a state machine diagram for a textually defined state machine, an extra file ".srtd" with the same name as the .srt file should be created next to it. This file should contain those graphical changes (e.g., the user-assigned colour of a state symbol) in a JSON format.	C2.7
T2.9	C	Comparing or merging changes in a textual state machine should use the usual Eclipse features that are used for textual files.	C2.8

3.3 UC3 - Vehicular Architectural Modelling in EAST-ADL

Development of automotive embedded systems at Volvo involves large amounts of data from multiple stakeholders. To organise 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.

3.3.1 Core Stakeholder Requirements

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: ● **Package Containment** Edit only elements in the selected package or its sub-packages. ● **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, behaviour. ● **Element Criteria** Edit only elements that fulfil a selected set of criteria, e.g., allocated to a certain ECU, realising 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, realise 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: ● Graphical information such as colours 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. ● Meta information needed by a textual editor shall also be separated.
C3.4	N	It should be possible to create models in the editor that do not fully conform to the meta-model 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.

(De-)Serialisation

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 serialisation. 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 serialisation.

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 sub setting, 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.

Textual Modelling

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 to the package for anything added. For example, packages with data types or other elements may be imported and subsequently visible and part of the scope.

Graphical Modelling

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], realisations; 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., realising 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 customise 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 modelling 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 specialised diagrams for the safety cases are also necessary.
C3.18	B	The editor should support auto-layouting that automatically selects the diagram type and the kind of visualisation (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.

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 modelling 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	Visualising 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.

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 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 to support engineers that have been working on the model without access to the concurrent editing environment.

3.3.2 Technical Solution 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 favoured 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. Particularly, this includes two aspects. First, the model information that is relevant to certain views / diagrams / text editors must be identified and	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

		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 synchronise if the information is relevant for both domains. This effortful procedure shall be conveniently guided for the language engineer for efficiency purposes (e.g., by means of documentation and/or automation).	

3.4 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 Modelling 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 modellers.

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 (and languages) 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.

3.4.1 Core Stakeholder Requirements

Modelling 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/organisation of models (folders/packages, as well as model roots), to achieve a maintainable organisation of the modelling 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 modelling environment and modelling entities.
C4.6	C	DSML users can generate/download/deploy modelling artefacts, so that modelling artefacts can be used outside of the modelling 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.
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 modelling reference (hyperlink) from the model editor, so that they can easily navigate the relationships between models.

Blended Modelling

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 simultaneously.
C4.27	B, C, T	DSML users can view and diff models (in different projections)

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 modelling 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 model 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.
C4.28	B, C, T	DSML users can edit the same part of a model simultaneously; if conflicts occur, they are preferably automatically resolved, or resolution actions are proposed to the users.

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.
C4.19	B, C, E, T	DSML developers can perform (automated) language migrations, so that the models become consistent with the new language.

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 modelling 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 visualise (interactively, inline, or in an external window) the results of the modelling artefacts, to achieve a smooth integration between the specification and the visualisation.
C4.25	C	DSML users can use model editors within a larger application that defines the workflow of the modelling 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.

3.4.2 Technical Solution 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
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 assign/remove users to groups.	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
T4.10a	C, T	Users can tag models and model versions using textual tags for later reference.	C4.4
T4.10b	C, T	Users can tag model versions using Git tags for later reference.	C4.4
T4.11	C, T	Users can generate and download/transport/deploy the artefacts to a defined location (local PC, Git, Windows Share).	C4.6
T4.12	C	Users can start (predefined) external tools (simulators, visualisations) from the modelling 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

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 modelling 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
T4.33	B, C, T	DSML users can see differences between two versions of a model, where both models are shown in a projection selected by the user.	C4.27
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 modelling conflicts.	C4.12
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
T4.34	B, C, T	When DSML users create a conflict by editing a part of a model simultaneously, conflicts are resolved automatically where possible.	C4.28
T4.35	B, C, T	When DSML users create a conflict by editing a part of a model simultaneously, and conflict resolution is not possible, the DSML users are informed about the conflict, and need to resolve the conflicts manually before they can continue editing the model.	C4.28

3.5 UC5 - Reactive and Incremental Transformations across DSMLs

The Modelling Value Group (MVG) aims for a generic (language independent) open-source solution for collaborative and blending modelling. The solution is built on top of Dclare. Dclare is an open-source framework for declarative and reactive model-based solutions. The use case of the MVG combines collaborative and blending modelling of two different state-transition modelling-languages that are transformed and synchronised instantly. The modellers (the users in the use-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 Modelling Value Group, and essential features relevant for most of the BUMBLE partners.

The two models can both be changed independently and synchronised later-on, or immediately synchronise when either model is changed. Furthermore, the two models are not wired together persistently, the transformation will match the models only when synchronised 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.

In addition to the model changes, information about all the (editing and viewing) users and the current focus of each user on a model-element are exchanged and visualised. The goal is to share information so that a user has a feeling where other users are working (vs looking) at in the synchronised models. This implies that this information should also be sent across the network and synchronised over the transformations between different languages.

3.5.1 Core Stakeholder Requirements

ID	Classification (B, C, E, T, N)	Description of Requirement
C5.1	B	Bi-directional immediate transformation and synchronisation for the DSML user.
C5.2	B	Non-trivial bi-directional transformation of abstract syntaxes.
C5.3	C	Remote synchronisation across different modelling clients for the DSML user.
C5.4	E	Combining immediate and deferred synchronisation by activating and deactivating immediate synchronisation and updating models via a VCS when not synchronising.
C5.5	B	Easy specification of non-trivial bid-directional transformations by the DSML developer.
C5.6	C	Focus information of the different collaborating users need to be exchanged and visualised, like google docs functionality.

3.5.2 Technical Solution 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 synchronisation and transformation of the models. The delta's broadcast server (part of Dclare) will be used to exchange mutations across different synchronised MPS clients. The MPS Git integration will be used to synchronise and or merge in a deferred manner. The rule definition-aspect of DclareForMPS will be used to define the (bi-directional) transformation between the two abstract syntaxes.

The exchanged meta-information will be part of any Dclare model-element. In this way we can exchange that information between different technologies like EMF and MPS.

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, C5.2
T5.2	C	The part (MPS models) that is remotely and immediately synchronised is chosen using a dialogue integrated within MPS.	C5.3
T5.3	C	The combination of immediate and deferred communication will be done consistently, hence no unnecessary model changes nor unnecessary conflicts may appear. In this use case we will use the standard GIT integration of MPS to support the deferred synchronisation. This implies that DclareForMPS needs to exchange the identity of the models and nodes between the (immediate) collaborating modellers so that they are recognised by the git pushes and updates as being the same model-elements.	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
T5.5	C	Model elements in Dclare will contain meta-information and all meta-information will be exchanged with the remote synchronisation and transported across the transformation relations. Meta information is not persistent and therefore not relevant for deferred synchronisation.	C5.6
T5.6	C	The focus (in the models on the nodes) of the collaborating MPS clients need to be visualised in the (projective) editors in MPS. The solution for this needs to be language independent, so that no additions to the language need to be made to support this functionality.	C5.6
T5.7	C, B	Since we want to combine blending and collaboration we need to 'transport' meta information across the transformed models to visualise the focus and collaborating users in each preferred language.	C5.6, C5.1, C5.2

3.6 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.

3.6.1 Core Stakeholder Requirements

ID	Classification (B, C, E, T, N)	Description of Requirement
C6.1	B, T	Bi-directional transformation and synchronisation 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.

3.6.2 Technical Solution 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

3.7 UC7 - Multi- and Cross-Disciplinary Modelling 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 modelling. Within BUMBLE we aim at creating a blended modelling 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 synchronisation is expected between Supermodels and MPS views on the multi-aspect system specification. Support of version control (Git, SVN) collaboration is expected between multiple DSML users working on the same model. A first iteration prototype is expected to visualise differences well enough between models using graphical DSMLs and resolve conflicting changes on the DSML level.

3.7.1 Core Stakeholder 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 an optional front-end (both running on the same machine). DSML users perceive the blended ME as one environment.
C7.2	B	DSML users 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 synchronised (on-the-fly).
C7.4	B	The blended ME should allow model checks to be triggered from MPS.
C7.5	B	The blended ME should allow generation to be triggered from 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 developers can further develop the existing Supermodels DSMLs.
C7.9	B, E	DSML developers can implement (new) DSMLs in MPS for which it can implement diagrammatic editors in Supermodels.
C7.10	B	DSML users can open (or create new) and save (persist) a model from MPS.

3.7.2 Technical Solution 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 by starting Supermodels and MPS.	C7.1
T7.3	B	The blended ME should allow for flexible deployments depending on DSML user needs: ● only MPS (Supermodels is not deployed/started). ● both Supermodels and MPS.	C7.1, C7.2
T7.4	B	Usability: Synchronisation 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

T7.7	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: visualise 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
T7.11	B	DSML user should be able to view and edit models in different notations (syntaxes) for the same DSML.	C7.1
T7.12	B	DSML user can follow and/or create links between models of different DSMLs irrespective of their notation (syntax).	C7.1

3.8 UC8 - Model-Driven Development of Workflow Models for Debt Collecting Advocacy

HERMES İletisim's main job is creating digital solutions especially in the 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 10.000 case files on average which must be handled only in one month. Modelling with BPM4DCA consists of both various modelling viewpoints and the construction of relations required for managing the desired workflows. The blended modelling approach brought by the BUMBLE project will facilitate modelling and implementation of both choreography and orchestration of complex business services inside BPM4DCA.

3.8.1 Core Stakeholder Requirements

ID	Classification (B, C, E, T, N)	Description of Requirement
C8.1	C	Users can be authenticated to login to get access to a graphical and textual modelling environment and repository.
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 to be able to trace the model's instances and if these are defined properly.
C8.7	B	Users can be informed about the notifications and errors in the modelling 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 visualise relations of their workflows in a single diagram.
C8.11	B	Users can modify the workflow in a textual and graphical editing environment with low code or no code.

3.8.2 Technical Solution 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.	C8.1, C8.4
T8.2	B	The workflow rules will be modelled in the Blockly environment based on BPM4DCA's graphical syntax for rule creation	C8.4, C8.6, C8.7
T8.3	B, T	Workflow rules will be graphically or textually edited at the same time (Synchronisation between rule models in JsonLogic and Blockly will be automatically provided).	C8.4, C8.11
T8.4	B	The model will be generated in XML format to store in the database.	C8.1, C8.3, C8.5, C8.9
T8.5	B	Design of the workflow will be performed inside BPMDCA's graphical modelling environment based on the MxGraph.	C8.2, C8.6, C8.7, C8.8, C8.10
T8.6	B	XML encoded workflow models will also be textually created or edited.	C8.10
T8.7	B, T	All BPM4DCA workflows will be graphically or textually edited at the same time (Synchronisation between workflow models in	C8.8, C8.11

		XML and MxGraph will be automatically provided).	

3.9 UC9 - Automated Design Rule Verification on Vehicle Models

Ford Otosan aims to have a software solution that automates design rule verification on vehicle models in collaboration with UNIT Information Technologies R&D Ltd. Two main components of the use case will benefit from the software deliverables of the BUMBLE project: the textual and graphical representation of design requirements and touch conditions; the synchronisation of design rules with the geometry and product manufacturing information.

We'll develop a textual domain-specific language to formalise clearance rules of Ford-Otosan conforming to ISO's XMI standard (ISO/IEC 19503:2005) and a graphical projection of the touch conditions of parts in the 3D models. We'll check the validity of the design rules against manufacturing and geometric data using ISO's JT Standard (ISO 14306:2017) using a synchronisation engine to be developed on top of traceability facilities of the BUMBLE project.

The BUMBLE features covered by this use case are:

- Blended Syntaxes and Modelling (B): We aim at using BUMBLE modules that support generation of the blended modelling environment. We will develop two domain specific languages: a graphical touch-condition diagram that identifies subsystem and part hierarchy as well as touch conditions and a textual, syntax-directed editor for design-rule specification.
- Collaborative Modelling (C): Since different engineering teams work on various subsystems of a vehicle model under development, the design rule repository should allow for collaborative editing. Therefore, we aim to exploit the features of the BUMBLE project that facilitates collaborative modelling, particularly on the textual part of the blended language.
- Traceability (T): All touch conditions should be traced back to the CAD designs of the vehicles aligning with the ISO's JT Standard (ISO 14306:2017). Actually, all touch conditions must be first generated from the CAD designs and then kept synchronised throughout the design process. If there is an inconsistency detected among synchronisation points, it should be reported to the development team pinpointing the source of the inconsistency.
- Model Non-Conformance (N): The main purpose of the project is to identify clearance violations among touch conditions, which requires automated geometric reasoning on CAD models. This can be only achieved by checking whether CAD design meets the design rules (mainly clearance rules). Therefore, this module will be separately developed by UNIT and integrated to the BUMBLE's blended modelling environment on top of the traceability infrastructure of the BUMBLE environment.

3.9.1 Core Stakeholder Requirements

ID	Classification (B, C, E, T, N)	Description of Requirement
C9.1	B	The textual Clearance Rule Specification notation should cover all aspects of part hierarchy and Clearance Rules as well as touch conditions of parts. The editor should allow DSML users to create and update clearance rules without using any other notation or view.
C9.2	B	Changes in specification rules of a given vehicle model should update the textual notation without losing non-semantic information it contains, such as

		comments, indentations, and other white-space characters.
C9.3	B, E, T	Changing a Rule of a specific part in the textual notation should update the semantic model in a way that the identity of the model elements and their relative geometrical positions to other parts are preserved.
C9.4	B, T	References in the textual specification that refer to other elements defined outside the clearance rule specification should be bound to the correct model element.
C9.5	B, T	Like C2.4 Content Assist (a.k.a. "code completion") for references should utilise an external part hierarchy model in which a hierarchical data structure is defined and for each part or subsystem in the hierarchy has its own properties such as identifiers and geometric data. This data should be able to be used for content assist.
C9.6	B	Semantic checks (a.k.a. validation rules) should be implemented which detects semantically incorrect models which the textual syntax permits creating. For instance, if there is no touch condition defined in the geometric data then a clearance rule from part to part should not be created. However, a clearance should be able to be created among modules.
C9.7	B	Changes in the state machine model should update the graphical notation without losing non-semantic information it contains, such as colours, symbol sizes, line routing and other layout information.
C9.8	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.

3.9.2 Technical Solution Requirements

ID	Classification (B, C, E, T, N)	Description of Requirement	Details Core Requirement(s)
T9.1	B, T	The Tool should enable synchronisation of a projectional Graphical notation for touch conditions.	C9.4
T9.2	B	A textually defined Design Rule should be persisted using the textual syntax, while other parts of the model be persisted as XMI.	C9.2
T9.3	B	a graphical projection of the touch conditions of parts in the 3D models. We'll check the validity of the design rules against manufacturing and geometric data using ISO's JT Standard (ISO 14306:2017) using a synchronisation engine to be developed on top of traceability facilities of the BUMBLE project.	C9.6
T9.4	N	The Design Rules should be validated against graphical Data.	C9.6, C9.7

3.10 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 utilisation of valuable resources, as well as in ensuring high quality in our products.

3.10.1 Core Stakeholder 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.

3.10.2 Technical Solution 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.	-

3.11 UC11 - Multi-Aspect Modelling 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 modelling. Regarding collaborative modelling, two dimensions are of interest: One is about enhancing existing/new DSLs in terms of collaborative modelling within a dedicated user group/department (e.g., graphical model diff), while the second dimension is about collaborative modelling across departments. 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).

3.11.1 Core Stakeholder 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 must always 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 aspects of the model (reduced information), this is not sufficiently expressed by the driving one (without additional 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).
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 are illustrated on the alternative representation (e.g., before-after-comparison).
C11.5	B, T	DSL must have the possibility to reference external elements either from other data sources or to related DSLs.
C11.6	B, T	Alternative data sources should be visualised 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 synchronise 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 favoured.
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.
C11.17	E	If models cannot be updated automatically, invalid model elements should be indicated, and users 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 visualised to support decision making in fixing the inconsistencies.

3.11.2 Technical Solution Requirements

This use case will be realised 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 synchronisation.	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

3.12 UC12 - Agile V-model System Architecture

A digital twin is a virtual representation of a real-world infrastructure that serves as its digital counterpart for maintenance purposes. The digital twin concept is observed in several applications domains around the world. In UC12, the application is structural engineering for infrastructure. To make the complex modelling of such digital twins accessible, multiple views on the same information need to be supported.

The digital twin must be supported by blended modelling 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 a building structure with thousands of elements and hinges in different materials, for example, bridges for roads and railway tracks in concrete with steel cables. The targets have requirements according to European standards with a high regard for safety for public transportation such as the European norms EN 1992 for concrete structures. These standards are continuously updated with more advanced practices in civil engineering for building bridges in concrete and steel. New civil engineering practices impose new requirements on the software packages which by using the blended modelling approach, better meet requirements on correctness and safety in the structural analysis calculations.

The purpose of the Civil Engineering structural design demonstrator in UC12 is to demonstrate the advantages with the blended modelling concept and an agile V-model approach to prepare a baseline for more efficiency in the process of fulfilling the quality and safety requirements in the structural design work as well as increased efficiency in the structural design process and that way shorten TTC (Time-To-Customer).

The demonstrator shall be implemented starting with basic functionality, that shall evolve to a complete prototype showing most benefits with blended modelling in three steps, that are outlined below. Our aim is to achieve step 1 and Step 2 in the BUMBLE project.

The steps are outlined below:

Step 1. The database for the model resides in MPS. A DSML User shall be able to perform limited blended modelling. This means that a change in the tabular notation, upon request, will propagate to the 3D view, thus a simple live/up-to-date visualisation of the model.

Step 2. The database for the model resides in MPS. A DSML User shall be able to perform blended modelling, i.e., changes and viewing of the model is possible in both the tabular and in the 3D notation that are synced.

Step 3. The database for the model resides in a cloud. DSML Users shall be able to perform collaborative blended modelling in real-time via web access. DSML developers shall be able to perform modelling support over web access. Single/Multi DSML Users shall be able to perform model administration over web access.

Model administration means ability to version the models and apply diff/merge features in a GitHub manner, ability to navigate the existing models, manage a hierarchy of models, to achieve a maintainable blended model handling.

Blended modelling in UC12 means to build a model of the Civil Engineering structure by assembling building elements in the tabular notation and in the 3D notation. Further it shall be possible to add symbols representing reaction forces and building element types.

Modelling support is the CRUD of symbols in the symbol table and of building element types of rods, beams, joint, reactions, polygons, frames etc. in other tables. It also covers CRUD of the table notation itself, if this needs to be extended with new functionality for example for dimensioning.

The DSML tabular notation shall be built using a de facto construction methodology. In the demonstrator the focus is on graphical representation of the building element types listed above and labelling of element types and reaction forces (symbols) to be used for demonstration purposes of the blended concept. This means that load indications and calculations are excluded.

The DSML tabular notation covers much more than what shall be covered in the demonstrator, the reason for this is that the structural designer shall be in a well-known environment. Lot of the parameters in the tables shall be left unattended thus indicated with a “not applicable” or simply filled with grey. The tables contain definition of needed element data and its geometrical characteristics, position and cross point data representing the complete structure.

3.12.1 Core Stakeholder Requirements

ID	Classification (B, C, E, T, N)	Description of Requirement
C12.1	B	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 synchronisation mechanisms (model transformations) to keep generated notation and DSML in sync.
C12.4	B, C	The framework shall allow change propagation across notations and upon user's choice.
C12.5	C	The framework shall allow a model to be viewed and edited in a collaborative manner, using a single blended fashion, which means that CRUD performed in the textual notation can be viewed in the 3D notation.
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.
C12.9	C	DSML users can manage (CRUD) a set of models, to achieve a maintainable organisation of the modelling content.
C12.10	C	DSML users can tag model versions, so that they can be used as snapshots for later reference.

3.12.2 Technical Solution 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 MPS 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 centre 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
T12.7	C, B	A DSML User shall be able to perform blended modelling by using tabular and 3D notation. Further it shall be possible to add symbols representing reaction forces and building element types	C12.1
T12.8	C, B	A DSML Developer shall be able to perform CRUD of symbols in symbol tables and of building element types of rods, beams, joint, reactions, polygons, frames etc. in other tables.	C12.1

3.13 UC13 - Automatic CFP (Cosmic Function Point) Value Generation for Software Analysis Documents

Turkcell aims to model the analysis documents of various existing Turkcell-developed services and new Turkcell services developed during the BUMBLE project from its own perspective. For this purpose, it is planned to apply the blended modelling methods to be developed in BUMBLE to Turkcell Academy software product use cases. Thus, it is aimed to express free text or semi-structured requirements in formal languages. In this way, standardisation of Turkcell requirements engineering practices will be ensured and automation possibilities will be examined. Thanks to the standardisation and potential automation of manually conducted analysis and scope measurement activities, the quality and efficiency of the service Turkcell provides for its individual and corporate customers will increase. Within the scope of the study, it is aimed to automatically calculate the CFP (Cosmic Function Point: Cosmic Function Point), which is used as a software scope measurement method.

The BUMBLE features covered by this use case are:

- Blended Syntaxes and Modelling (B): In this project, it is aimed to automatically measure the CFP used in the analysis documents prepared for the Turkcell Academy software product within the scope of the project.
- Collaborative Modelling (C): Analysts in different teams or in the same teams use the Confluence application while preparing their analysis documents. Therefore, we aim to exploit the features of the BUMBLE project that facilitates collaborative modelling, particularly on the textual part of the blended language.

- Traceability (T): In this project, it is aimed to automatically measure the CFP used in the analysis documents prepared for the Turkcell Academy software product within the scope of the project. It is aimed to express free text or semi-structured requirements in formal languages. In this way, it will be possible to express free text or semi-structured requirements in official languages.

3.13.1 Core Requirements

ID	Classification (B, C, E, T, N)	Description of Requirement
C13.1	B	Changes in specification rules of a given software analysis document model should update the textual notation without losing non-semantic information it contains, such as comments, indentations, and other white-space characters.
C13.2	B, T	References in the textual specification that refer to other elements defined outside the clearance rule specification should be bound to the correct model element.
C13.3	B	Semantic checks should be implemented which detects semantically incorrect models which the textual syntax permits creating.
C13.4	B	Changes in the state machine model should update the graphical notation without losing non-semantic information it contains, such as colours, symbol sizes, line routing and other layout information.
C13.5	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.

3.13.2 Technical Requirements

ID	Classification (B, C, E, T, N)	Description of Requirement	Details Core Requirement(s)
T13.1	B, T	The tool should enable to calculate CFP values according to use case items.	C13.2
T13.2	B	A textually defined Design Rule should be persisted using the textual syntax, while other parts of the model be persisted as XMI.	C13.1
T13.3	N	The Design Rules should be validated against graphical Data.	C13.3, C13.4