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 D2.1

Use Cases for Blended Modeling

Edited by: BUMBLE Team
Date: July 2021

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

Contents

Acronyms

2D	Two Dimensional
3D	Three Dimensional
API	Application Programming Interface
AST	Abstract Syntax Tree
BPM4DCA	Business Process Management for Debt Collector Advocates
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CPU	Central Processing Unit
CR	Change Request
CRUD	Create, Read, Update, Delete
DB	Data Base
DCA	Debt Collector Advocates
DSL	Domain-Specific Language
DSML	Domain-Specific Modeling Language
ECU	Electronic Control Unit
EMF	Eclipse Modelling Framework
EN	European Norm
FAA	Federal Aviation Administration
FDA	Food and Drug Administration
FTNCHEK	Fortran Checker
GEF	Graphical Editing Framework
GLSP	Graphical Language Server Platform
GMF	Graphical Modelling Framework
HDA	Housing Development Account

HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LSP	Language Server Protocol
MARTE	Modeling and Analysis of Real-Time and Embedded systems
MOF	Meta-Object Facility
MPS	Meta-Programming System
OEM	Original Equipment Manufacturer
OIL	Open Interface Language
SBVR	Semantics of Business Vocabulary and Business Rules
SMS	Short Message Service
UC	Use Case
UI	User Interface
UML	Unified Modelling Language
UML-RT	UML Real-Time
UX	User Experience
VCS	Version Control System
WP	Work Package
WPF	Windows Presentation Foundation
XML	Extensible Markup Language

1. Introduction

This document defines the use cases for blended modelling identified in BUMBLE. The purpose of these use cases is to support the elicitation of requirements for the technologies to develop: Based on the refined use cases, a set of top level functional and non-functional requirements will be derived. If necessary, a further breakdown into low-level requirements with finer granularity is made. The resulting requirements specification (Deliverable D2.2) is used as input for the concept and implementation WPs in BUMBLE, more specifically: WP3, WP4 and WP5. After realising the use cases based on the developments in those WPs, the BUMBLE technologies will be evaluated by exercising the use cases and assessing the validity and utility of the outcome.

BUMBLE identifies use cases addressing two different kinds of areas:

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

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

Table 1. BUMBLE Use Cases

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

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

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

1.1. Structure of this Deliverable

This deliverable presents each use case according to a common structure as follows:

- **Background**: Explains the domain / engineering context from where the use case originates.
- **Use Case Description**: Detailed description of the use case including user stories.
- **Bumble Features**: Description of how BUMBLE technologies are expected to contribute to realising the use case. This gives a high-level overview of technical needs to be addressed.
- **Demonstrator**: Describes how the use case and hence relevant BUMBLE technologies will be demonstrated.

2. UC1 - Software Open-Source Blended Modeling

2.1. Background

This use case is intended to provide the main public demonstrator of a full-fledged open-source blended modeling framework conceiving the core features of BUMBLE. Starting from a set of experiments run on a simple prototype for blended modeling of UML and MARTE, we assessed the potential benefits of providing for instance a textual notation for UML state-machines, but also for hardware platform modeling and software-hardware allocation in MARTE. In this use case, we will target multiple DSMLs and build upon previous works on UML state-machines and UML profiles in order to demonstrate a full-fledged blended modeling toolchain in open-source. Besides individual blended modeling, we will also provide and demonstrate collaborative features.

While the other use cases are driven by industrial actors, this use case is intended to enclose the set of open-source solutions provided by the academic actors, supported by industry. The rationale is that this use case will represent the public channel for BUMBLE to disseminate and demonstrate the project results to a broader audience of both researchers and practitioners. The use case will cover all major project outputs planned in BUMBLE.

2.2. Use Case Description

Starting from a DSML, the 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 framework will need to support model synchronization mappings between the DSML and the generated notations.

Given the DSML, the generated notations, and the model synchronization mappings, the framework is expected to semi-automatically generate synchronization mechanisms between notations and DSML and co-evolution transformations. In addition, the framework should provide an API to access the elements of the abstract syntax tree in order to enable traceability to model elements independent of the concrete notation in which the model is edited.

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

Eclipse and MPS will be exploited as base modeling and development platforms. Additional technologies will depend on the needs of the industrial use-cases, since we aim at maximizing the effort on a minimal set of technologies that allows us to cover as many use cases as possible. We will leverage at least three concrete DSMLs: EAST-ADL and RTist's UML-RT in Eclipse and OIL in MPS. We will investigate the possibility to provide Ecore-based language exchange between Eclipse (EMF) and MPS too. The mechanisms for Ecore-based language exchange will operate both at the modeling (M1) and metamodeling (M2) levels of abstraction.

2.3. BUMBLE Features

The BUMBLE features covered by this use case are:

- Automatic generation of blended modelling editors for MOF-based DSMLs to support blended graphical-textual modelling.
- Semi-automatic generation of synchronization mechanisms across notations.
- Automatic synchronization between multiple editors/notations and support for seamless collaborative editing.
- Support for semi-automatic co-evolution of generated artefacts in response to evolution of the original DSML (e.g., a change in the metamodel will be reflected also on the blended modeling editors generated from the metamodel).
- Automatic generation and maintenance of representation-agnostic traceability links in situ for synchronization and co-evolution purposes.

2.4. Demonstrator

The demonstrators will cover all features mentioned in Section 2.3 in relation to the core aspects of the industrial use-cases. More specifically, we will demonstrate the following:

- From a given DSML, a blended editing environment (including editors and synchronization mechanisms) is automatically generated.
- Once a model is created in the blended environment, it can be opened and edited using multiple notations/editors (e.g., textual, graphical, tabular, etc).
- A model change in one of the editors is seamlessly reflected to the others. Change propagation and synchronization can be either on-demand or on-the-fly, upon user’s choice.
- A model can be viewed and edited in real-time in a collaborative fashion by multiple users. Versioning and diff/merge features are handled in a GIT-based fashion.

We will provide one demonstrator per target platform -- Eclipse and MPS.

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

3.1. Background

The most important modeling language used in HCL RTist is UML RealTime (UML-RT), described by the UMLRealtime profile (it's actually a very small subset of the full UML, as described here). Some other DSMLs are also used in the tool, for example to describe how the models are transformed into C++ code. But for BUMBLE it's UML-RT that is important, and more specifically the part of UML-RT which describes state machines.

Our customers use RTist to design and develop various kinds of real-time applications implemented in C++. This can be anything from embedded systems deployed on micro controllers to distributed systems deployed on powerful servers, and anything in between. UML-RT allows our users to design their applications at a higher abstraction level compared to doing it in plain C++. It also provides automation so that a large part of the C++ code can be automatically generated from the UML-RT model.

The UML-RT language is at the very core of RTist and users create, view, and edit these models using RTist. However, often, not all parts of the application benefit from the abstractions of UML-RT. Therefore, it is very common to combine the generated code with handwritten C++ code. Both generated and handwritten code then gets compiled into the final executable or library. Also note that RTist uses C++ as action and expression language inside the model. This means the UML-RT model contains embedded snippets of C++ code, for example to define the entry action of a state, or the guard condition of a transition.

3.2. Use Case Description

Our BUMBLE use case is about letting users create, view, and edit state machines of their UML-RT model using a textual syntax, as an alternative to the current graphical notation. We believe this will be useful in several scenarios:

- For an experienced user, that has learnt the textual syntax, it can be much faster to work with a state machine in a text editor as opposed to using the current graphical editors.
- Some editing, for example state machine refactoring, will benefit from standard text editor features such as copy/paste, incremental find etc.
- When state machines are used for modelling test cases it is convenient with a textual notation since test cases often need to be updated to match changes in the application behaviour. Many small updates are easier to do textually than graphically, and it is common to copy/paste contents from one test case to another.
- When comparing or merging a state machine it can be easier to understand changes or conflicts using a textual notation as a complement to graphical diagrams.
- A textual state machine notation can be a first step of letting RTist support more IDEs than just Eclipse. Text editors are readily available in all IDEs while graphical editors tend to be rather specific to a certain IDE.

HCL owns this use case. Participation is welcome from any BUMBLE partner with similar interests. Canon is an experienced RTist user and is considering the opportunity of providing early feedback on the solution's usability for working with industrial models (we may also involve other non-BUMBLE customers that we think are interested). An important part of the work is to define a good textual state machine syntax (unfortunately, there is no already established syntax that can be reused for this). We think MDH can contribute here with their experience of XText and related technologies.

There is currently no support for textual state machines in RTist, but we have done some experiments and prototypes in the past. A long time ago (in a previous product), we had a feature allowing state machines to be edited textually. From that we learned the importance of giving a "true" text feeling when editing. The user's indentations and comments must be preserved exactly as written. We also learned the importance of using a syntax that is easy to learn and that is not too "exotic" compared to other syntaxes the users are used to working with. Finally, it is important to preserve the identity of model elements when they are being edited in a textual syntax. Links to the edited elements should not become easily broken. We have automatic layout implemented for our state machine editor, although it certainly could be improved.

Currently in RTist, we have an editor (a transformation configuration editor) which combines usage of form-based editing with a textual syntax (JavaScript). We have some of the problems mentioned above also in this editor (like comments and formatting that get lost when editing in the forms). We've also learned the importance of providing features such as code completion (a.k.a. content assist) to make text editing more productive and less error prone.

3.2.1. User Stories

- As a DSML user, I want to have a textual syntax which covers all features of UML-RT state machines, so that I can define a state machine completely in a text editor without having to also use other notations.
- As a DSML user, I want standard text editor features such as code completion, navigation, source formatting etc, to increase my productivity.
- As a DSML user, I want the textual syntax to include C++ code snippets (used for action code and expressions), integrated in a seamless way (for example, it should not be necessary, or at least very rarely, to use any kind of escape characters in the C++ code).
- As a DSML user, I want syntax colouring both for the state machine syntax and the embedded C++ code snippets for increased readability.
- As a DSML user, I want a command for navigating from the textual syntax to the generated C++ code from the UML-RT syntax.
- As a DSML user, I want a command for navigating from a line in an embedded C++ code snippet to the corresponding code line in the generated C++ code, so that I can use features such as code completion and navigation at C++ level.
- As a DSML user, I want commands for navigating from the textual syntax to the Project Explorer and state machine diagram.
- As a DSML user, I don't want a small syntax error in a textual state machine definition to prevent anything in the text file from being successfully parsed. Parsing should try to recover from errors, and existing model elements should not be deleted from the model just because of transient errors in one of the notations.

3.2.2. Non-Functional Requirements

The textual state machine syntax should be easy to learn and use. It should be an "Algol-style" syntax to look familiar for users familiar with other languages used in RTist (C++, JavaScript, Java).

The usability of editing state machines textually should be the same as when working in any other text editor. User formatting, comments etc. should be preserved (to an as large extent as possible) when the textual representation needs to update because the model is changed.

Performance should be good and there should be no noticeable delays when editing a state machine in the text editor.

If it turns out the current GMF editor needs anything but small changes, new web-based editor frameworks should be considered to make it easier in the future to support non-Eclipse IDEs.

Textual editing should not break incoming links to edited model elements as it's poor usability to have to recreate such links all the time.

3.2.3. Current baseline of tools and technologies

The baseline of RTist to be compared with is version 11.0 2020.50 (using Eclipse 2019.06 with OpenJDK 8). The model is the standard open source UML2 model, version 2.3 (i.e. not the latest version of UML). It is an EMF model. Graphical editors are implemented using GMF and GEF.

3.3. BUMBLE Features

3.3.1. Blended Syntaxes and Modeling

It should be possible to freely choose between the textual and graphical notation when creating or editing a state machine. Changes in one notation should be synchronized so other notations update automatically (in a way that feels natural and non-intrusive for the user).

Note: For our use case, automatic generation of editors is not of big interest since in our experience usability of such editors tends to become too poor. The metamodel used in RTist is very stable (UML-RT is more or less unchanged for the past 15 years) so we don't expect a need to handle an evolving metamodel (if needed it will anyway have a huge impact that requires manual modifications in many places).

An XText-based editor should be implemented which implements a textual state machine syntax. The concrete syntax used in the editor will be mapped to the existing UML2 metamodel that is currently used in RTist. The Compare/Merge editor of RTist will also be extended to provide a new view that shows changes and conflicts in state machines using the textual syntax in addition to the graphical, tree and tabular views currently offered.

The XText-based editor parses the state machine syntax continuously as soon as the user stops typing for a while. It builds a state machine UML2 model as its AST. This model is then merged with the RTist model to which it belongs so that changes made in text get reflected in the model. This is synchronization from text to model. If the user changes the state machine model in RTist (using some other view than the text editor) then synchronization in the opposite direction will have to take place (model to text). The demonstrator (see section 3.4) will show some use cases that involve both kinds of synchronization. The overall goal is to show that the synchronizations are as precise as possible, leaving the model always in a consistent state and with as few negative side-effects as possible:

- For text to model synchronization all textual references in the parsed text should be attempted to be bound. There are both internal links (where the target element is inside the parsed model itself) and external links (where the target element is somewhere else in the RTist model). Both types of links should be bound and enable navigation to the target element.

  Example
  The state machine snippet
  t1: State1 -> State2 on timer.timeout;
  contains 2 internal links (State1 and State2 - assumed to be declared previously in the file) and 2 external links (timer and timeout - assumed to be possible to look up from the context of the capsule that owns the state machine in the RTist model).

- For text to model synchronization incoming links to elements in the RTist model must be preserved even if those elements get replaced during the merge with the parsed model.

  Example
  A dependency from a capsule to a state in its state machine should not become unbound when editing the text file. However, if the state itself gets deleted or renamed the dependency will get unbound, but if that state later reappears (e.g. by renaming another state) then the dependency will get bound again.

- For model to text synchronization white space, comments and formatting should be preserved to an as large extent as possible.

  Example
  Deletion of a state outside the text editor (e.g. in the Project Explorer) requires a serialization of the state machine model and a merge of the resulting text with the text in the file.

To avoid conflicting modifications in these synchronizations it is important that they happen instantly (or at least very shortly after an edit). In some cases, this may not be possible, for example if deleting a state when the text editor with its definition is not open. In this case, synchronization must happen immediately when the text editor opens before the user has a chance to make any modifications himself in the text.

3.4. Demonstrator

A demonstrator will show how a state machine can be defined textually in RTist. A simple state machine will be created from scratch, using the content assist feature to simplify typing. The model will then be built with the model compiler, and the generated executable will be run, to show that the textual state machine behaves identically to a graphical one.

We will also show several scenarios that require synchronization, as mentioned in the previous section. For example, changing a textually defined state in the Properties view, adding new states using the Project Explorer etc.

Finally, we will show how a textual version of a state machine can help in a scenario of merging two state machines. In addition to the graphical, tree and tabular views that RTist currently uses, the textual view of a state machine can make it easier to merge the state machine in the same way as other text documents are merged.

4. UC3 - Vehicular Architectural Modeling in EAST-ADL

4.1. Background

Development of automotive embedded systems at AB Volvo involves large amounts of data from multiple stakeholders, including requirements, specifications, log data, components, code, binaries and so on. To organize this data efficiently and ensure that syntax and semantics of the content are consistent, a metamodel is required. On such a basis, engineers work with a standardized representation, content is machine readable for automated engineering tasks, and Continuous Integration (CI) pipelines can use data in a non-ambiguous way.

Autosar and EAST-ADL are architecture description languages for automotive embedded systems. Both are based on an infrastructure where an M3 metamodel provides rules for the definition of the AUTOSAR and EAST-ADL languages, and Eclipse tooling generates XML schemata and editor infrastructure for basic editing and serialization. Models are stored in XML format as *eaxml* and *arxml*, and tool platforms are EATOP and Artop respectively. Since EAST-ADL complements AUTOSAR, all results for EAST-ADL in BUMBLE will also align with AUTOSAR, even though AUTOSAR's intellectual property protection complicates working with the language directly.

The role of software and system architecture description languages is to identify common data between different parts of the organization and define its representation. Such representation secures non-ambiguous, complete, and consistent information and allows tooling for analysis and synthesis to act on harmonized input.

BUMBLE technology is anticipated to provide multimode editors for the EAST-ADL metamodel (and indirectly for Autosar, due to shared technology). The editors are expected to support tree-based, textual, and graphical editing. Currently, EATOP uses Ecore and Sphinx to a) (de-)serialize *eaxml* to conform with industry standards for data exchange and b) generate a tree-based editor and the storage layer for EAST-ADL data. It is expected that this existing infrastructure is reused in BUMBLE where appropriate. Alternatively, a new infrastructure is set up to provide editors with serialization capability.

4.2. Use Case Description

AB Volvo expects that BUMBLE provides new frameworks to generate editors for graphical, textual, and tree-based editing of EAST-ADL models.

4.2.1. User Stories

To characterize this use case, three example user stories are used. The detailed capabilities and functionalities support these user stories and more. More user stories are given in Section 4.2.5.

Graphical Editing

- As a DSML user, I want to be able to graphically view and edit system descriptions represented as *eaxml* files, so that I get a good overview of my system.

Textual Editing

- As a DSML user, I want to be able to see system descriptions represented as *eaxml* files as plain text, so that I can efficiently edit, diff, and merge my system description.

Views and Viewpoints

- As a DSML user, I want to be able to work with partial views of my system descriptions represented as eaxml files, so that I can focus on relevant parts of a large information set.

4.2.2. Current Status and Existing Functionalities

Currently, AB Volvo uses EATOP to model EAST-ADL models, mostly with tree-based editors, but also with graphical editors for different, specific viewpoints. EATOP uses Ecore and Sphinx to a) (de-)serialize eaxml to conform with industry standards for data exchange and b) generate a tree-based editor and the storage layer for EAST-ADL data. The following describes the existing functionality.

Define Metamodel According to MOF Subset defined as AUTOSAR M3
The meta-model is currently defined as a UML model in Enterprise Architect based on MOF M3 and uses some constraints like:

- Only element allowed is *Class*.
- Enumerations can be defined with <<*enumeration*>> stereotype.
- Only relations allowed are associations and compositions.
- Associations may be marked <<*isoftype*>> and <<*InstanceRef*>>.

Generate XML Schema for Model Exchange
This schema defines how the model is serialized. The schema is currently generated by the EATOP Metamodel generator org.eclipse.eatop.metamodelgen. The generator takes information from the EAST-ADL meta-model and wraps it in schema entries that provide additional structure. This ensures that the schema for eaxml files conform to industry standards for information exchange in the automotive industry.

Generate Tool for Tree-Based Editing
The existing tree-based editor and the persistence layer is currently generated by the EATOP Metamodel generator org.eclipse.eatop.metamodelgen.

Generate Tool for Graphical Editing
There are limited graphical editing capabilities available. A graphical view of the model is currently available that supports drag-and-drop of existing entities and which shows their relationships, in particular ones of type <<*InstanceRef*>> (org.eclipse.eatop.volvo.sgraphml.gefeditor). This editor is only capable of renaming entities and has no other capabilities to edit the underlying model.

In addition, there are view-specific editors (org.eclipse.eatop.examples.graphicaleditor). These editors provide function modelling and safety modelling viewpoints among others but are limited to these specific viewpoints. In addition, these editors have dependencies to plugins that are no longer maintained.

Generate Tool for Text-Based Editing
Text-based editing is currently only available for XML, which is not efficient and readable. ARText is a textual editor for an AUTOSAR subset that can serve as an example for a future text-based editor for EAST-ADL models.

4.2.3. Desired Functionalities to be Provided by BUMBLE

General Requirements for Editors

- 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.
- 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, realizing a certain feature, part of a certain variability configuration, active in a certain mode, etc.
    In adding elements in such a view, the model will be updated such that the new element complies with the criterion. For example, the new element may be allocated to the ECU, realize the Feature, be part of the variant, etc.
- 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 the textual editor shall also be separated.
- It should be possible to create models in the editor that do not fully conform to the meta-model in order to ensure rapid prototyping and evolution of content.
- 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.

General Requirements for (De-)Serialisation

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.

General Requirements for Tree-Based Editing

The tree-based editor shows all elements of a model using the metamodel element hierarchy in packageable elements to structure the information.

Views shall be possible to define based on information subsetting, i.e. only a subset of model content is exposed according to criteria defined by the user or pre-defined by the editor (e.g., to only show elements in a specific package of the meta-model such as timing or variability).

The order of elements in the underlying model can be changed by dragging elements into a different order in an unsorted view.

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.

General Requirements for Textual Modelling
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 the package for anything added. For example, packages with datatypes or other elements may be imported and subsequently visible and part of the scope.

General Requirements for Graphical Modelling
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).

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.

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.

Deleting anything in a diagram is primarily about deleting from the diagram canvas. If an element shall also be deleted from the model, it must be done explicitly, e.g., by right clicking or ctrl-deleting. This is because a user may want to customize the viewpoint and include/exclude elements depending on the purpose of the diagram.

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.

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.

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.

General Requirements for Diffing and Merging
There should be functionality for diffing and merging of EAST-ADL models to support collaborative modelling of different team members. 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.

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.

General Requirements for Multi-User Support
Ideally, multi-user editing should be supported, even though these requirements have low priority.

It should be possible to define access and editing rights for different stakeholders that are automatically enforced by the tooling in order to limit users' ability to see certain parts of the model or change certain parts of the model.

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).

Even if multi-user concurrent editing is available, it should still be possible to diff and merge a model that has been modified offline with a model that has been concurrently edited in order to support engineers that have been working on the model without access to the concurrent editing environment.

4.2.4. Mapping of Language Elements to Editors

Not all language elements need to be edited in all editors. Certain language elements lend themselves better to modelling in a certain type of editor. The table below details which language elements should be edited where.

What to Model	Tree	Textual	Graphical	Details
Element	X			Anything in the tree view should be possible to sort and subset
Element	X			Anything in the tree view should have a property window where its attributes can be seen and edited

What to Model	Tree	Textual	Graphical	Details
Element	X	x	x	Anything in the tree view should have a property window where its attributes can be seen and edited
Types	X	X	X	Anything with an isoftype relation shall exhibit the properties of its type
Components and Compositions (FAA, FDA, HDA, ErrorModel, ...)	X	X	X	Parts typed by Types will exhibit the ports and properties of the type
Connectors		X	X	Connectors that connect port groups shall be collapsed to one entity
Connectors			X	Connectors shall be possible to hide using goto blocks
Allocation			X	Allocation shall be visualized using relations or containment
Packages and packable elements			X	Package containment can be shown as a circumventing box or as line with filled diamond
Elements and composite elements			X	Element containment can be shown as a circumventing box or as line with filled diamond
Attributes			X	Element Attributes are contained but should be shown as shortname: value on the element rather than as a line with filled diamond.
References			X	Lines between any element in a diagram
References		X		Packagepath string to elements
InstanceReferences			X	Lines to specific occurrences of parts in a diagram
InstanceReferences		X		Prototype path string to parts
InstanceReferences	X			Prototype path string to parts and tree-based editing of instanceref

What to Model	Tree	Textual	Graphical	Details
Reference Navigation	X			It shall be possible to navigate from one element to any of its referenced or referencing elements
Instance Reference Navigation	X			It shall be possible to navigate from one element to any of its instancereferenced or instancereferencing elements
“Reference Element” Navigation	X			It shall be possible to navigate from one element to any of its referenced or referencing elements also when “reference elements” are used (associations where a containing entity identifies the source and target)
Element Navigation		X	X	It shall be possible to navigate from a textual or graphical entity to its corresponding model entity in a tree view. Alternatively, to copy its package path or instanceref path.

4.2.5. User Stories Illustrating Typical Engineering Tasks

The following user stories illustrate typical tasks an engineer would do with the EAST-ADL editors generated by the BUMBLE framework. Therefore, they refer to concepts in the EAST-ADL language. Please note that EAST-ADL uses a prototype-based form of inheritance.

Allocate Functional Components to Hardware Components

- A user selects one or several function (proto-)types.
- The user can now assign it to one hardware (proto-)type (i.e., a representation of a physical hardware component).
- These prototypes are navigable and selectable as a prototype path in a tree view.
- A semantic check may warn about invalid allocation decisions (e.g., a prototype and its type being allocated to different nodes).

Connect Ports of Parts in Component Diagrams

- A user selects two ports in a (composite) type.
- If the two ports are located on contained prototypes, an assembly connector is created. Otherwise, a delegation connector is created.

Auto-Connect Ports of Parts in Component Diagrams

Selecting two or more parts should allow an “auto-connect” action, where ports of matching name and/or type should be connected.

- A user selects two or more prototypes in a composite type.
- A connector is created for each matching port pair.
  Matching means that either:
    - Names are matching (near-match or precise match could be chosen).

    - Types are matching (same type, same type name, compatible type could be chosen).
    - Send - receive on two prototypes results in assembly connector.
    - Send - receive on the composite type results in a delegation connector.
    - Send - send or receive - receive on the composite type and prototype results in a delegation connector.

Collapse and Explode Components

- A user selects two or more prototypes in a composite type.
- On selecting "collapse"
    - a new type is created with all the parts of the two or more prototype's types.
    - a new prototype is created with its type set to the new type.
    - connectors are re-connected to the new prototype.
    - (possibly other relations are also transferred to the new prototype).
- On selecting "join"
    - a new type is created with the two or more prototypes inside and existing assembly connectors.
    - ports are added to the new type as well as delegation connectors to the contained prototypes.
    - a new prototype is created with its type set to the new type.
    - connectors are re-connected to the new prototype.
    - (possibly other relations are also transferred to the new prototype).
- A user selects one prototype in a composite type.
- On selecting "explode"
    - the type of the selected prototype is copied.
    - a new prototype is created which is typed by the new type.
    - connectors of the selected prototype are connected also to the new prototype.
    - (now the user can start to delete ports and connectors).

Auto-Edit Ports and Connectors

- A user selects one port on a prototype in a composite type.
- On dragging-dropping the port on another prototype, the corresponding types are updated, and the connector follows to the new prototype.
- A user selects two ports and selects "Connect".
- If ports are in the same composition an assembly or delegation connector are created.
- When connecting a port across a hierarchy, the ports and delegations should be automatically added to the parts.

Synchronize Content between Abstraction Levels

- Two compositions are linked with a *Realize* relation.
- A user adds or removes elements in either of the compositions.
- The corresponding elements are added or removed in the synchronized composition, subject to approval of the engineer.
- On adding components, corresponding *Realize* relations are added.
- Synchronization results can be visualized in tree, text, or graphical view.

Create and Connect Component Prototype Based on Available Ports

- A user selects a type and places it in another type.
- A prototype that is typed by the type is added.

- Ports are automatically connected based on port names and types.

4.2.6. Current Baseline of Tools and Technologies

Tool and technologies related to the EAST-ADL/Autosar workflow include

- Eclipse EATOP Metamodel Generator.
  Based on a metamodel expressed in a UML subset using Enterprise Architect, this too generates
  - XML schema.
  - Java code for EATOP plugins for (de-)serialisation, persistence and basic tree editing of EAST-ADL models.
    EATOP supports splitting models over multiple files and the editing/navigating of models, in particular for the type/prototype pattern of EAST-ADL/Autosar.
- Eclipse EATOP Plugin.
  Plugins on the EATOP platform use the EAST-ADL model in ECORE for analysis and synthesis. There is also a GEF-based plugin to allow a graphical view of models.

BUMBLE solutions would have to interface to the EATOP platform in order to capitalize on the infrastructure for (de-)serialisation and basic tree editing. On the other hand, if another technology can provide the same capabilities without EATOP, it is enough to respect the EAST-ADL metamodel (expressed in any UML tool) and *eaxml* (de-)serialization format.

4.3. BUMBLE Features

4.3.1. Blended Syntaxes and Modeling

It is expected that the architecture model can be edited both in tree view, textually and graphically, and the same model element may appear in any of these representations.

4.3.2. Collaborative Modeling

Collaborative can be supported by allowing parallel edits of the same model. In such cases, the tooling needs to support diff and merge of the edits, without corrupting the model. Concurrent editing is a nice feature, but not as important as diff and merge.

4.3.3. Evolution

The metamodel will change and multiple versions of the metamodel may be used interchangeably. In such cases the tooling should make a best effort mapping of content to the chosen metamodel. Possibly, the mapping can be based on mapping rules between original and new metamodel.

4.3.4. Traceability

Traceability within the model is represented by various relations, such as Realize, Verify and is-of-type. Tooling support is helpful to navigate the model back and force along those relations. Further, the associations can be used to create various views.

4.3.5. Model Non-Conformance

The tooling may warn about non-conformance of models vs. the metamodel. Such deviations are inevitable as content evolves, but identifying non-conformance, possibly with correction suggestions, is a helpful feature.

4.4. Demonstrator

A demonstration at a review would show a main scenario where the BUMBLE capabilities to handle model files and edit them in different modes is demonstrated:

- Open model file in tree editing mode
- Open model file in textual editing mode
- Make edits and save
- Open model file in graphical editing mode
- Make edits and save
- Show changes in tree view

On-the-fly synchronization is beneficial but optional. However, while editing in a graphical editor, changes shall appear in the textual editor after refreshing or reopening the model, and vice versa.

In addition to the basic capabilities, a selection of engineering tasks could be demonstrated. Further, some evidence of meeting the requirements could be demonstrated.

5. UC4 - Cross-Disciplinary Coupling of Models

5.1. Background

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. Specifying the software to perform the media handling is a core activity to achieve a productive and reliable digital cut-sheet printer. 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 underperformance.

We have built several JetBrains MPS DSMLs that focus on: (1) capturing/importing the 2-dimensional nominal paper path layout and points of interest in the layout of printer modules, (2) specifying the allowed variability within a product family, along with the concrete configurations, and (3) specifying the functional usage of the paper path, such as routes, timing behaviour of a single sheet, functional timing constraints between subsequent sheets.

This collection of linked specifications can then be used to generate (part of) the real time embedded software, as well as artifacts for early analysis and visualization of the specifications. These are coupled to sheet-flow simulations with different levels of integration of the final embedded device control software.

These DSMLs are starting to show their value by consistently supplying a high-quality media handling software component that is integrated in the embedded device control. The mechanical engineers and function designers (for the print process, fixation (heating), and cooling) provide and review information about how the media should be transported. Keeping the notation close to the familiar domain notation is therefore essential to the speed at which iterations of the product development are performed.

5.2. Use Case Description

5.2.1. Current Status and Existing Functionality

The DSMLs mentioned are being implemented in JetBrains MPS and are becoming quite mature. The DSMLs are being adopted by the software developers in the media handling component, but not yet by the mechanical engineers/function designers. The software developers can import CAD/CAE models through MPS, visualize the paper path layout, and specify the variability and functionality at an abstract level. The models are then combined for different printer configurations and generated into C++ and XML artifacts used to compile and configure the media handling component of the embedded device control software.

5.2.2. Desired Functionalities to be Provided by BUMBLE

A collaborative environment (preferably web-based) that allows reusing the projective editor definitions of our DSMLs in MPS.

5.2.3. User Stories

The following user stories detail the kinds of actions and benefits envisioned by a collaborative blended environment. It is assumed that only one version of each language is deployed at a time and accessible through the website.

Modelling and Model Management

- As a modelling user, I can login to a website, so that I can identify myself and get access to a blended collaborative modelling environment.
- As a modelling user, I can navigate the existing models on a website, so that models can be found with a low threshold.
- As a modelling user, I can manage (CRUD) a hierarchy/organisation of models (folders/packages, as well as model roots), to achieve a maintainable organisation of the modelling content.
- As a modelling user, I can tag model versions, so that they can be used as snapshots for later reference.
- As a modelling administrator, I want to set access levels for models and packages, so that these models and packages are visible/readable/writable by a particular set of users.
- As a modelling user, I can generate/download/deploy modelling artifacts, so that modelling artifacts can be used outside of the modelling environment.
- As a modelling user, I can start/perform analysis on a model, to check for the model for certain properties (correctness, performance, etc).
- As a modelling user, I can see errors and feedback (if any) in the model editor, so that I can quickly identify issues in the model.
- As a modelling user, I can see an overview of errors and feedback (if any) in an overview, so that I can quickly identify issues in the project.
- As a modelling user, I can follow a modelling reference (hyperlink), so that I can easily navigate the relationships between models.

Blended Modelling

- As a modelling user, I can view the model through my selected projection, so that I can simplify/extend the information shown in the model based on my needs.
- As a modelling user, I can see my model in multiple (at least two) views, with different projections, so that I can focus on the structure and particular details at the same time.
- As a modelling user, I can use textual syntax (with highlighting, completion, cross-referencing) within a graphical (diagrammatic/tabular) model.
- As a language engineer, I can set the default view of a model (entity) to a particular projection, so that I can simplify/extend the information shown in the model based on my needs.
- As a modelling user, I can edit text, tables, diagrams, and forms in my model, so that I have the freedom to choose the most effective representation.

Model Collaboration

- As a modelling user, I can see the current state of the model when I am connected to the modelling environment, so that I am always up to date.
- As a modelling user, I can retrieve and export models from external sources (like Git), so that I can collaborate with external versioning systems.
- As a modelling user, I can apply (free-form text) reviewing annotations to the model, so that we can review and track progress.

- As a modelling user, I want to see which users have the model open, to improve communication and avoid modelling conflicts.
- As a modelling user, I can see the mutation history of a model, so that the differences over time can be viewed.
- As a modelling user, I can select model versions in the mutation history, so that I can compare the current model to the old model.
- As a modelling user, I can resolve merge conflicts, so that the models remain in a consistent state.
- As a modelling user, I can use a notebook-style view on my models, so that I can mix the content with the description/documentation.
- As a modelling user, I can perform undo actions inside a model, so that I can undo my own changes.

Integration

- As a modelling user, I want to instantiate a template for new (related) models using a web-based wizard, so that creation of new models is low-effort.
- As a language developer, I want to 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.
- As a modelling user, I want to (incrementally) import (i.e., uploaded by me, or retrieved from a server) data from a CAD/CAE repository, so that the external relationships can remain up to date.
- As a language engineer, I want to 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.
- As a modelling user, I want to visualize (interactively, inline, or in an external window) the results of the modelling artifacts, to achieve a smooth integration between the specification and the visualization.
- As a modelling user, I want to 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.
- As a language engineer, I want to integrate model editors with web-based components, so that I can create simplified workflows.

5.2.4. Non-Functional Requirements

- We expect the collaborative modelling environment to provide a very low threshold for modelling by the end user, while providing a mature modelling front-end with high usability. For example:
    - No or little installation required for end user (through, for example, a web front-end).
    - Very quick feedback on viewing models and model editing actions.
    - Very little clutter around the actual modelling view and editing capabilities.
    - To the point and easy-to-use user interface that performs well in collaboration between engineers (low latencies etc). Note: we do not exclude the need for using non DSML technologies in combination with MPS-based technologies to achieve such easy-to-use user interface.

5.2.5. Current Baseline of Tools and Technologies

We are using JetBrains MPS to define our DSMLs and edit our models. We use the IETS3 (KernelF) and mbeddr plugins from itemis A.G., as well as their shadow model transformations. The collaborative environment provided by BUMBLE should allow access to the modeling features of MPS (so explicitly not the language development features).

5.3. BUMBLE Features

5.3.1. Blended Syntaxes and Modeling

It is expected that the models in the DSMLs can be edited in their provided notations in MPS. It should be easy to mix notations and switch between notations (even within a single model instance) where relevant.

5.3.2. Collaborative Modeling

Multiple users can view and edit the same model in different notations, while immediately syncing parallel edits.

5.3.3. Evolution

It should be possible to migrate to newer versions of the DSML. We expect to use only one deployed version of the DSML/metamodel at a time for all users.

5.3.4. Traceability

It should be possible to trace relationships between the multi-disciplinary models. I.e., from CAD/CAE to linked functional specifications, to parts of the generated embedded software component.

5.3.5. Model Non-Conformance

We do not expect to require features regarding Model Non-Conformance. The default behaviour of MPS suffices, where we do not expect that the projective editor allows applying non-conforming changes to the models.

5.4. Demonstrator

The following demonstrators are expected for the reviews, to show the BUMBLE capability of creating and updating models with multiple users collaboratively, in multiple notations. The demonstrator will be based on the MPS-based DSMLs described above:

- 2nd review
    - Editing the default textual projection with rich features such as auto-completion, cross-referencing.
    - Near-immediate synchronisation of models between multiple users.
- 3rd review
    - Importing paper path layouts for printer modules.
    - Editing the graphical diagram of paper path layouts.
    - Generating a piece of embedded software, which allows visualizing the sheet movement behaviour in an (external) visualization tool.
    - Exporting the generated embedded software to a VCS like Git.

6. UC5 - Reactive and Incremental Transformations across DSMLs

6.1. Background

The Modelling Value Group has customers in different domains. Therefore, the Modelling Value Group has developed many DSLs for different purposes. Examples are: Insurance products, business logic, decision support, document structures and diagnostic knowledge. Those languages are used by groups of domain experts in the organisation that work together to maintain the models. In all cases, we provided means to validate, run, test, or simulate the models in the modelling environment itself. On top of that the models are transformed into software that is executed in production environments. The modelling-workbenches are built using either EMF (Eclipse) or MPS, extended with our own domain-independent libraries like Dclare.

6.2. Use Case Description

Since the Modelling Value Group is a tool-builder and not limited to any specific domain, we will define a use-case that is more based on the requirements and context of other members of the BUMBLE consortium.

The use-case combines collaborative and blending modelling of two different state-transition modelling-languages that are transformed and synchronized immediately. The modellers (the DSML users in the uses-case) can change their models in different network locations and can view and edit their models in their own preferred syntax, yet still be able to edit the models together. Changes made by one user are immediately visible by other users. The rationale behind this use-case is that it combines two major goals of the BUMBLE project: blending and collaboration.

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

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

The use-case is owned by the Modelling Value Group. We are using models and examples from other BUMBLE members to create a use-case that is understandable and valuable to other members.

6.2.1. Current Baseline of Tools and Technologies

We will use MPS and DclareForMPS to build the use case. We are currently enriching and improving DclareForMPS to support the combination of blending, immediate-transformation, and collaboration.

We will also use at least one (perhaps only partially) language for defining state-machines that are developed by another BUMBLE partner. We are now considering the language OIL from Canon for this purpose.

6.3. BUMBLE Features

This use case will combine blending and collaboration. Blending by immediately transforming two languages with different abstract syntaxes, and collaboration by synchronizing the two models across the network, each model living in its own MPS run on different machines.

6.4. Demonstrator

We will demonstrate two remote client environments, each showing the same state-machine. One in a tabular representation and one in a textual representation. Changes in one client environment will be (immediately) transformed to the other client environment, and vice-versa. The two languages will have different concrete and different abstract syntaxes, where the two abstract syntaxes do not have a trivial mapping. Hence, it will involve a non-trivial bi-directional transformation.

Apart from demonstrating immediate synchronization, we will also demonstrate deferred synchronization: The two clients will be disconnected, the models changed, and connected again to show that after reconnecting the two clients the models will be synchronized again.

We will demonstrate this using MPS and DclareForMPS. If requested we will later demonstrate the same functionality together with an extra synchronized client using Eclipse, EMF using (yet to be developed) DclareForEMF.

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

7.1. Background

At SAAB, developing large distributed systems consisting of multiple complex logical functions distributed over several hardware units is a complex task that requires a lot of systematic work to ensure that all parts of the system work together. Systems are modelled with SysML to ensure that all functions are consistent and coherent from a functional perspective. All functions are divided into smaller manageable blocks, called system components. Each system component realizes a part of a function and is realized either in software as a software component or is realized in hardware. Each software component is allocated to one CPU in a hardware unit. Software components are always allocated to a CPU as a whole component. In the SysML model, each system component is augmented with a state machine in order to reason about the total state of a system function.

It is fair to say that the SysML model describes the functional architecture by describing the functional partitioning of the system into system components and describing their required interactions to fulfil the system functions.

Modelling is only used on the system level, whereas software components are selected from either existing ones or implemented from scratch. In any case, the system model is not used for generating code.

In addition to the SysML model, the detailed behaviour of a system component may be specified in a range of different artefacts, such as Matlab models, structured or unstructured data, textual descriptions, or pseudo code. The SysML model is the most important model from an architectural perspective and is the leading artefact describing the intended architecture of software components.

The BUMBLE framework is expected to facilitate:

- The consistency checking between the intended architecture expressed in the system architectural model (SysML) and the implemented architecture in C/C++ software components and their interaction.
- Blending notations of architecture and implementation to enable bidirectional navigation of portions of the SysML model and corresponding portions of the C/C++ code.
- Traceability and visualization of unstructured data artefacts additional to code and model, used for describing the internals of system components.

7.2. Use Case Description

To exemplify the use case, several user stories are described.

7.2.1. User Stories

Implement a Software Component from the System Model

- As a software developer, I want to be able to edit code and models in a blended fashion, that is to say concurrently and while keeping consistency between them intact seamlessly.
- As a software developer, I want to be able to seamlessly link unstructured data (such as data sheets, notes, etc.) to code and visualize them together in a blended fashion.

- As a software developer, I want to be able to graphically view all the traced artefacts (system model, documents, etc.) that should be implemented in the software.
- As a software developer I want to use a single IDE for implementing and viewing.

Validate Implementation

- As a developer I want to be able to automatically validate architectural violations of my implementation compared to the systems model.
- As a developer I want to be notified of architectural violations in the graphical views.

Feedback Changes to the System Model

- As a developer I want to suggest or feedback changes to the system model required to implement the system.

7.3. BUMBLE Features

7.3.1. Blended Syntaxes and Modeling

Models in SysML, related code and unstructured data shall be visualized in a structured and blended manner. Models and code should be editable in a blended fashion keeping consistency intact seamlessly.

7.3.2. Evolution

The system model and implementation will evolve throughout the development and (long-term) maintenance of the system. The traceability and consistency checks should as much as possible be automatically updated upon a change to the system model or the implementation. We will distinguish between automatically established traceability links and manually added traceability links, since they need to be treated differently during the system's evolution.

7.3.3. Traceability

We initially aim to facilitate traceability between the system model and corresponding implementation. Additionally, we want to also link unstructured data to the implementation and provide traceability and a visualization thereof to the developers.

7.4. Demonstrator

We will focus on two technologies to create a demonstrator for this project. In particular, we will create a bridge between the modeling tool Rational Rhapsody (for SysML modeling) and the IDE CLion (for implementation of software components and system code). The bridge shall allow navigation between code fragments related to a selected model element and vice versa. Moreover, we will show how consistency between the different artefacts is modeled, established, and kept intact.

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

8.1. Background

At Sioux, we use an in-house developed modeling workbench, named Supermodels. Supermodels workbench provides facilities to create and use graphical DSMLs (with diagrammatic notations) in high fidelity editors.

In Supermodels, we have built interconnected DSMLs that focus on specifying different aspects of a system and its control software like: (1) the structure [decomposition], (2) behavior [state machines] and (3) constraints [SBVR]. Such interconnected specifications are captured in a DSMLs instance (model) which is used for generating (part) of control software, documentation, review purposes and/or simulators (of the controlled system).

8.2. Use Case Description

8.2.1. Description and Rationale

We intend to blend graphical DSMLs of Supermodels with multi-notation DSMLs of MPS.

Strong point of Supermodels is that it allows us to create WPF based diagram editors for our DSMLs which are flexible and customizable. But graphical editors are not so convenient for DSMLs that are better represented as text, such as SBVR. SBVR is a formal language that describes requirements of a system, but it’s readable in a natural way. We want to use a subset of SBVR to specify interlocks (constraints) on system behavior. JetBrains MPS projectional text editor would be a great way to edit SBVR, while Supermodels can be used for the graphical DSMLs (like state machines, decomposition).

A weaker point of Supermodels is its meta modeling environment. It is implemented directly in C# with almost no special tooling for editing meta models. Also, the MetaMetaModel does not define itself, but is defined by C#. On the other hand, MPS has a strong and mature meta modeling environment but lacks high fidelity graphical editors.

MPS has a proven merge functionality. That works with multiple versions of the MetaModel and has great integration with the textual editors. On the other hand, Supermodels has a merge tool that has several limitations in certain edge cases when merging. There is no diff tool with proper visualization of differences. Also, evolution of the language poses a set of problems in the merging.

To get the best of Supermodels and MPS we imagine an approach where MPS is used as back-end and Supermodels as front-end (view). Thus, Supermodels benefits from proven language services provided by MPS while keeping its simpler user experience (as opposed to programming IDE like experience) and nice graphical editors.

8.2.2. Current Status and Existing Functionalities

Supermodels provides a plain text editor to edit SBVR rules. It can check and indicate syntax errors based on ANTLR4 grammar. It doesn’t provide autocompletion in referencing elements from the other DSMLs (like components or states).

In Supermodels, we store a DSMLs instance (model) as a file. File format and model meta-meta formats are not compatible with JetBrains MPS. The provided diagram editor is fully implemented for the used graphical DSMLs (state machines, decomposition).

There are some 3rd party solutions to interoperate with JetBrains MPS like MPSServer, Modelix, Java – dotNet bridges that are worth exploring at least as a starting point.

Supermodels provides a basic merge tool that can visualize merge conflicts but no diffs. That merge tool resolves most merge conflicts problems, except for some edge cases.

8.2.3. Desired Functionalities to be Provided by BUMBLE

SBVR editor in MPS that could reference items from Supermodels DSMLs (state machines and decomposition), has autocompletion and has good intentions for refactoring.

Live synchronization between the Supermodels views (created in .NET/C#) and JetBrains MPS views (JVM). Possibility of using JetBrains MPS language services (model checks, model storage, generators, intentions, DCLARE, etc.) and showing, if necessary, the results in Supermodels views.

Exposed MPS API for diff/merge functionality (server) coupled to .NET based diff/merge client (prototype) with WPF to visualize source/target, differences and conflicts in diagram editor.

8.2.4. User Stories

Modeling and Model Management

- As a DSML user, I want to open (or create new) and save (persist) a model from both Supermodels workbench and MPS.
- As a DSML user, I want to use some DSMLs (mostly diagrammatic like State machines, Decomposition) in Supermodels workbench to edit (parts of) a model.
- As a DSML user, I want to use some DSMLs (like SBVR) in MPS to edit (parts of) a model.
- As a DSML user, I want to be able to observe live (with negligible latency) the changes made in one view/environment (e.g. MPS) from the other (e.g. Supermodels) and vice versa [assumes both are used on the same host computer].
- As a DSML user, I want to trigger model checks from both Supermodels and MPS (engages model checkers in both Supermodels and MPS).
- As a DSML user, I want to trigger generation from both Supermodels and MPS (engages generators in both Supermodels and MPS).
- As a DSML user, I want to be able to work with big models of 50+K elements while keeping the UI responsive enough.

Blended Modeling

- As a DSML user, I want to specify some, potentially interconnected, aspect(s) of system control software by using DSMLs (views) appropriate to the aspect at hand.
- As a DSML user, I want to use such interconnected specifications for generation of control software, documentation, and/or simulators.

Model Collaboration

- As a DSML user, I want to use version control (like git, svn) to collaborate with my fellow modellers.

- As a DSML user, I want support for diff and merge on DSML level from MPS (and optionally from Supermodels).
- As a DSML user, I expect to use both Supermodels and MPS on the same host computer only by myself (thus no multi-user collaboration on the same machine).

Language Development

- As a DSML developer, I want to be able to continue developing the existing Supermodels DSMLs (structure, diagram editor, generator, model checker).
- As a DSML developer, I want to specify (new) DSMLs (structure, editor, generator, behaviour, type system etc.) in MPS and have the possibility to implement diagrammatic editors in Supermodels.

8.2.5. Desired Non-Functional Requirements

- Usability: user experience in editing the SBVR rules should be close enough to a plain text editor (hiding the projectional nature of the editor).
- Usability: synchronization between MPS and Supermodels views should happen fast enough. to be perceived by the user as live updates (probably less than 0.5s).
- Scalability: handle models of 50+K elements.
- Usability: visualize differences/conflicts of elements and their properties in a concise, readable, and clear way.

8.2.6. Current Baseline of Tools and Technologies

Within Sioux, Supermodels is actively developed and used for predominantly graphical DSMLs. Also, JetBrains MPS is used for development of mostly textual DSMLs. The multi- and cross-disciplinary modelling workbench provided by BUMBLE should allow Sioux to benefit from the best of both Supermodels and MPS.

Sioux is interested in DCLARE to explore its ability to synchronize models between multiple DCLARE instances. The synchronization interface provided by DCLARE can be potentially useful to synchronize between MPS and Supermodels.

Sioux is interested in exploring MPSServer as it exposes an interface of MPS to query and modify the model via HTTP and/or WebSockets.

Sioux is interested in exploring Modelix as it gives a way to directly include MPS editors into other environments like Supermodels.

8.3. BUMBLE Features

8.3.1. Blended Syntaxes and Modeling

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. Thus, facilitating multi- and cross- disciplinary modeling. Live synchronization between Supermodels and MPS views on the multi aspect system specification.

8.3.2. Collaborative Modeling

Support version control (git, svn) collaboration model between multiple DSML users working on the same model.

[Feasibility] Approach to visualize differences between models using graphical DSMLs and resolve conflicting changes on DSML level.

8.4. Demonstrator

A demonstration at final review would show a main scenario where the BUMBLE capabilities of blending different but interconnected aspects of a system specification. The demonstrator will be based on the MPS-based DSMLs described above.

- Editing SBVR in MPS with rich features such as auto-completion, cross-referencing to model elements created in Supermodels.
- Near-immediate synchronisation between Supermodels and MPS views.

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

9.1. Background

HERMES İletisim's main aim is to design and implement a model-driven engineering platform to ensure Business Process Management for Debt-Collector Advocates (DCAs), shortly called BPM4DCA. DCAs try to reach their customers/debtors via many different ways as shown in the following sample workflow.

In the above sample workflow, a DCA collects debts in the following steps:

1. Reach the debtor using various communication channels such as Phone Call, SMS, Voice Message or National ID SMS.
2. If the debtor couldn't be reached, his/her guarantor, mother, father, or other relatives in many different ways will be reached using various communication channels such as Phone Call, SMS, Voice Message or National ID SMS.

Moreover, to reach debtors in these complicated and iterative ways, these debt collectors should deal with more than 10K case files on average which must be handled only in one month.

Managing this complex process leads to various errors and difficulties as follows:

- Difficulties to trace the workflow in verifying the accomplishment of a task.
- Errors resulting from performing unnecessary and extra tasks despite the completion of a task that leads to complaints and dissatisfaction from the users.
- Being error-prone when modifying an existing workflow.
- Complexity of designing a new workflow.

9.2. Use Case Description

9.2.1. Current Status and Existing Functionalities

Currently, a DCA reaches a debtor from different communication channels manually. As illustrated in Figure 1. Call center employees take each row of an Excel file as a new case and tries to accomplish the task until achieving the desired result. These processes are done consecutively (sequentially) for more than 10K cases in one month. In this current design, there is not any model driven / blended modelling which is seen as a need in our use case as managing complex workflows since constructing specific workflow for each case is so hard, error prone and time consuming.

9.2.2. User Stories

In the following, details of the user stories are given to characterize expected functionalities of the BPM4DCA use case.

- As a system designer, I want to login to a platform to get access to a graphical and textual modelling environment.
- As a system designer, I want to design my desired workflow by drag-and-dropping elements in an editing environment to alleviate the modifying process of the model.
- As a system designer, I want to be able to reach my previous models on a platform to modify them easily.

- As a system designer, I want to view and draw graphically my workflow's rules represented in the JsonLogic format.
- As a system designer, I want to give different priorities to users to give access to the existing models.
- As a system designer, I want to perform live tests of my workflow to validate and evaluate the applicability and correctness of it.
- As a system designer, I want to be informed about the notifications and errors in the modelling environment to satisfy problems that have occurred.
- As a system designer, I want to easily edit my defined rules to represent them in a graphical view simultaneously.
- As a system designer, I want to be able to get the output of the model to use it programmatically.
- As a system designer, I want to generate the model in XML format to store in a database to ease the access of them when the models are needed.
- As a system designer, I want to create new tasks by using the attributes of the current task and visualize relations of their workflows in a single diagram so that I will have better workflow management.
- As a system user, I want to be able to modify the workflow in a textual and graphical editing environment without writing any code or low code.

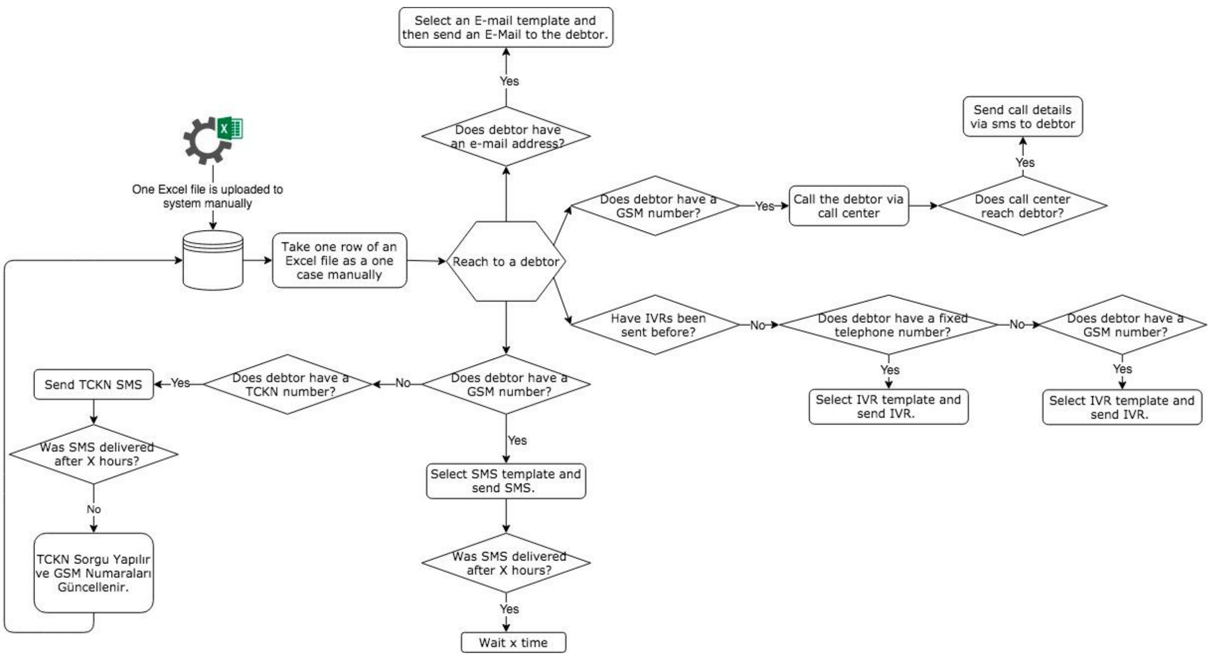

Select an E-mail template and then send an E-Mail to the debtor.
Yes
Does debtor have an e-mail address?
Send call details via sms to debtor
Yes
One Excel file is uploaded to system manually
Does debtor have a GSM number?
Yes
Call the debtor via call center
Does call center reach debtor?
Take one row of an Excel file as a one case manually
Reach to a debtor
Have IVRs been sent before?
No
Does debtor have a fixed telephone number?
No
Does debtor have a GSM number?
Yes
Yes
Select IVR template and send IVR.
Select IVR template and send IVR.
Send TCKN SMS
Yes
Does debtor have a TCKN number?
No
Does debtor have a GSM number?
Yes
Select SMS template and send SMS.
Was SMS delivered after X hours?
No
TCKN Sorgu Yapılır ve GSM Numaraları Güncellenir.
Was SMS delivered after X hours?
Yes
Wait x time

Figure 1. Current dept collecting workflow.

9.2.3. Current Baseline of Tools and technologies

Advocators trace debtors until they collect the whole debt manually. Information of the debtors are stored in an Excel file where each row in it is related to a debtor. Advocates take a row of an Excel file and try to reach the debtors or their relatives and guarantors via different communication channels such as Telephone, SMS, and email. Advocates perform this process for all debtors which produces more than 10K cases in one month. Modeling the management of this complex workflow in a blended (textual and graphical) modeling environment eases the trace of the debt collecting process.

9.3. BUMBLE Features

9.3.1. Blended Syntaxes and Modeling

It is expected that models can be generated, viewed, and modified in a blended environment to satisfy graphical and textual representation.

9.3.2. Collaborative Modeling

Collaborative modelling can be satisfied by allowing to run multiple workflows in parallel so that it can trigger different flows of a process for related tasks. We expect the tool to be able to support task forking to use in the same workflow or different workflows.

9.3.3. Traceability

It is expected to trace dependencies and relations among model (workflow) artifacts to measure the model correctness and performance.

9.3.4. Model Non-Conformity

Non-conformance report is expected to keep track of deviations and accepted standards to warn and notify the user about the failures of standards and particular specification.

9.4. Demonstrator

We expect a demonstration brought by BUMBLE project capabilities to facilitate modeling and implementation of both choreography and orchestration of complex business services inside the BPM4DCA use case. A demonstration would show a blended modeling environment where visual business process models seamlessly integrate with the use case components which are defined in some sort of textual rule formalizations. We will demonstrate all MDE tools enabling users (e.g. advocates with no programming skills) to create their communication way and debt collecting process workflows visually with drag and drop facilities which paves the way for the low coding and auto-generation of the required software.

10. UC9 - Automated Design Rule Verification on Vehicle Models

10.1. Background

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 synchronization of design rules with the geometry and product manufacturing information.

We will develop a textual domain specific language to formalize 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 will check the validity of the design rules against manufacturing and geometric data using ISO’s JT Standard (ISO 14306:2017) using a synchronization engine to be developed on top of traceability facilities of the BUMBLE project.

10.2. Use Case Description

The following 3D design view demonstrates the engine cover submodule of a vehicle’s engine, see Figure 2. Each manufacturing part in this module has some touch conditions with the other inner parts or other subsystems of the engine such as engine and transmission wiring as shown on the CAD design. However, the environment will be capable of analysing the whole vehicle.

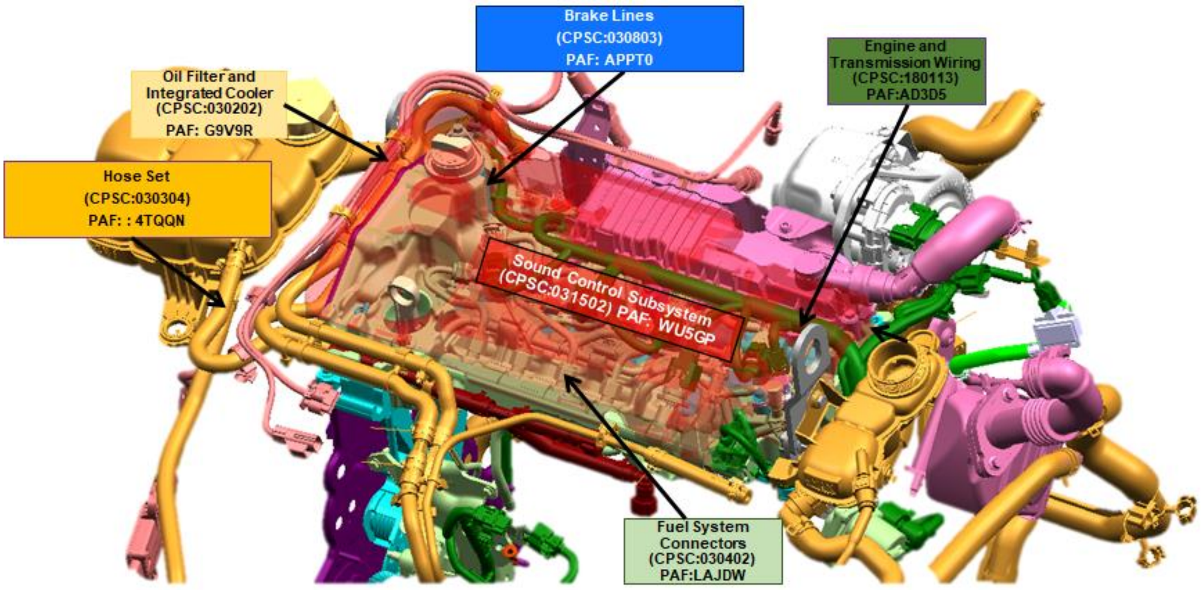

Brake Lines (CPSC:030803) PAF: APPT0
Engine and Transmission Wiring (CPSC:180113) PAF:AD3D5
Oil Filter and Integrated Cooler (CPSC:030202) PAF: G9V9R
Hose Set (CPSC:030304) PAF: : 4TQQN
Sound Control Subsystem (CPSC:031502) PAF: WU5GP
Fuel System Connectors (CPSC:030402) PAF:LAJDW

Figure 2. 3D-Design Overview of an Engine.

We have prepared the following mock-up to visualize the graphical representation of the touch conditions based on Ford-Otosan’s internal manufacturing part hierarchy, see Figure 3.

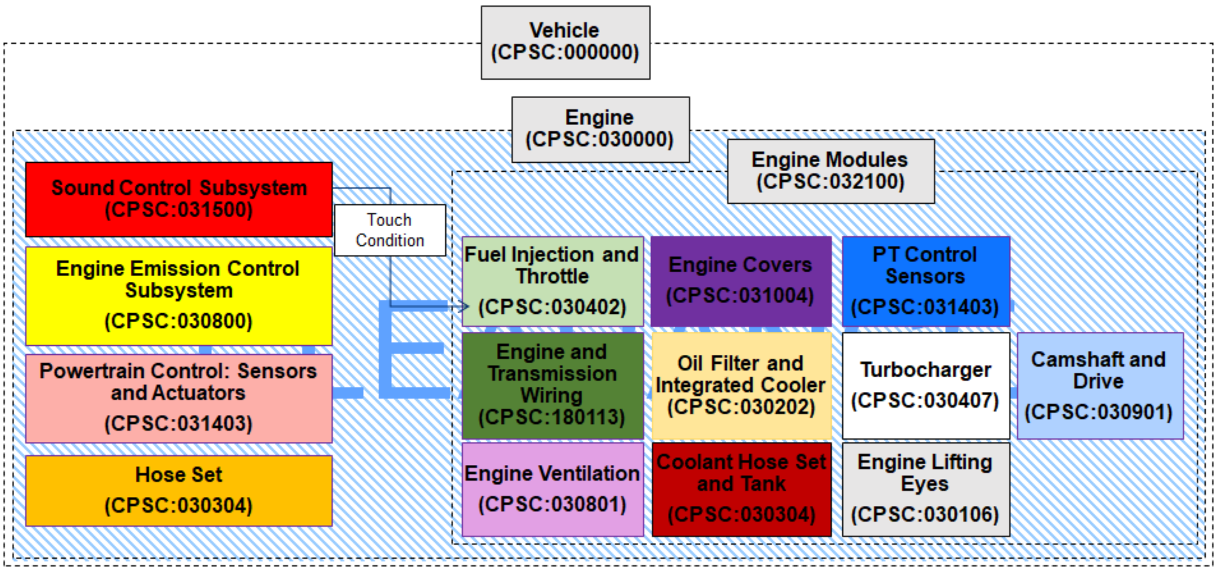
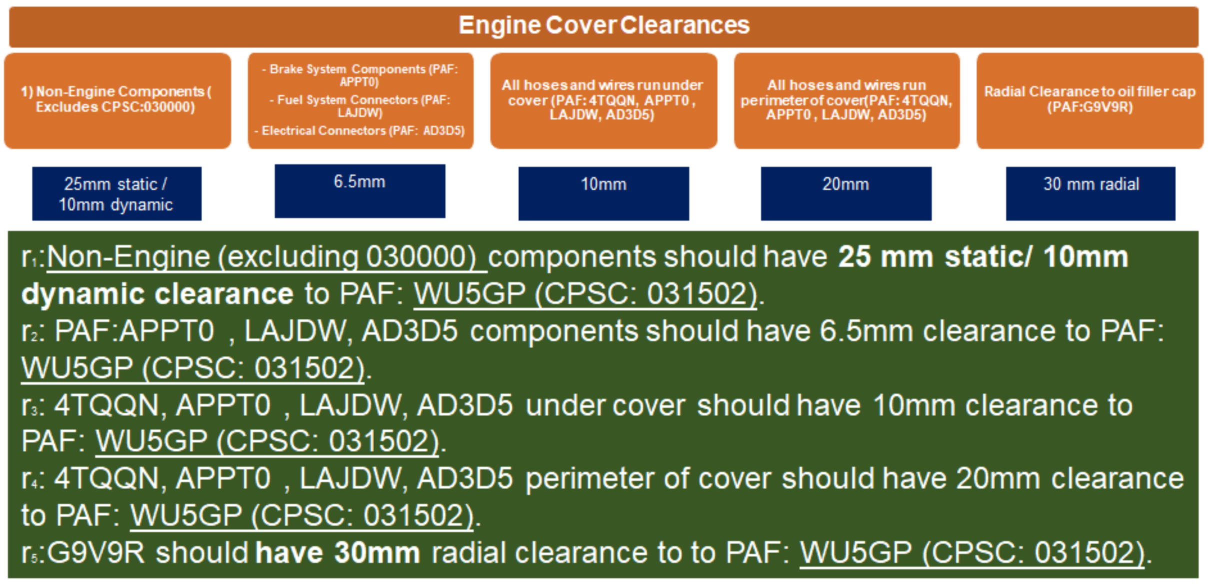

Vehicle (CPSC:000000)
Engine (CPSC:030000)
Engine Modules (CPSC:032100)
Sound Control Subsystem (CPSC:031500)
Touch Condition
Engine Emission Control Subsystem (CPSC:030800)
Powertrain Control: Sensors and Actuators (CPSC:031403)
Hose Set (CPSC:030304)
Fuel Injection and Throttle (CPSC:030402)
Engine Covers (CPSC:031004)
PT Control Sensors (CPSC:031403)
Engine and Transmission Wiring (CPSC:180113)
Oil Filter and Integrated Cooler (CPSC:030202)
Turbocharger (CPSC:030407)
Camshaft and Drive (CPSC:030901)
Engine Ventilation (CPSC:030801)
Coolant Hose Set and Tank (CPSC:030304)
Engine Lifting Eyes (CPSC:030106)

Figure 3. Mock-up Visualization of Touch Conditions.

Each touch condition comprises one or more design rules (clearance specifications), which should be provided with a syntax-directed textual language as sketched in the following (rules from r_1 to r_5) by the design-engineering experts. Example clearance specifications are shown in Figure 4.


Engine Cover Clearances
1) Non-Engine Components (Excludes CPSC:030000)
- Brake System Components (PAF: APPT0)
- Fuel System Connectors (PAF: LAJDW)
- Electrical Connectors (PAF: AD3D5)
All hoses and wires run under cover (PAF: 4TQQN, APPT0 , LAJDW, AD3D5)
All hoses and wires run perimeter of cover(PAF: 4TQQN, APPT0 , LAJDW, AD3D5)
Radial Clearance to oil filler cap (PAF:G9V9R)
25mm static / 10mm dynamic
6.5mm
10mm
20mm
30 mm radial

r_1:Non-Engine (excluding 030000) components should have **25 mm static/ 10mm dynamic clearance** to PAF: WU5GP (CPSC: 031502).
r_2: PAF:APPT0 , LAJDW, AD3D5 components should have 6.5mm clearance to PAF: WU5GP (CPSC: 031502).
r_3: 4TQQN, APPT0 , LAJDW, AD3D5 under cover should have 10mm clearance to PAF: WU5GP (CPSC: 031502).
r_4: 4TQQN, APPT0 , LAJDW, AD3D5 perimeter of cover should have 20mm clearance to PAF: WU5GP (CPSC: 031502).
r_5:G9V9R should **have 30mm** radial clearance to to PAF: WU5GP (CPSC: 031502).

Figure 4. Example Clearance Specification.

Our aim is to provide a tool for design teams at Ford-Otosan to support the design-rule verification process by checking the design under development conforms to clearance design rules. The modelling environment will blend the graphical projection of touch conditions from 3D design and manufacturing data with the textual design rules.

10.3. BUMBLE Features

10.3.1. Blended Syntaxes and Modelling

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.

10.3.2. Collaborative Modelling

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.

10.3.3. Traceability

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 synchronized throughout the design process. If there is an inconsistency detected among synchronization points, it should be reported to the development team pinpointing the source of the inconsistency.

10.3.4. Model Non-Conformance

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 traceability infrastructure of the BUMBLE environment.

10.4. Demonstrator

We aim to demonstrate the following features on a Ford-Otosan's real vehicle model: blended language, collaborative modelling, traceability and synchronization, consistency checking.

11. UC10 - Development Process of Low-Level Software

11.1. Background

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 engineers. Like in all tech companies, our projects flow along the chain:

Case → Idea → Implementation, testing and documentation → Review → Maintenance

where each step typically involves different people, skills, and tools. This diversity introduces many error sources, such as

- miscommunication between customers, sales staff, project managers, engineers, etc.,
- difficulties in finding suitable reviewers, with the right skills and enough time, within the engineering team,
- complicated documentation, as it needs to be readable to all involved parties, and
- a high learning threshold for other engineers when the code needs adaptation to new conditions.

The possibility to automatically switch between representations of a model would, of course, simplify communication between the involved parties, as well as eliminate the risk of introducing errors during manual translation between representations. It would also make the process more efficient, both in terms of working hours, and distribution of resources:

- Automatically generated code would decrease implementation time, and, possibly, reduce the risk of errors.
- Less time would need to be spent on documentation, as the possibility to visualize (part of) the implementation in different ways would make it partly self-explanatory.
- The threshold would be lower for engineers with different skill sets to get to know, interact with, maintain, and adapt implemented functionality, as each could work with the representation that suits them best.

In short, BUMBLE technology would support companies like Unibap in efficient utilization of valuable resources.

At present, Unibap has no company standard for modelling languages, but the choice varies among both projects and engineers, depending on skills and requirements. Please see the following sections for a list of languages and tools that are used in the particular project on which this use case is based. We use graphical models and modelling languages to

- model safety critical software parts to ensure their robustness, and
- convey design information between engineers that have knowledge of different parts of the platform.

Textual modelling languages are used to, for example, set up complex systems with internal modules and communication, as some frameworks used in the company do not support automatic code generation from graphical representations.

11.2. Use Case Description

11.2.1. Current Status and Existing Functionalities

Currently available tools support automatic generation of code (for example, C, C++, Java) from graphical representations of state machines. In our perspective, there are, however, two major problems:

- *Dependencies on internal libraries*
  We have found no tools that allow dependencies on internal libraries. This prevents us from using graphical methods to implement more complex parts of the software, where the approach would otherwise be very desirable.
- *Version control*
  We have also found no tools with sensible diff/merge functionalities or proper version control.

The design tool that we currently use does not support blended modelling, which is a big loss as it makes communicating design decisions harder and thus increases the time it takes to develop code.

11.2.2. User Stories

- As a system designer, I want to automatically translate my (Yakindu) models to UML and XML, to increase readability for other people in the project and company.
- As a system designer, I want to be able to analyze coverage based on a graphical model, to ensure the sanity of my design.
- As a developer, I want to automatically generate C code from a graphical representation of state machines, to make development more efficient.
- As a developer, I need the code of my models to have dependencies on internal and external libraries, to be able to efficiently develop custom software.
- As a developer, I want to do proper version control and diff/merge operations on my models, to simplify cooperation on a model, and thus increase development speed.
- As a developer, I want to generate different graphical representations of implementations, where applicable, for use in documentation.
- As a reviewer, I want to be able to convert a representation of a model to other modeling languages, textual and graphical, to help me review an implementation without being familiar with the tool used to create it.
- As a developer, I want to include code snippets with full standard C support in the textual syntax, to be able to customize state machines.
- As a developer, I want to be able to use nested includes, to make development efficient, and to keep my code concise and readable. (Nice to have).

11.2.3. Current baseline of tools and technologies

Purpose and Role of the Currently used Relevant Tools and Technologies

- Yakindu Statechart Tools
  Yakindu statechart tools is used to model, simulate, and test state machines. The created models can then be transformed into source code in C (among other languages) that then can be compiled and used. Unibap uses Yakindu to model and test safety critical parts of our software platform. Modelling the state machines graphicly reduces the risk of introducing bugs during development, and also makes it easier to evaluate the robustness of the system when testing.
- Atlassian Confluence and DrawIO

Model designs are represented using XML, created in DrawIO in Confluence, to explain the workings of the specialized parts of our platform to representatives of all parts of the company, and to convey critical design information to contributing engineers.

Tools and Technology that BUMBLE Solutions need to Interface With

- Yakindu Statechart Tools
  We have very positive experience from using this particular tool, which is also an Eclipse plugin. A similar tool would do, but this one is high on our wish list.
- DrawIO / XML / …
  We would have great use of a tool for drawing a representation of a model from which code can be generated automatically. The choice of tool is not important, but we need the functionality. The tool should be collaborative (diff/merge operations should be possible), but it does not need to be web based.

11.3.BUMBLE Features

11.3.1. Blended Syntaxes and Modeling

- We expect that models can be created, viewed, and edited both as graphical and textual representations.
- We expect that models can be viewed and edited as C code. It is, however, not required that a model can be created from scratch in C. We need the possibility of adding snippets of full C17 to the C code representation of models.

11.3.2. Collaborative Modeling

Multiple developers are expected to contribute to a model. We expect to be able to use git-like diff/merge operations, as well as version control, to enable them to work efficiently, and to keep track of changes.

11.3.3. Traceability

We expect to be able to analyse coverage in the graphical, textual and code representations of the models.

11.3.4. Model Non-Conformance

Some non-conformance must be expected in conversions between graphical and from text-based representations. We expect all information, such as for example comments in code, to be preserved behind the scenes. A warning message, notifying the user that there is information that cannot be conveyed by the chosen modelling language, would be of great help.

11.4.Demonstrator

We expect to provide a report in which we describe the experiences we gain from our case study, for example:

- communication difficulties stemming from different model representations being used simultaneously,
- difficulties in resource allocation due to skill distribution within the company,
- time spent on design, implementation, and documentation, and where we think time could have been saved, and

- how product quality is affected by choices that must be made because of tool limitations.

If a BUMBLE prototype, fulfilling some or all of our listed user stories, is available in time, we hope to provide corresponding observations for a similar project in which the prototype is used. If only some of the user stories are fulfilled, the evaluation of the workflow in the second project will focus on them.

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

12.1. Background

AVL is the world's largest private company providing automotive OEMs with test equipment and development knowledge. Virtualization and the use of models has a long tradition at AVL. However, there is a strong focus on the application of models in vehicle development (e.g. application to simulation and physical models) and less focus on the use of models for test equipment development. In the case of test equipment, we are usually talking about a very complex system of systems that need to interact with each other. Developing the individual systems and assembling them into a (often customized) system of systems adds a certain complexity to the development process involving different stakeholders and departments. A reduced focus on models does not mean that there is no focus. However, the extent to which models are used by different stakeholders varies and thus cross-departmental collaboration remains complex. A main goal of our participation in BUMBLE (and the national part HybriDLUX) is to bring some of the involved departments to the same level of model usage in order to increase collaboration efficiency and indirectly product quality. We believe that DSLs strongly oriented towards end-user requirements, which include UX aspects as a first-class requirement, have great potential to achieve this goal.

12.2. Use Case Description

12.2.1. Description and Rationale

At BUMBLE (and HybriDLUX) we want to extend existing and new DSLs with the ideas of blended and collaborative modelling. With regard to 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. In order to somewhat concretise the DSLs applied in this context, the following three DSLs will be considered here:

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

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

12.2.2. Current Status and Existing Functionality

The DSLs mentioned have different levels of maturity in terms of functionality and departmental integration.

- DSL A: There is a quite mature prototype based on Eclipse Theia, but the departmental integration is still limited.
- DSL B: There is a very mature and industrialised DSL that is textually based. It is currently being extended by a graphical user interface, which also has a fairly high degree of maturity and is currently being industrialised and evaluated for UX aspects. The link to DSL A is made via a DB between the two DSLs (DSL A writes to the DB, while DSL B has read access here).
- DSL C: A mature prototype exists that is currently being integrated into several departments. A UX evaluation is planned shortly. In addition, an integration of the DSL into a third-party (Eclipse-based) graphical DSL tool is being discussed. The reference to DSL A will be made in the same way as for DSL B.

The functionality of DSL B was presented at the first review meeting of BUMBLE. The focus of the functionality of DSL A is on a so-called round-trip integration with the underlying DB including intra-departmental collaboration functions (Model Diff/Merge on textual and graphical level). The focus of DSL C's functionality is currently on use case-oriented textual language design and interaction with 3D CAD representations.

12.2.3. User Stories

DSL A (Device Modelling for Device Knowledge Base)

- Blended Modelling
  - As a modeller, I want to create, modify, and manipulate state machine models using both at textual (JSON-based) and at graphical level.
  - As a modeller, I expect that any change on a specific textual model view causes an immediate update on the graphical view and vice versa.
- Collaborative Modelling
  - Inter-Department (same DSL)
    - As a modeller, I want to commit any change on my model to a shared DB (write access only within the same department).
    - As a modeller, I want to see any merge conflicts of my modified model to the version stored in the DB.
    - As a modeller, I want to see conflicts in both views, i.e., in the graphical and textual representation.
    - As a modeller, I want to resolve any conflicts on both views, graphical and textual representation.
  - Intra-Department (different DSLs)
    - None (DSL B and C are consumers of DSL A).
- Model traceability / debugging
  - Not in the focus for this DSL.
- Model evolution
  - Not in the focus for this DSL.

DSL B (Device Integration Testing)

- Blended Modelling
  - As a modeller, I expect that the textual representation of my DSL is the driver, but a graphical representation should show me certain aspects not easy to discover on the textual representation.
  - As a modeller, I expect that the graphical view is updated after a change in the textual DSL accordingly.

    - As a modeller, I expect that I can examine in the graphical representation, how a change of the textual model affects certain aspects shown in the graphical representation (e.g. before/after illustration).
- Collaborative Modelling
    - Inter-Department (same DSL)
        - As a modeller, I would like to re-use similar existing models, if I start a new modelling project based on certain criteria (model zoo), like device classes, which are based on similar properties and thus on similar test cases.
    - Intra-Department (different DSLs)
        - As a modeller, I expect to get information of the device interface defined by DSL A.
        - As a modeller, I expect that this information is represented by corresponding suggestions in my editor (e.g. code completion).
        - As a modeller, I would like to see immediately, if I am not conforming to the device interface (e.g. red underlines).
        - As a modeller, I would expect a notification mechanism, if something changed on DSL A, where I am depending on.
        - As a modeller, I would expect a semi-automatic support in reacting to any changes on DSL A, where I am depending on.
- Model traceability / debugging
    - As a testing engineer, I would like to generate test cases out of the models.
    - As a testing engineer, I would like to trace, which generated test case belongs to which artefacts.
    - As a testing engineer, I would like to understand in case of failed test case executions, which model artefacts are related to those failed ones.
- Model evolution
    - As a modeller, I would like to migrate from one language version to another with the least possible effort.
    - As a modeller, I would expect an automatic migration for most cases.
    - As a modeller, I would expect a navigation to all model parts, where automatic migration is not possible.
    - As a modeller, I would expect suggestions or proposed alternatives to finalizing the migration.
    - As a testing engineer, I would like to evaluate the migration with the least possible effort (e.g. all tests the went through in the old version, go through in then new version - or if not, I am pointed to those cases in the model -> relation to model traceability/debugging).

DSL C (Step-by-Step Guidance)

- Blended Modelling
    - As modeller, the textual language representation is the main driver, however interaction with other model representation may change the textual representation in certain cases.
    - As a modeller, I expect a preview of my definitions in an interactive 3D player (e.g. steps and animations defined in the textual DSL).
    - As a modeller, I expect to use this 3D player as well to link to certain aspects of the underlying CAD model (e.g. (spare) parts), predefined animations.
    - As a modeller, I also want to have a 2D representation of the decision tree of the sequence of task steps.
    - As a modeller, I also want to use the 2D view to manipulate the decision tree.

    - As a modeller, I want that the textual representation is updated automatically, if the 2D representation is manipulated.
- Collaborative Modelling
    - Inter-Department (same DSL)
        - None at the moment of UC definition (however same mechanism as for DSL A could be useful on a mid-term perspective).
    - Intra-Department (different DSLs)
        - As a modeller I want to refer to artefacts, which are external to DSL C
            - This external source is DSL A
                - especially measurement channels, states, and errors.
            - The underlying CAD models
                - especially (spare) parts and animations.
        - As a modeller, I would expect a notification mechanism, if something changed the mentioned external sources, where I am depending on.
        - As a modeller, I would expect a semi-automatic support in reacting to any changes on external sources, where I am depending on.
- Model traceability / debugging
    - As a modeller, I would like to execute the step-by-step guidance and see the effect on the 3D player and on the 2D view (decision tree).
    - As a modeller, I would like to apply breakpoints if applicable.
    - As a modeller, I would like to be pointed to the respective task step definition, if running the 3D step-by-step guide interactively.
- Model evolution
    - As a modeller, I would like to migrate from one language version to another with the least possible effort.
    - As a modeller, I would expect an automatic migration for most cases.
    - As a modeller, I would expect a navigation to all model parts, where automatic migration is not possible.
    - As a modeller, I would expect suggestions or proposed alternatives to finalizing the migration.

12.2.4. Desired Functionality to be Provided by BUMBLE

- Blended textual/graphical (incl. 3D) DSL modelling support. In particular, improvements to maintenance requirements regarding blended modelling in case of language evolution.
- Improvements regarding intra-departmental model collaboration, in particular regarding graphical model diff/merge support.
- Improvements regarding interdepartmental model collaboration for independent but related DSLs, with particular focus on model change management and notification.
- Model traceability and debugging support for the generation of artefacts from DSL models (e.g. generation of test cases and feedback of test case execution results into the related DSL model).

12.2.5. Desired Non-Functional Requirements

- UX aspects must be treated as first-class aspects for all the above functionalities (focus in HybriDLUX).
- Long-term support of the underlying frameworks must be guaranteed to a certain extent (e.g. by relying on open source frameworks with a very active community avoiding the direct or indirect use of outdated frameworks).

12.2.6. Current Baseline of Tools and Technologies

All DSLs mentioned are more or less based on technologies/frameworks such as Xtext, Xtend, JSON, EMF, LSP, GLSP, Eclipse IDE, Sprotty and Theia. Special technologies are various DBs for data storage and (web-based) 3D players based on Unity. The different DSLs differ in terms of focus, which technology is used for UX, historical, socio-cultural, organizational and use case reasons. This makes the combination of DSLs partly challenging, partly exciting. End-user acceptance factors sometimes trump technological arguments for straightforward implementation. This underlines the importance of flexible combinations of different techniques, such as those supported by LSP, to separate the DSL definition from its modelling authoring tool.

12.3. BUMBLE Features

12.3.1. Blended Syntaxes and Modeling

Required by DSL A, B and C (including 3D CAD support).

12.3.2. Collaborative Modeling

- Required for intra-department collaboration (i.e. working on the same model) required for DSL A (including model merge/diff on textual and graphical level).
- Required for cross-department collaboration (linking different kinds of DSLs) in a loose coupled manner (links). Change notification mechanism required.

12.3.3. Evolution

Required especially for DSL B and even more for DSL C for two aspects
- Ensuring the models from previous DSL versions can be re-used in later version (for both textual and graphical representation).
- Ensuring that the maintenance effort for blending modelling is kept low in case of language evolution.

12.3.4. Traceability

- Vaguely yes for DSL B: Tracing back integration test results to the model representations.
- Required for DSL B, model debugging feature in case of designing interactive documentation applications.

12.3.5. Model Non-Conformance

How can a (partly) invalid textual model be (partly) visualized in other views?

12.4. Demonstrator

Demonstrators for all three DSLs are already available in varying degrees of maturity. DSL B was demonstrated at the 1st review meeting (beginning of slide 126 of this presentation). It shows a unidirectional approach to blended modeling, where the textual DSL aspect remains the main actor and the graphical aspect is considered as a specialized view to present certain aspects that are difficult to cover by the textual aspects in a user-friendly way. More information on the principles applied can be found here.

DSL A should be ready to demonstrate for the 2nd review session. Focus should be on demonstrating intra-departmental collaboration using textual and graphical DSL aspects that also support model diff/merge functions for both aspects. Risks: graphical model diff/merge remains challenging also within the scope of the project.

DSL C could be considered for the 3rd review meeting. It could even go beyond the traditional blended modeling use cases where a textual representation is combined with a graphical representation such as 2D diagrams. Instead, 3D CAD models including animation and online connections to the referenced devices should play a central role here.

Depending on the level of maturity reached, the cross-DSL relationships/dependencies could be demonstrated at either the 2nd and/or the 3rd review meeting. The focus of all demonstrations should be on blended modeling and why BUMBLE/HybriDLUX improves the maintainability of model evolution for blended modeling.

The DSL B concepts currently in use were presented by EclipseSource in a previous workshop and are discussed as compatible in relation to the BUMBLE architecture presented above.

13. UC12 - Agile V-model System Architecture

13.1. Background

The V-model System Architecture is of great interest to Pictor because of its role in fulfilling functional safety-standard requirements. There could be great benefits achieved in the safety critical domain if agile methods are researched and implemented. However, the few years of research in this focus suggests that there are challenges in combining safety critical systems and agile methods. The solutions to these challenges represent an opportunity to be industrialized and applied transversely in many different sectors. Several studies have attempted to balance the agile methods with support for the V-model of safety cases. This use case is ideal for the BUMBLE project, because the solution may lie in bridging the existing gaps with the development of better blended modelling of graphical and textual representation that can be used for the safety inspections.

Our focus for this use case is in regards with public infrastructure for civil engineering and their safety requirements, with a special interest in bridge design for roads and railway tracks. The safety inspections have several steps including the drawings and calculation of many types of stress, deformation, and reactions under a growing number of combinations of static and dynamic loads.

There is modelling used in proprietary definitions for various applications systems like the actual SW package used for structural analysis work and the physical design object for civil engineering. Currently there is a lack of usage of modelling tools due to lack of tools support, which provides an opportunity. The lack of software functionality results in costly and error-prone manual calculations.

It is of note that modelling work is often made ad-hoc. Also, a large SW base for the legacy that needs to be maintained in industrial usage. There are several DSML for the application domain depending on the program vendor. For checking and consistency of Fortran programs there is FTNCHEK with options for producing artifacts. There is a call graph option that produces a *.vcg* file which can be further processed in other tools. The tools are graphical editors and analysers.

The table-based specifications of geometry and characteristics of the members and hinges. The target system is a model of building structure with thousands of elements and hinges in different materialsFor example a concrete bridge with steel reinforcement and prestressing cables. These target systems are subject to norms that must fulfill high safety standards for public transportation in European Union member states. The European norms “EN 1992” for concrete structures are updated with more advanced practices in civil engineering for building bridges in concrete and steel.

New civil engineering norms and practices impose new requirements on the software package. These requirements must always fulfill technical aspects such as correctness and system safety for the calculation of structural analysis. In total more than 100 man-years have been in the development of just one of the software packages.

13.2. Use Case Description

13.2.1. Description and Rationale

There is a need to generalize the domain specific language and interwork with other tools.

The confirmed customer project is a consultancy company in Sweden with structural engineering projects using software packages for the structural analysis of large bridges and tunnels. The structural analysis software packages are used for calculation of gross forces and different load cases that are combined into load combinations. The result is used to check the dimensioning and safety limits of the design of civil engineering structures.

First discussions with the customer have taken place and the customer requires a blended modelling solution that just BUMBLE can provide. The domain specific language is based on textual input in the form of a table for the geometric elements in concrete and steel. These tables are translated into 3D-geometric shapes. There are tables with proprietary formats for definitions of geometry and elements for structural engineering. This can provide significant added value for civil engineering firms.

Overall, there is an opportunity for the BUMBLE project to solve a major weakness in the SW tools currently used in modern civil engineering.

13.2.2. User Stories

The following user stories detail the kinds of actions and benefits envisioned by a collaborative blended environment. It is assumed that only one version of each language is deployed at a time and accessible through the website.

Modelling and Model Management

- As a modelling user, I can login to a website, so that I can identify myself and get access to a blended collaborative modelling environment.
- As a modelling user, I can navigate the existing models on a website, so that models can be found with a low threshold.
- As a modelling user, I can manage a hierarchy/organisation of models (folders/packages, as well as model roots), to achieve a maintainable organisation of the modelling content.
- As a modelling user, I can tag model versions, so that they can be used as snapshots for later reference.
- As a modelling administrator, I want to set access levels for models, so that these models are available to a particular set of users.
- As a modelling user, I can generate/download/deploy modelling artifacts, so that modelling artifacts can be used outside of the modelling environment.
- As a modelling user, I can start/perform analysis on a model, to check for the model for certain properties (correctness etc.)
- As a modelling user, I can see errors and feedback (if any) in the model editor, so that I can quickly identify issues in the model.
- As a modelling user, I can see an overview of errors and feedback (if any) in an overview, so that I can quickly identify issues in the project.

Blended Modelling

- As a modelling user, I can view the model through my selected projection, so that I can simplify/extend the information shown in the model based on my needs.
- As a modelling user, I can see my model in multiple views, with different projections in 3-D, so that I can focus on the structure and particular details at the same time.
- As a modelling user, I can use textual syntax (with highlighting, completion, cross-referencing) within a graphical (diagrammatic/tabular) model.

- As a language engineer, I can set the default view of a model (entity) to a particular projection, so that I can simplify/extend the information shown in the model based on my needs.
- As a modelling user, I can edit text, tables, diagrams, and forms in my model, so that I have the freedom to choose the most effective representation.

Language Development

- As a language engineer, I can deploy a new language version, so that the model users can make use of the new language features.
- As a language engineer, I can perform language migrations, so that the models become consistent with the new language.

Model Collaboration

- As a modelling user, I can see the current state of the model when I am connected to the modelling environment, so that I am always up to date.
- As a modelling user, I can retrieve and export models from external sources (like Git), so that I can collaborate with external versioning systems.
- As a modelling user, I can apply (free-form text) reviewing annotations to the model, so that we can review and track progress.
- As a modelling user, I can see the mutation history of a model, so that the differences over time can be viewed.
- As a modelling user, I can select model versions in the mutation history, so that I can compare the current model to the old model.
- As a modelling user, I can resolve merge conflicts, so that the models remain in a consistent state.
- As a modelling user, I can use a notebook-style view on my models, so that I can mix the content with the description/documentation.
- As a modelling user, I can perform undo actions inside a model, so that I can undo my own changes.

13.3. BUMBLE Features

13.3.1. Blended Syntaxes and Modeling

- Blended syntaxes with synchronization.
- Automatic generation of blended modelling editors for MOF-based DSMLs to support blended graphical-textual modelling.
- Semi-automatic generation of synchronization mechanisms across notations.

13.3.2. Evolution & Traceability

- Support for semi-automatic co-evolution of generated artefacts in response to evolution of the original DSML.
- Automatic generation and maintenance of representation-agnostic traceability links in situ for synchronization and co-evolution purposes.

13.3.3. Model Non-Conformance

The tooling may warn about non-conformance of models vs. the metamodel. Such deviations are inevitable as content evolves, but identifying non-conformance, possibly with correction suggestions, is a helpful feature.

13.4. Demonstrator

A demo with Eclipse will enable showing the complete V-model tools chain from HL modelling included blended modelling steps across the V-model to the system test. We will demonstrate the graphic output of 3D graphs modelling structural engineering design. The current StripGraf can be used as an example of an implementation. There are also other prototypes developed to meet the current for blended modelling of structural design in the civil engineering application domain. The requirements on the demonstrator can be derived from the current tool prototypes for generating graphical representation. The demonstrator should at least have the capability to produce a graphical model of the textual input.

From the textual domain, we will demonstrate

- Textual element with characteristics.
- Geometric specification of frames of interconnected elements.

From graphical domain and textual domain, we will show

- Changes of positions of members in the structure.
- Addition and removal of members in both graphical and textual domain.
- Changing the dimension of members.
- Changes hinge type.

Figure 5 shows a possible graphical view of the envisioned blended modelling solution. The symbols connecting the elements are hinges. Different types of hinges are listed as graphical symbols.

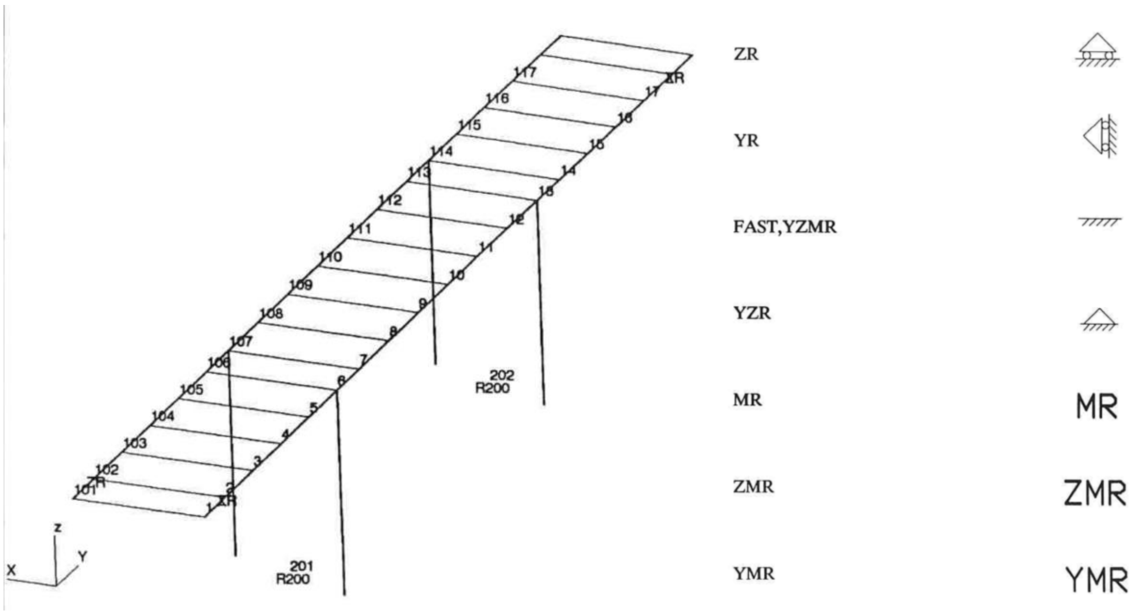


Figure 5. Example graphical view for a DSML instance.