# MetaEdit+ for Collaborative Language Engineering and Language Use (Tool Demo)

Juha-Pekka Tolvanen

MetaCase
Ylistönmäentie 31, FI-40500 Jyväskylä, Finland
jpt@metacase.com

## Abstract

Almost all software development activities require collaboration and language engineering is no exception. First, there is a need for collaboration among language engineers as it is not realistic to expect one man to master all. Second, there is a natural need for collaboration among language users. Finally, there is a need for collaboration among language engineers and language users: Not only when languages are originally designed but more importantly when they are maintained along with the work already created with them. Unfortunately too often tools ignore collaboration by unnecessarily splitting the work into separate formats, tools and roles. We describe and demonstrate collaborative tool capabilities implemented into MetaEdit+ tool and describe experiences on their use in practice.

***Categories and Subject Descriptors*** D.2.2 [**Software Engineering**] Design Tools and Techniques - user interfaces, state diagrams D.2.6 [**Software Engineering**] Programming Environments - programmer workbench, graphical environments D.3.2 [**Programming Languages**] Language Classifications - Specialized application languages, very high-level languages

***General Terms*** Design, Languages.

***Keywords*** Language Engineering, Language Workbench, Metamodeling, MetaEdit+, Collaboration

## 1. Introduction

Almost all software development activities require collaboration and language engineering is no exception. In modern language workbenches collaboration comes in three ways. First, language engineers want to create, edit and check the same shared language specification, with minimal time taken away from the actual work by things like handling conflicts, running diff and merge activities, and fighting with tools. Second, developers using the languages for system and software development want to collaborate in a similar way. Last and most importantly to language engineering, collaboration is also needed between language engineers and language users. All three ways of collaboration are crucial for any larger project as it is not realistic to expect a single person to handle everything. We claim that tools in particular must enable collaboration between both roles: Allow language engineers to get quickly feedback on language design – also when the language is already in use. This helps to evaluate the impact of language changes. Similarly language users should be able to access the metamodel (depending on rights given) and at least propose changes. We describe and demonstrate proven approaches for collaborative working with models and metamodels (language specifications). These collaborative tool features are implemented in MetaEdit+ [3].

We start from collaborative language creation (metamodeling) showing how it can be used in typical language engineering situations as well as describe the benefits it can provide. The same is done for collaborative language use, aka modeling. Finally we move to collaboration among language users and language engineers. We conclude by sharing experiences how this kind of collaboration enables scalability in terms of multiple engineers, multiple languages and large models as applied in industrial cases.

## 2. Collaborative Language Engineering

A language definition contains several parts: Abstract syntax, concrete syntax and semantics like constraints and transformations. Unfortunately these parts are often defined in separate activities, by different engineers and the results are specified into own documents with dedicated formalisms. Lack of collaboration and disconnected specifications then easily lead to inconsistencies and incompleteness. An example of this is UML standard as it contains more than 300 errors in abstract syntax and related constraints (expressed in OCL) [1, 6]. A better way would be to keep the language specification integrated and accessible for all language engineers involved. Several language engineers could then work with the same integrated language specification and changes could be traced among its parts.

Figure 1 illustrates this idea using ETSI's TDL (Test Description Language) as an example. Top left (A) shows a fragment of the abstract syntax with an element called 'ComponentInstance'. Top right B) shows a constraint related to the uniqueness of name. Bottom left (C) illus-

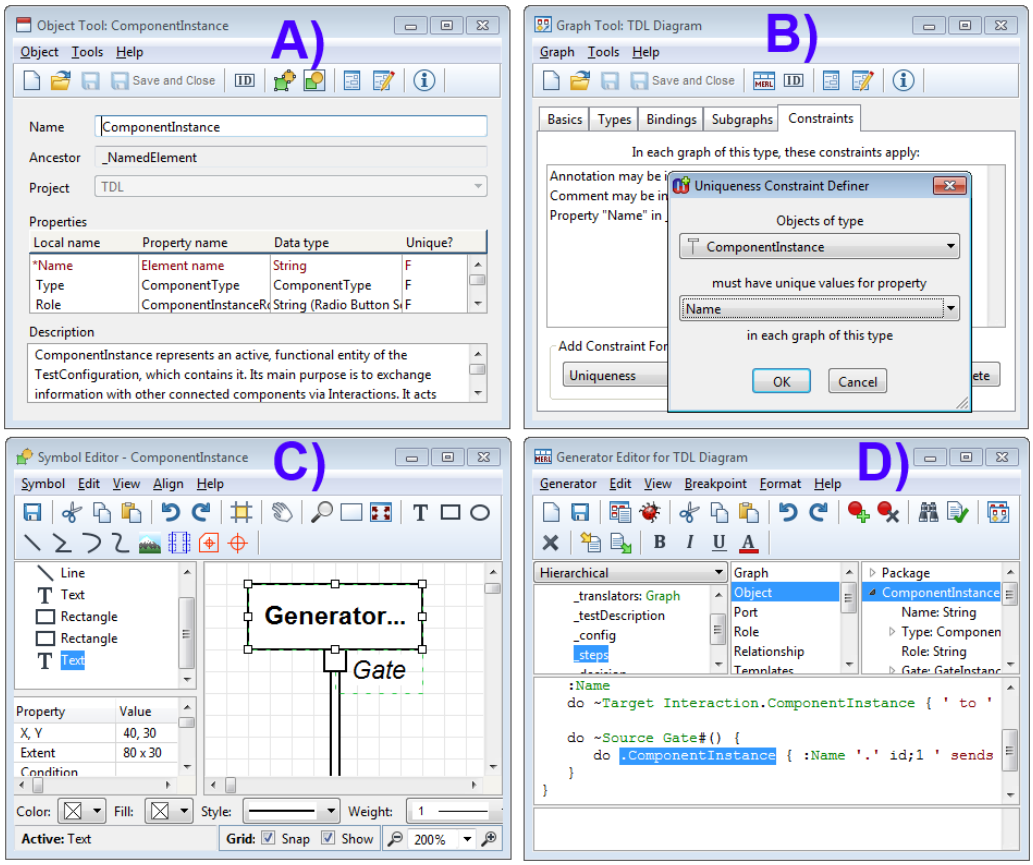


**Figure 1** Collaborative specification of TDL: A) abstract syntax of 'ComponentInstance', B) constraints and rules C) concrete syntax of 'ComponentInstance', and D) TDL generator accessing component instances.

trates how the concrete syntax for component instance is defined from various symbol elements. Finally, bottom right (D) shows a related TDL generator which highlights the component instances accessed from TDL model when the generator is executed. All these four parts of the language definition are integrated in the tool so any changes e.g. in abstract syntax automatically reflects or can be traced to the constraint (B), notation (C) and generator (D).

As an extreme case of collaboration, each part of the language could be defined at the same time by separate language engineers. This kind of collaborative capability supports the typical practice that some language engineers define abstract syntax whereas others can define notation or generators. Also, when defining several languages that are integrated, it is a necessity that integration links between the multiple languages can be defined. This type of collaborative tooling for language engineering provides several benefits:

- Languages and generators can be checked early while been defined. As in any team work, several persons can see more than one, can discuss about language definition and test it in collaboration. This leads to better quality languages.
- Development of languages and generators is faster: Not only because different generators can be developed by different engineers, but because things like notation and some of the checking rules do not need to be completely ready before making generators.
- Speeds up the move to modeling since while generators are being still developed modeling can already start. This is particularly relevant for shorter term projects in which languages are needed quickly (in days).

MetaEdit+ supports collaboration among several language engineers and gives different options on how collaboration can be done: e.g. if others can use the language at the same time while it is been defined, if other language engineers can define related languages, or if several engineers can edit the same language definition in the same repository. When working in the same collaborative environment MetaEdit+ gives also possibility to give access rights to language engineers.

## 3. Collaborative Language Use

The needs for collaborative modeling are basically similar than for collaborative metamodeling, but now the scalability aspects come to play as there are more persons involved, more shared elements and more languages used to specify the models. Basically, collaborative modeling allows engineers to edit the same models at the same time. To make this happen and keep models formal at the same time (rather than plain drawings), collaborative tools apply locking scheme at some level of the model. To provide usable collaboration the granularity for locking must be small. This way several persons can edit even the same diagram (matrix, tree etc.) at the same time. The pessimistic concurrency control is widely used in repositories and

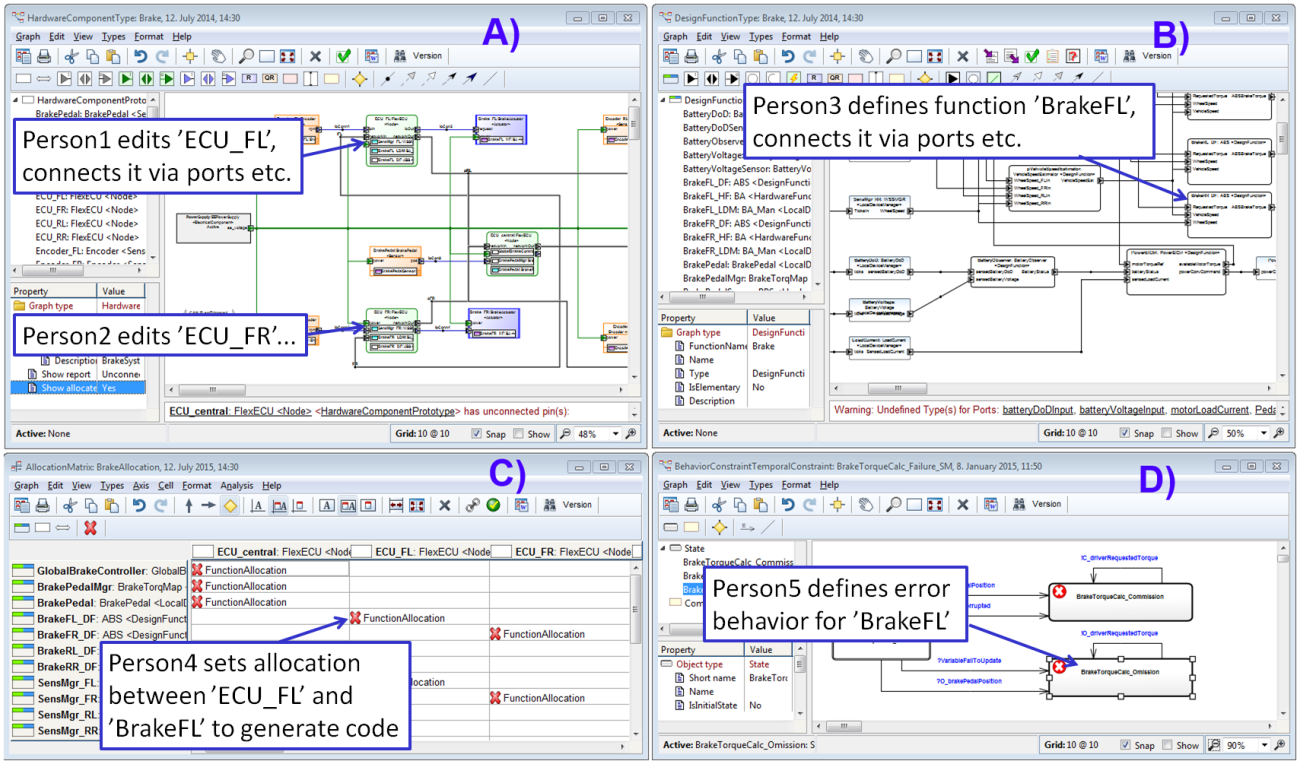


**Figure 2** Collaborative modeling with automotive DSLs: A) functional architecture, B) hardware architecture, C) allocation for AUTOSAR generation, D) behavior.

databases avoiding handling conflicts and performing merge for collaboratively edited models.

Figure 2 provides an example of collaborative editing in MetaEdit+ and adds one aspect of scalability to the scheme: Collaboration is based on several yet integrated languages (see [4] for details). First, in top left A) two persons are editing the same diagram and different hardware components there. At the same time in B) person3 defines the logical component of the system and then in C) a person4 defines the allocation between hardware and logical architecture using the very same components been edited in A) and B) to generate software allocations. Finally, in D) safety engineer as person5 defines error models for the logical component been defined in B). This kind of collaborative modeling provides several benefits:

- Team can work in the same design space in parallel (as the case in Fig 2). If feedback or opinions are needed from colleagues all can see the same model (or model element) immediately and update the model.
- No time and effort is needed to handle conflicts, like run diff and merge after having made changes.
- Development is faster as all model information is available as needed.
- Work from multiple views, such as hardware, timing, analysis function etc., that are often accessed partly on from different languages can be integrated at the whole model level.
- Trace is available among views and model elements, like see in C) where BrakeFL is been defined.
- Models can be checked and verified early – not based on one diagram only but on larger model which may be based on different languages.

When collaborative editing is not wanted, like developer takes some models home, those parts can be exported and used in own copy and later integrated back – with a payback of requiring doing diff and merging that in collaborative modeling environment was handled automatically. MetaEdit+ provides also change and difference analysis and integration with versioning systems.

## 4. Collaboration Among Language Engineers and Language Users

Finally and most importantly for language engineering, collaboration is not limited among language developers only as language users can apply the developed language at the same time. In MetaEdit+ the collaborative tools allow language engineers and language users share the same repository and make changes there in parallel. There are no compilation, packaging, installation etc. activities needed for a language delivery. Language users do not need either do any extra actions to get the new language version. Figure 3 illustrates these features: On the left is a portion of language definition and on the right is the language in use. If the abstract syntax or concrete syntax of the 'Sensor' is changed (language element on the left) its influence to the models can be immediately tested by using the language or by inspecting the existing models. Saving the language specification makes it then available for all language users (as on the right). The same for constraints, rules and transformations related to the language. Tight collaboration

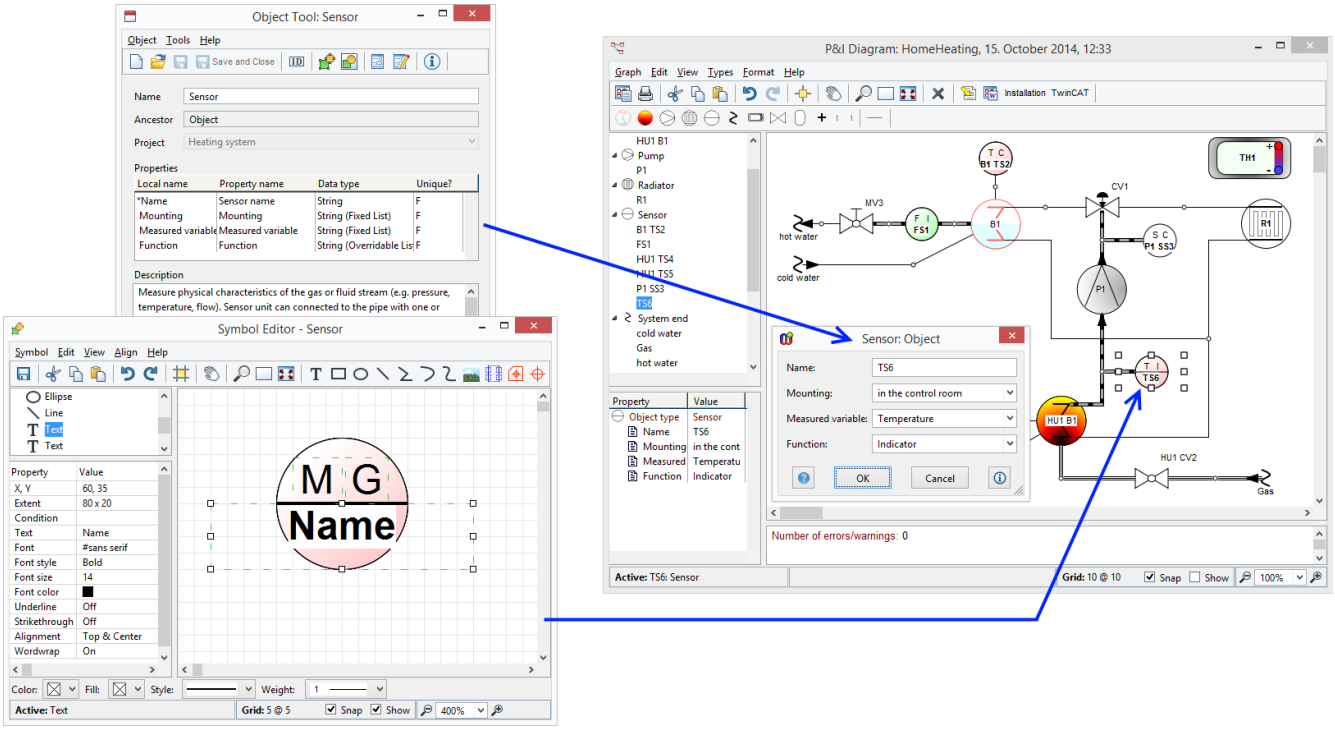


**Figure 3.** Collaboration among language engineer (left) and language user (right)

between language engineers and language users provides several benefits:

- Enable early feedback and validation of language definition. Users may immediately test the language and not only verify that the definition is correct, but also validate that the language allows specifying the kinds of things for which it is intended.
- Minimize the risk of creating the wrong language constructs and enable the language to be defined in small increments. This is particularly important if the domain is new, evolving, or the language engineers do not have prior experience on language engineering.
- Language adoption and acceptance improves since language users are involved early in the language creation.

Finally and most remarkable for language engineering is that in MetaEdit+ the models already created update automatically to the new language version and can be immediately edited in various editors. Using the sensor from Figure 3 as an example: If the name of the 'Sensor', or any of it's property types are modified (added, renamed or removed), the modification is updated automatically to models done – without any further action expected from the language users. This is highly valuable for industrial use with a lot of models and language users. It is also particularly relevant for DSLs because they evolve more frequently than general-purpose languages and because the most of the work, often 90% on used DSLs, deal with the maintenance. Therefore it is important that language engineering tools manage the 90% to avoid horror stories like [5] stating that developers needed to wait 5 months to get their models updated when the DSL implemented with Eclipse EMF&GMF changed.

## 5. Experiences and Concluding Remarks

Collaborative features must also address scalability needs that are crucial in industrial-scale. Not only developers can work together in the same model, but they can use several integrated modeling languages. Such integrated languages have been successfully used with MetaEdit+ in industries like automotive where is a need to specify logical, hardware, safety, software etc. aspects in an integrated manner as in Fig 2. A move from editing a single file to a repository approach enables scaling to bigger models too, like millions of model elements [2]. As witnessed in many industries (finance, telecom, IT) and application areas, repositories can handle large number of data and transactions. Similarly, lazy loading provides fast access to models and when running transformations the model data is easily available as there is no need to read multiple files, parse them and create internal memory structures to execute transformations.

We have experiences on applying collaborative features for language engineering in several projects: e.g. in automotive a large language (with over 20 integrated languages) was defined collaboratively by 3 language engineers. Also we have applied collaborative tools during language creation in scenarios where 2 engineers modify the languages based on the feedback given by 20 developers using the languages at the same time. This has been working well and leads almost automatically to better quality languages that meet the users' needs. Move to collaboration provides benefits beyond modeling.

While the space does not permit describing further details an evaluation version of MetaEdit+ along with tutorials, user guides and sample languages is available to download from www.metacase.com.

## References


[1] Bauerdick, H., Gogolla, M., Gutsche, F., Detecting OCL Traps in the UML 2.0 Superstructure: Experience Report. Proceedings of Unified Modeling Language - Modeling Languages and Applications (UML 2004), LNCS 3273, Springer (2004)

[2] Kelly. S., Mature Model Management: More than just XML under Version Control, "Ausgereiftes Modell-management: Mehr als nur XML unter Versionskon-trolle", OBJEKTspektrum, October 2010, www.metacase.com/papers/Mature_Model_Manageme nt.html

[3] MetaCase, MetaEdit+ Users Guide, 2016, www.metacase.com/support/

[4] MetaCase, EAST-ADL Tutorial, 2016, www.metacase.com/papers/MetaEditPlus_Tutorial_for _EAST-ADL.pdf

[5] Warmer, J., Bast, W., Developing an Insurance Product Modeling Workbench, Code Generation Conference, Cambridge, 2011.

[6] Wilke, C., Demuth, B., UML is still inconsistent! How to improve OCL Constraints in the UML 2.3 Super-structure, Proceedings of OCL and Textual Modelling workshop, 2011.