Collaborative Creation and Versioning of Modeling Languages with MetaEdit+

Steven Kelly
MetaCase
Jyväskylä, Finland
stevek@metacase.com

Juha-Pekka Tolvanen
MetaCase
Jyväskylä, Finland
jpt@metacase.com

ABSTRACT

Language engineering is a collaborative endeavor with several people creating and using the modeling languages and related generators. We describe key features of the MetaEdit+ multi-user version that support collaborative language engineering, evolution and use. Several language engineers can create and refine the same modeling language at the same time, see each other's changes, and version language definitions to version control systems. A key characteristic of MetaEdit+ is continuous integration of language definitions and of models created with the languages.

CCS CONCEPTS

• **Software and its engineering → Model-driven software engineering**; **Domain specific languages**; **Collaboration in software development**; *Integrated and visual development environments*; *Application specific development environments*; *Software version control*;

KEYWORDS

Domain-specific language, metamodeling, code generation, version control, collaboration

ACM Reference Format:
Steven Kelly and Juha-Pekka Tolvanen. 2018. Collaborative Creation and Versioning of Modeling Languages with MetaEdit+. In *ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS '18 Companion), October 14–19, 2018, Copenhagen, Denmark.* ACM, New York, NY, USA, Article 4, 5 pages. https://doi.org/10.1145/3270112.3270132

1 INTRODUCTION

Language definitions follow a similar lifecycle to other software development artifacts such as code, models, requirements, tests etc. They are created, edited, and maintained over time, typically with multiple people collaborating in their editing and reviewing. We can also identify some characteristics particular to modeling language engineering, which influence the collaboration:

- Language engineering teams are typically smaller than software development teams, usually just one or a few people.
- Changes to the language definition influence existing software development work — models made with a language need to be updated to follow the changed language. The effort to update existing work should be minimal and existing model information should not be lost during the language update — unless there has been a clear objective to do so.
- Customers of language engineers are often professionals in their own field: software engineers or other experts like test engineers, safety engineers, insurance specialists, interaction designers etc.
- Language users are normally more closely involved in the definition process, and typically more active in giving feedback, than customers and users in traditional software development projects.

Modeling language engineering tools should support the collaboration and management of language definitions to ensure creation of high-quality languages, their maintenance and evolution along with models already made with them.

We describe key functionality provided by MetaEdit+ (5.5 SR1, [4, 8]) for collaborative creation, management and versioning of evolving modeling languages, generators and models[1]. We first introduce MetaEdit+ and its collaboration features and then focus on language engineering functionality: concurrent collaboration among language engineers, integration of language definitions, management of language engineering rights, tracing language changes and integration with version control systems.

2 METAEDIT+ AND COLLABORATIVE WORK

MetaEdit+ is a mature, commercial language workbench that supports graphical, matrix and table-based Domain-Specific Modeling (DSM) languages. It offers projectional editors with manual and automatic layout, a generator system with debugger, various browsers, programmable APIs and integration with programming environments and version control systems. MetaEdit+ has been used to define hundreds, if not thousands, of modeling languages and code generators along with tool support[2].

For collaboration MetaEdit+ provides a multi-user version, where users can share and edit the same models at the same time. Their work is continuously integrated without additionally launched merging steps or locking out modelers when someone is editing the model. The modeling history and changes are automatically recorded and traceable. Fig. 1 shows the tool support for three alternative ways to view changes made in models (tree, graphical,

MODELS '18 Companion, October 14–19, 2018, Copenhagen, Denmark

ACM ISBN 978-1-4503-5965-8/18/10…$15.00
https://doi.org/10.1145/3270112.3270132

[1] A short video demonstrating collaborative language engineering and versioning to GitHub is available at https://vimeo.com/279657668 with password: Models2018
[2] For over 100 public examples, see DSL of the week: https://twitter.com/search?q="DSL of the week"

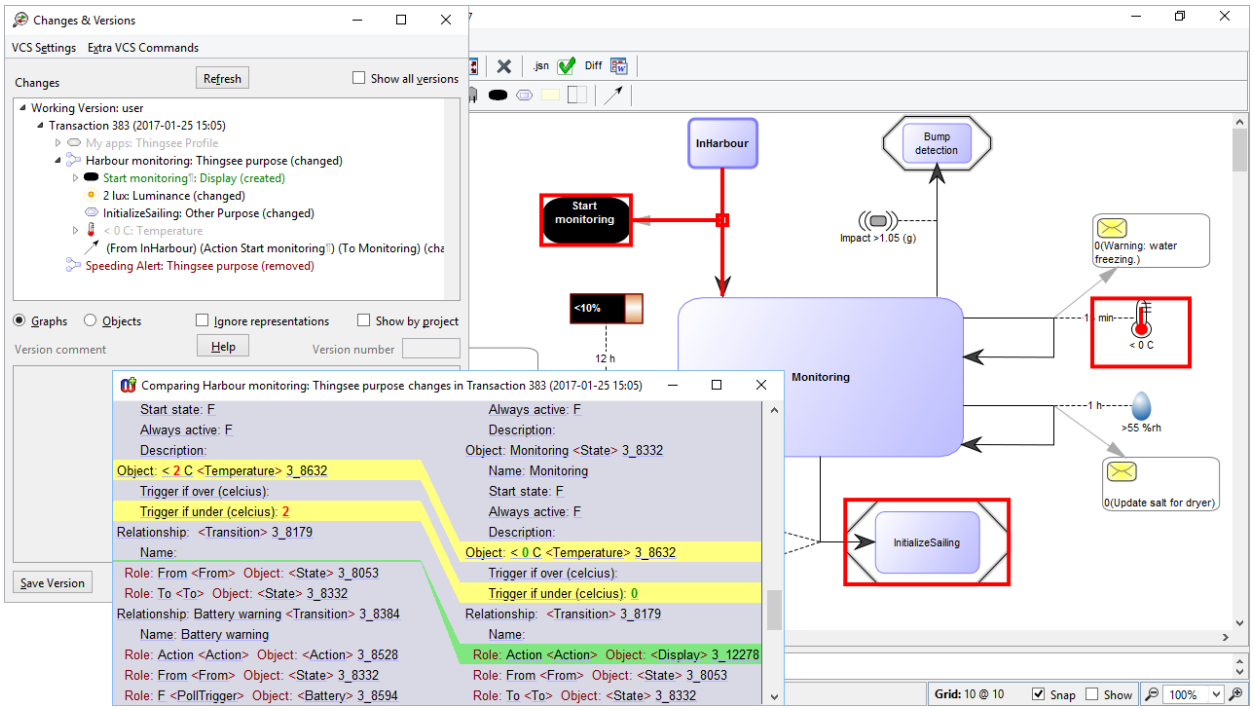

Figure 1: Viewing model changes and modeling history in MetaEdit+

and textual). Both individual modelers and multi-user modelers can version their work to version control systems like Git and SVN. For a detailed description of combining concurrent collaborative modeling with any external version control system see [6] and Sections 3.4–3.5 in the MetaEdit+ 5.5 User's Guide [8]. MetaEdit+ is available to download at https://www.metacase.com/download.

3 COLLABORATIVE LANGUAGE ENGINEERING

For language engineering MetaEdit+ provides similar collaborative capabilities as for language use: Multiple people can jointly define languages and the language definitions are continuously integrated. When one or more of the language engineers saves an update to the definition, the updated language is immediately supported in the modeling editors.

3.1 Language evolution

An additional requirement for language definition with multiple users is that changes made in the language definition are automatically reflected to existing models. The design rationale in MetaEdit+ is that existing models must always remain usable as the languages evolves. A few examples of typical maintenance tasks and how MetaEdit+ handles them:

(1) Defining and adding new concepts to a language, and adding new properties to concepts are handled automatically.
(2) Renaming of a language element is automatically reflected to existing models: they follow the new name. Also renaming in the abstract syntax is automatically updated to the concrete syntax and constraints.
(3) Deleting concepts from a language is handled automatically and non-destructively: new instances of them cannot be created; the old concepts can be obsoleted, marking their instances visually. Existing generators produce correct output, allowing language engineers to set their own timetable for when modelers must have removed instances of old concepts.
(4) To support human migration of the models, language engineers can make checking reports and model annotations that show which elements require update.

3.2 Collaborative language definition

Fig. 2 illustrates the collaborative features of MetaEdit+ in a language engineering scenario among four language engineers. All language engineers are defining the same language, each focusing on different, yet integrated, parts of the language definition. First, language engineer Bob refines a language concept 'State', adding properties such as 'DisplayFn' referring to a display definition. While Bob is working, Jill starts editing the constraints, e.g. 'Alarm' is allowed to be in only one 'From' role. In the currently open window Jill adds a uniqueness constraint for state names: There should not be two states with the same name. Next, Bob saves his work, Jill continues defining constraints, and Jim joins in by starting to define concrete syntaxes. He uses the Symbol Editor to define notation for the language element 'Alarm' for which Jill set constraints. Finally, Jill commits her work and Tim joins in to define a code generator producing C code from the models. The

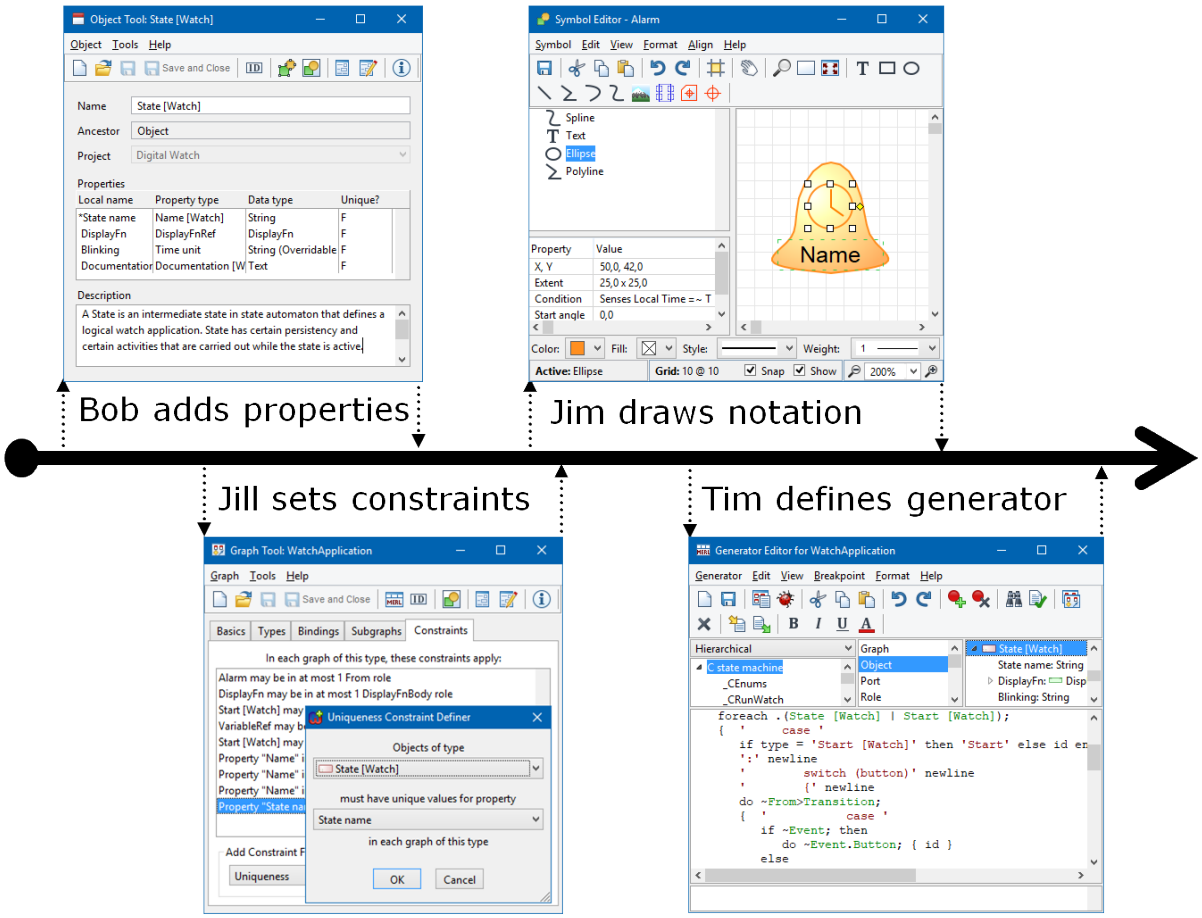

Figure 2: Collaboration and continuously integrated language definition

Generator Editor shows how it accesses the State information that Bob refined earlier.

While the above scenario is perhaps extreme with four persons defining the same language and related generator, the tight collaboration is typical between the roles of language definer and generator developer. Collaboration in MetaEdit+ is not restricted to language engineers: language users may also apply and test the languages while they are being defined. Enabling close user participation is known to improve the quality and acceptance of results, in language engineering [5] as in development in general [7].

Collaborative language engineering is also particularly useful when defining several languages which are integrated, sharing some elements from the metamodel, generator modules or notational symbols. For example, in the most complex case we are aware of, MetaEdit+ and its collaborative features have been applied while defining 24 languages — each focusing on a different view of the system yet in an integrated manner.

Keeping the language definition integrated is particularly important to the process of language engineering and to the quality of languages. For example, languages whose parts have been defined separately have been shown to have many errors. For example, the definition of UML is found to contain hundreds of errors between its metamodel and constraints [1, 10]. Another example is ArchiMate [9], whose versions have repeatedly had dozens of errors (see e.g. error addendums for 2.0 and 3.0). Enabling collaboration between team members and integration between language definitions, as in MetaEdit+, helps teams to create better quality languages.

3.3 Collaborative definition of tool behavior

Collaboration in MetaEdit+ is not solely focused on language and generator definition (as in Fig. 2), but also covers aspects related to tool behavior, such as dialogs, icons for toolbars and tree views, and implementing integration with other tools in the tool-chain. Their editing in MetaEdit+ is collaborative, as with the language definition. Fig. 3 shows examples of the Icon Editor and Dialog Editor for a relationship 'Roll' used in the language. Both of these are optional customization as MetaEdit+ automatically provides default, automatically updated icons and dialogs based on the language definition.

3.4 Language engineering rights

To manage language creation and editing MetaEdit+ provides a set of options for language engineering rights.

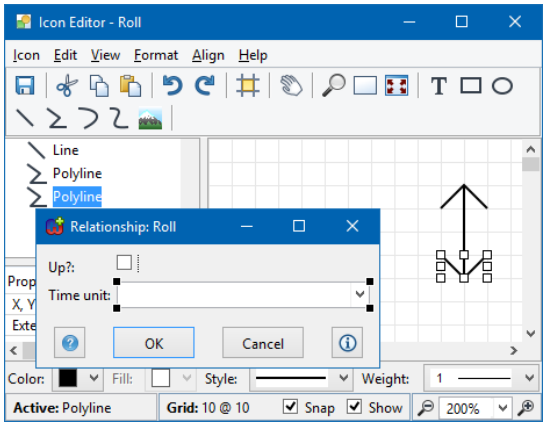

Figure 3: Icon and Dialog Editor

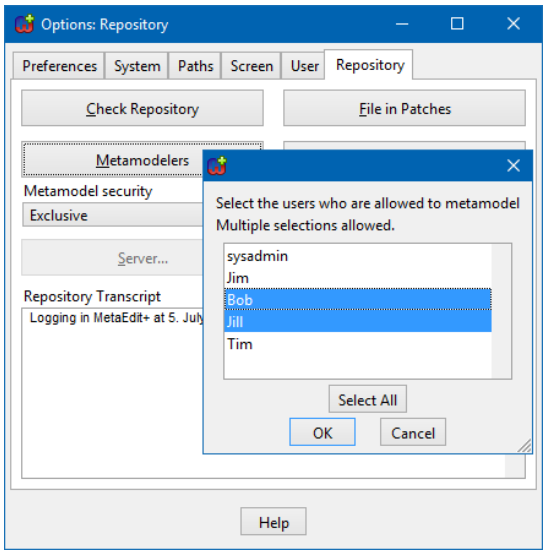

Figure 4: Setting rights for language engineers

3.4.1 Metamodeling rights per user. One clear requirement has been that not all users should be allowed to define languages. For that purpose administrators can specify which users have access to language definition tools or can change language definitions. The former requirement is supported with MetaEdit+ Workbench providing the language engineering tools. Even if all users have Workbench tool, administrators can set rights for individual persons, as shown in Fig. 4 where metamodeling rights are granted for Bob and Jill.

3.4.2 Graphical metamodel rights per project. MetaEdit+ allows metamodels to be defined graphically. Rights to such graphical metamodels, as for models, can be assigned per project, per user.

3.4.3 Metamodeling collaboration. The granularity and volatility of changes in the language definition tends to vary a lot during the life-cycle of the language: typically a lot of changes in the beginning, less in later phases, yet bigger refinements in the middle

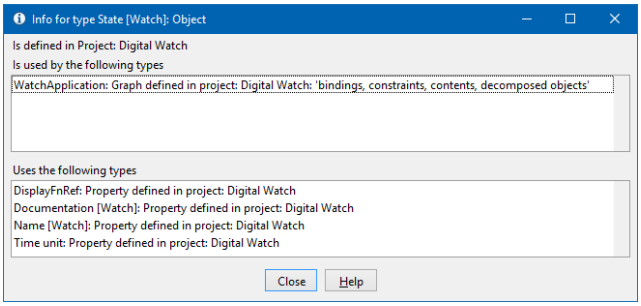

Figure 5: Info on metamodel element 'State'

once the domain changes or there are new generation needs. While the changes can be done in a single-user version of MetaEdit+ and exported to the multi-user version, often the refinement changes are done directly in the multi-user version. For managing language modifications MetaEdit+ provides four metamodel security levels which administrators can set. These are:

- Exclusive: the language engineer must be the only user in the repository. This allows the language engineer to update all the models without considering language users' work at the same time. An example is updating models with the API. It also protects users from the sometimes disconcerting effects of seeing the language changing under their feet.
- Single: there can be only one language engineer in the whole repository at a time, but any number of simultaneous language users. This allows language engineers to work on their own languages without considering other language engineers' work.
- Project: only one language engineer can access a specific project, but any number of language users may work in the project, and other language engineers may work in other projects. This is typical when language definitions are shared for different language engineers and their projects.
- Type: any number of language engineers and language users can access all projects and types, and saving a change to a particular language element locks it first.

3.5 Managing language definitions

In addition to direct language definition, MetaEdit+ provides tools for managing and browsing the language definitions as well as a shared symbol library for reusing notational elements. The Metamodel Browser and Type Browser allow language engineers to view, access and remove elements of the languages. They also provide tools to trace among language elements, e.g. to inspect where a particular language element is used, or what elements are used by it. Fig. 5 shows such information for the 'State' element defined in Fig. 2. This concept is used in the 'WatchApplication' language in the 'Digital Watch' project. It is used in bindings of relationships, as one of the contents of WatchApplication graphs, it is referenced in constraints, and in a decomposition structure for nested states. 'State' also uses four other language elements, namely 'DisplayFnRef', 'Documentation', 'Name' and 'TimeUnit'. For all these items further similar traces can be launched from the Info tool.

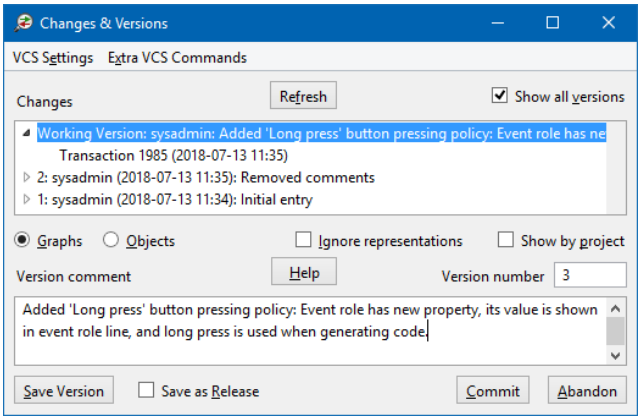

Figure 6: Creating a new version of the language definition

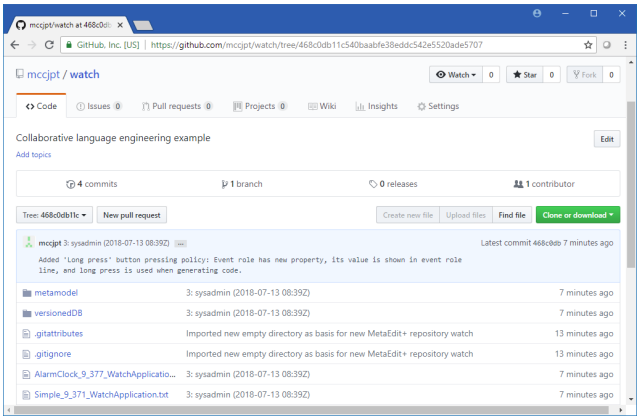

Figure 7: Inspecting a versioned DSM solution in GitHub

4 VERSIONING A DSM SOLUTION

A DSM solution consists of many parts, e.g. individual languages of the whole modeling solution, or for a given language, its metamodel, generators, individual notational symbols, etc.. Rather than split these interlinked and interdependent parts and version them separately, MetaEdit+ enables to treat them together. This makes versioning easier and ensures consistency among the parts of the language across versions.

Versioning of a DSM solution follows the same principles as versioning of models. If the metamodels are defined using a graphical metamodeling language [8], its management is identical to versioning and managing models, with history and viewing changes graphically, textually or with a tree (see Fig. 1).

Fig 6. shows how a new version is created and saved to an external version control system. The Language engineer has entered a version number and version comment. Pressing 'Save Version' commits the changes made to the MetaEdit+ repository and push the version to the chosen version control system. In addition to versioning the language definition, the versioning function also adds a textual human-readable description of the language, the generator source code, and the notation library symbols as SVG. In our example, Fig. 7 shows the view from GitHub after Save Version in MetaEdit+: the metamodel directory includes these diffable forms of the language definition for inspection with standard GitHub tools. Note that models are versioned too, as the change in the language definition also influenced them. This is because the language engineer added a Boolean property 'Long press' to 'Event' roles, and so all models with instances of 'Event' would get a new 'Long press' property with the default value.

The collaboration and versioning approach of MetaEdit+ is made to ensure continuous integration and consistency: within the language definition as well as between the model versions. Since both models and metamodels are versioned together, their internal consistency can be managed better than if versioned as separate parts. In this way MetaEdit+ can ensure that any model version is always used with the correct language definition version. However, when necessary a different version of the metamodel can be checked out and applied to the models. MetaEdit+ updates the models automatically, following the same behavior as for language evolution.

5 RELATED WORK

In GMF-based editors, a change to one part of a GMF language definition may often invalidate another part, breaking editors [3]. Perhaps worse, many changes to GMF language definitions will break existing models [2]. The situation is similar in many other tools. A change in GMF or Microsoft DSL Tools version also often invalidates an existing language definition, breaking editors.

6 CONCLUSIONS

MetaEdit+ supports collaborative engineering, evolution and use of languages. The multi-user environment makes sure all language engineers' work is integrated, and tooling updates automatically. There is no need to manually fetch others' changes, deal with diff, merge, and conflicts, or think about any of these details when versioning. Since metamodels are integrated with models, the approach also integrates language versions with model versions. Where integration with a version control system is desired, simply entering a version number and comment is enough. This makes language definition, use and versioning easier, and collaboration enjoyable.

REFERENCES

[1] H. Bauerdick, M. Gogolla, and F. Gutsche. 2004. Detecting OCL Traps in the UML 2.0 Superstructure: Experience Report. In Proceedings of Unified Modeling Language - Modeling Languages and Applications (UML 2004), LNCS 3273, Springer.
[2] J. Di Rocco, D. Di Ruscio, A. Pierantoni, and L. Iovino. 2015. Supporting Users to Manage Breaking and Unresolvable Changes in Coupled Evolution. In DSM 2015, SPLASH. http://www.dsmforum.org/events/dsm15/Papers/DiRocco.pdf
[3] J. Di Rocco, D. Di Ruscio, H. Narayanankutty, and A. Pierantonio. 2018. Resilience in Sirius Editors. Models and Evolution Workshop, MODELS.
[4] Steven Kelly, Kalle Lyytinen, and Matti Rossi. 1996. MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. In Proceedings of CAiSE'96. Springer.
[5] Steven Kelly and Risto Pohjonen. 2009. Worst Practices for Domain-Specific Modeling, IEEE Software, July/August 2009.
[6] Steven Kelly. 2018. Collaborative Modelling with Version Control. In: M. Seidl, S. Zschaler (eds) Software Technologies: Applications and Foundations. STAF 2017. Lecture Notes in Computer Science, vol 10748. Springer.
[7] J.D. McKeen, T. Guimaraes, and J.C. Wetherbe. 1994. The relationship between user participation and user satisfaction. MIS Quarterly 18, 4, pp. 427-451.
[8] MetaCase. 2018. MetaEdit+ 5.5 User's Guide. Retrieved August 22, 2018 from http://www.metacase.com/support/55/manuals
[9] The Open Group. 2018. ArchiMate Specification, 2.0.1, 3.0.1. Retrieved July, 2018 from http://www.opengroup.org/subjectareas/enterprise/archimate
[10] C. Wilke and B. Demuth. 2011. UML is still inconsistent! In Proceedings of OCL and Textual Modelling workshop.