D. Kuryazov, F. Schmalriede, C. Schönberg, K. Meyer, A. Winter: Mind Maps sans Frontières. In: Proceedings of the 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Workshop on Collaborative Modeling, 2021.

Mind Maps sans Frontières

Dilshod Kuryazov
Software Engineering Department
Urgench branch of TUIT
Urgench, Uzbekistan
kuryazov@se.uol.de

Florian Schmalriede
Software Engineering Group
University of Oldenburg
Oldenburg, Germany
schmalriede@se.uol.de

Christian Schönberg
Software Engineering Group
University of Oldenburg
Oldenburg, Germany
schoenberg@se.uol.de

Kevin Meyer
Software Engineering Group
University of Oldenburg
Oldenburg, Germany
kevin.meyer@uol.de

Andreas Winter
Software Engineering Group
University of Oldenburg
Oldenburg, Germany
winter@se.uol.de

***Abstract*—Collaborative development is becoming more and more important and popular in everyday life. Development of any system or project, idea, design, model, or realization by teams of several experts promises high performance during development and ensures a high quality of the developed system. For this purpose, there is a need for advanced and elaborated tools for collaborative design and development.**

This paper presents the tool *CMCM* (Collaborative Mind- and Concept-Mapping), which is used for collaborative modeling of Mind Maps to show the feasibility of tool collaboration provided by model differencing. *CMCM* manages Mind Maps through models conforming to a self-developed modeling language for Mind Maps. The *CMCM* modeling language differentiates between abstract, diagram and concrete syntax following the DD (Diagram Definition) standard of the OMG. To support collaborative modeling, instances of *CMCM* exchange model differences conforming to the operator-based Difference Language (DL) via the CoMo-X (Collaborative Modeling – eXtend) server. Macro and micro versions of models are equally described with DL and enable asynchronous as well as synchronous collaboration.

Using a self-reflective Mind Map of collaborative Mind Mapping as a use case, the functionality of *CMCM* is demonstrated, the concepts behind *CMCM* are explained, and *CMCM* is evaluated. The underlying approach is shown to be both effective and efficient, as well as transferable to other modeling domains and languages.

***Index Terms*—Collaborative Modeling, Collaborative Mind Mapping, Macro-versioning, Micro-versioning, Real-time Synchronization, Delta Language, Diagram Definition.**

I. Introduction

Collaborative development is the creation of large and complex software systems, projects, designs, models, and ideas (i.e., *artifacts*) by teams of several domain and technical experts. All *artifacts* are produced during evolution and maintenance. The complete model is stored in a central repository; collaborators work simultaneously on distributed, synchronized copies. The team-members usually work on the shared projects in real-time.

There are many modeling languages that are used in the field of software engineering, including the various UML diagram types. In this paper, we focus on another modeling paradigm that stresses the conceptual work of software engineering in general: Mind Maps. As a feasibility study for developing collaborative modeling tools, we present a tool *CMCM* (short for Collaborative Mind- and Concept-Mapping) that allows multiple users to work at possibly distributed locations on the same Mind Map simultaneously – at the same time, or asynchronously – at different times. We show that our tool is both effective and efficient, due to an optimized way to synchronize changes between collaborators.

Depending on the nature of interaction, collaborative development systems can be divided into two main form s, namely concurrent collaborative modeling and sequential collaborative modeling [1], [2].

Synchronous Collaboration. Synchronous collaboration indicates creating, modifying and maintaining huge and centralized artifacts, that are shared as copies among modelers in real-time. The collaborators can be located in different places geographically and work at the same time. The collaborators can access the centralized and shared repository, and can directly modify the artifacts on the central repository or on their local copies. There are several approaches widely used in the synchronous collaborative editing of textual documents, for example Google Docs [3], Etherpad [4], or Overleaf [5].

As long as synchronization of changes occur in real-time, performance of interaction matters. In synchronous collaboration, the changes can be constantly detected by listening to the users' actions on a particular client system. They can be represented in the form of a small set of model changes. These changes are usually not stored, but synchronized directly among collaborators. In contrast to textual document editing, the increased performance of change synchronization is very crucial in synchronous collaborative designing because of the complex graph-like structure of the models. Thus, artifact changes have to be identified continually, and represented and synchronized using very compact notations. Synchronous collaboration is enabled by micro-versioning [6], [7] in this paper.

Asynchronous Collaboration. Asynchronous collaboration intends to identify changes between subsequent revisions of artifacts, stores and reuses these changes when needed.

There are several source code-driven asynchronous collaborative development approaches (also known as version control systems) such as Subversion [8], Git [9], and others. These asynchronous collaborative development systems for source code are the best aid in handling development and evolution of large-scale, complex and continually evolving software systems. The same support is needed for designing and modeling. For instance, while designing projects in asynchronous collaboration, collaborators intend to store the current revision of their model and reopen it at a later date or at another location to continue development. Or collaborators are located in different time zones and work at different times on the project. These model management activities are facilitated by macro-versioning [6], [7] in this paper.

Collaborative modeling requires both synchronous collaboration in real-time and asynchronous collaboration by merging model variants created in a distributed manner. Since managing both micro- and macro-versions is done by model differences, both scenarios can be supported by a common technical approach.

II. Use Case

Mind Mapping is the process of structuring concepts in a graphical manner [10]. Starting with a core concept in the center, related and subordinate concepts are arranged around it on branches. Further concepts are placed on additional branches that originate at any of the existing concepts, forming a radial tree structure. In addition to the hierarchical structure, cross references can show directed connections between concepts. Colors can be used to mark and differentiate separate aspects [11].

Figure 1 shows an example of a Mind Map with four subconcepts around the center and three cross-references. Mind Maps are used for structuring domain knowledge, thought processes and brainstorming sessions. Mind Mapping can be used by individuals in one-on-one sessions, but it is also an appropriate approach to collect and structure ideas together. Mapping the concepts in the structured manner of Mind Maps ensures that all relevant aspects are considered and that all aspects are arranged hierarchically.

As a running example in this paper, we will have the authors map the concepts of collaborative Mind Mapping itself. As not all of us reside in the same location (not even in the same country), and as currently most of us work from home, a tool for online collaborative Mind Mapping is needed. Since collaborative modeling is a group activity that requires discussion in addition to the shared model, at least one open channel of communication is required. These channels are provided separately from our tool approach, however, allowing modelers to use their system of choice for the discussions.

The geographical distance varies widely between us, as does the quality of the internet connection available. This leads to large fluctuations in connection properties like latency and capacity. In turn, this necessitates fast and efficient network transmission of Mind Map data, and a modeling concept that is largely tolerant of transmission delays and re-orderings. Local changes need to be transferred in real-time to other parties with a minimum of data volume.

For multiple users that model at the same time, the system also needs to be robust versus conflicts. During modeling of a Mind Map, many – sometimes conflicting – ideas are put forward and added to the common model. When two changes clash, e.g. because two people change the same concept to different names, the system needs to be able to deal with this situation.

Experiences with collaborative modeling of mind maps, UML class diagrams [6], and UML activity diagrams [7] have shown that conflicts do not affect synchronous collaboration when small changes are shared immediately. The use of a channel for parallel communication between the collaborators also ensures that modeling surprises are avoided.

When multiple persons change aspects of a Mind Map, it is also helpful to know who made a specific change. The modeling system therefore needs to visualize the ownership of all changes in some way. This allows for a targeted discussion of specific aspects via the aforementioned communications channels.

We solve these challenges using the Delta Language (DL) approach, which is capable to sharing small and extensive model differences, combined with a strict separation of abstract, diagram and concrete syntax as described in section IV.

III. Methodology and Tool

Instead of relying on a shared screen, where one person models while everybody else makes suggestions, we have developed a true collaborative modeling tool for Mind Maps called *CMCM* (short for Collaborative Mind- and Concept-Mapping) [12]. It allows for simultaneous editing of the Mind Map by different users, each with their own graphical client. The clients are connected through a common server (our CoMo-X Server, see section IV).

Using a server hosted by the University of Oldenburg and a custom tool, we remain independent from international data servers. This is an advantage when using collaborative modeling for teaching, examination, or for international research projects, as it allows us to comply with the requirements of the General Data Protection Regulation (GDPR).

We will now describe a modeling session, where we created a Mind Map for collaborative Mind Mapping using the *CMCM* tool. During that session, we communicate via the video-conferencing tool *BigBlueButton* [13], which is approved by our respective universities.

A. Installation

You can download *CMCM* from https://uol.de/se?cmcm. You will also find the installation instructions and software dependencies there.

B. CMCM Tool

Using a modeling scenario, we will introduce the functionality and the user interface of the *CMCM* tool.

Upon startup, *CMCM* shows a generic sample Mind Map. But before we can start collaborating, the collaborators have

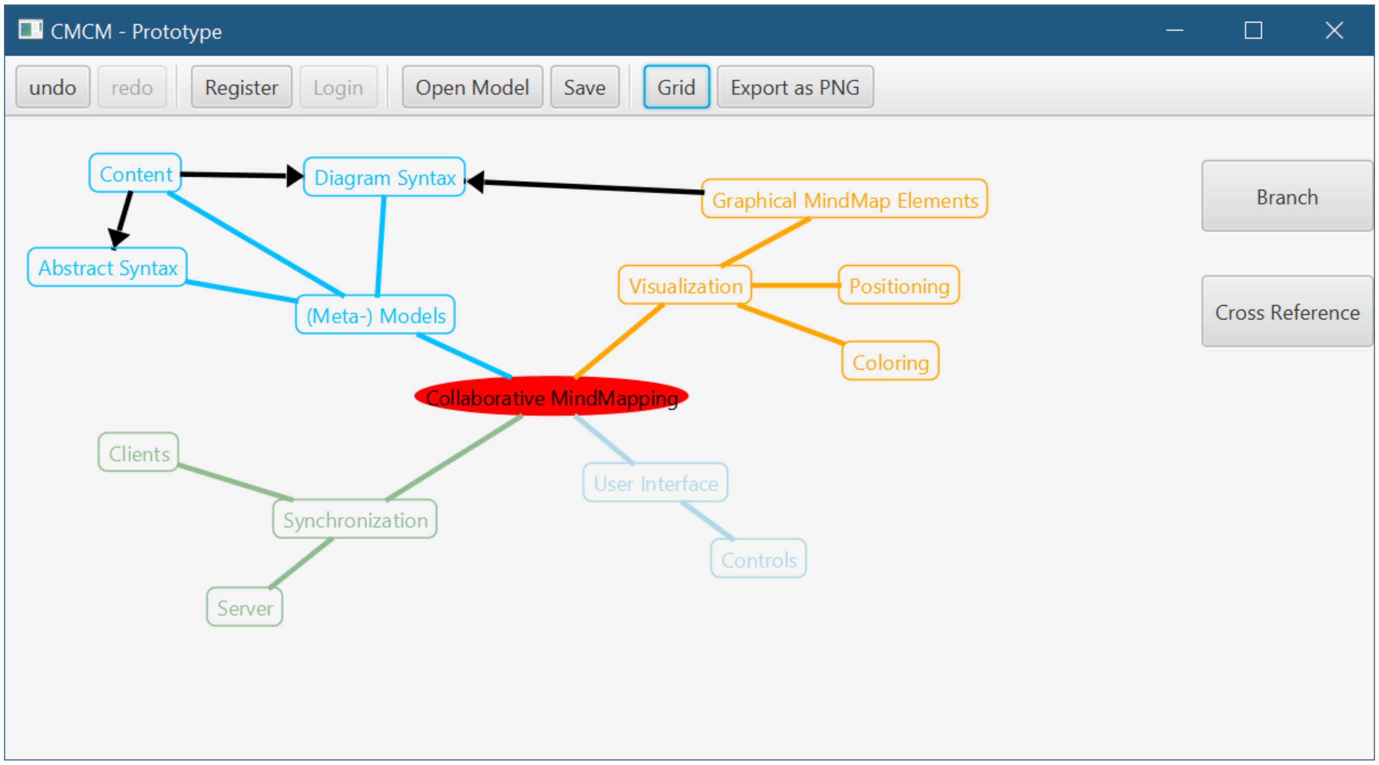

Fig. 1. Running example: Mind Map about collaborative Mind Mapping.

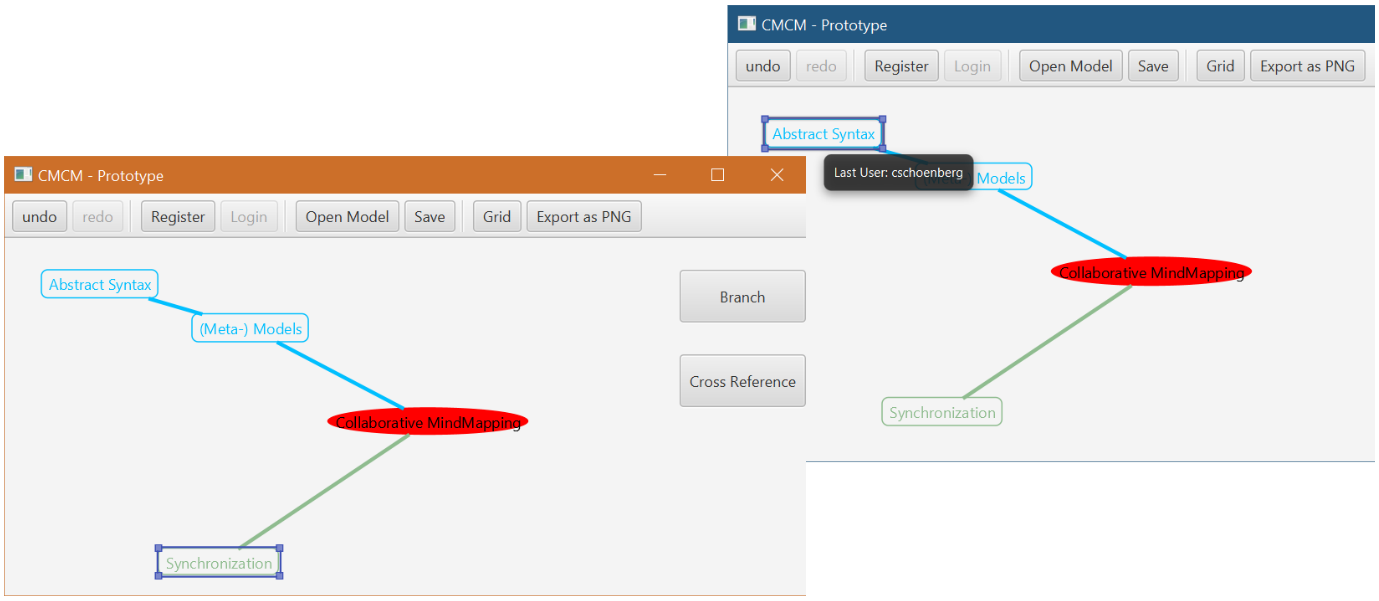

Fig. 2. *CMCM*: Collaborative modeling.

to identify themselves to the server: Each of us clicks on "Register" to create a new user identity, or on "Login" if we already have one. We are then presented with a list of Mind Map models that are available on the server. We can either select one of these existing Mind Maps, or create a new model.

One of us (Andreas) creates a new model, giving it the name "Collaborative Mind Mapping". All of us then select this model from the list (refreshing the list as necessary) and click "Open Model". We are now presented with identical views of the same Mind Map, which currently only consists of the root concept that is initially named "Root".

The final version of this Mind Map is shown in figure 1.

Andreas starts the brainstorming session by right-clicking on the root concept, selecting "Change Title" from the context menu, and changing the concept to "Collaborative MindMapping". Florian similarly changes the concept's color to "red" from the same context menu.

Christian creates a new branch by clicking on "Branch", then clicking on the concept from which to branch the new sub-concept off. He calls the new sub-concept "(Meta-) Models" and arranges to the top-left of the root concept. At the same time in Uzbekistan, Dilshod creates a new branch "Synchronization", which he places to the bottom-left of the root concept. To keep the concepts better apart, they each choose a different color for their branch.

Figure 2 shows two of the clients at this point. The right-hand client (Christian's) also shows who added the "Abstract Syntax" sub-concept as a mouse-over hint. Christian then adds another sub-concept "Content" to the "(Meta-) Models" branch. Florian takes this opportunity to add a cross-reference between "Content" and "Abstract Syntax": he clicks on the "Cross Reference" button, then on the "Content" concept as the source of the reference, and finally on the "Abstract Syntax" concept as the target of the reference. To separate it clearly from the sub-concept branches, he selects the color "black" for this cross-reference. Labels for references and bi-directional references are also possible.

At a later time, when the other collaborators have finished their (synchronous) modeling session, Kevin adds a new branch "User Interface". This asynchronous collaboration is seen by the other modelers when they open the Mind Map the next time.

Other controls allow collaborators to undo or redo specific actions (the "undo" and "redo" buttons, respectively), to save the current Mind Map as an image file ("Export as PNG"), to open another model ("Open Model"), to turn the grid of dots on and off ("Grid"), and to save specific versions of the model ("Save").

The latter function is useful for storing interim versions of the model that we can return to at a later date. This function is not yet implemented on the server side, however. We refer to these complete models that are stored as *macro versions*, in contrast to the *micro versions* that are swapped between clients and only contain the latest local changes. Whenever a new client logs in and opens a model, they receive the complete model as the optimized sum of micro versions (minus superfluous changes like things that were overwritten or deleted later).

Macro versioning also allows us to use our *CMCM* tool for asynchronous modeling, as shown in the scenario above: A collaborator can make changes to the Mind Map, while the other users are offline. These changes will be shown in their clients when they reconnect.

C. Performance and Efficiency

The micro versions (modeling deltas) in synchronous collaborative designing are represented by the operations of Difference Language (DL) [14]. This enables quick synchronization of changes between the collaborators in real-time. The model changes represented by small, textual operations provide high performance in change synchronization in synchronous collaboration. The high performance by DL-based change representations avoids conflicts in real-time.

According to the general language design, DL serves as a common change representation and exchange format for various services of asynchronous and synchronous collaborative development. The synchronous and asynchronous collaborative designing use cases take advantage of the same common change representation approach enabling a single point of truth. The same underlying change representation is also utilized to store the differences between several sequential revisions of a model in asynchronous collaborative development.

IV. Solution

Each instance of *CMCM* maintains its own model to represent a Mind Map. The models conform to a self-developed meta-model that consists of a Mind Map Modeling Language (M3L) and a Mind Map Modeling Language Diagram Interchange (M3L DI). Changes to a model are encoded using a DL (Difference Language) generated for M3L/M3L DI according to [14]. In encoded form, the changes are synchronized with the CoMo-X (Collaborative Modelling – eXtend) Server [15]. The CoMo-X Server serves as a single point of truth for the model. It distributes the changes as micro versions to all other instances of *CMCM* of the collaborators in a session. The instances of *CMCM* apply these changes to their respective local copy of the model. Accordingly, the models and thus the Mind Maps are synchronized between the collaborators of a session.

A. Modeling Language

Before a collaborative modeling session can start, the collaborators must agree on the modeling language. A modeling language specifies the modeling concepts and their possible combinations as well as the representation of these. Accordingly, collaborators can interpret and modify a representation of a model according to the chosen modeling language. Additionally, modeling software must present models to collaborators so that they can understand and, if necessary, modify the models. In *CMCM* the designed modeling language (cf. figure 3) for the description of Mind Maps is implemented.

a) Abstract syntax of Mind Maps: The three modeling elements root, branch and cross reference are needed for modeling Mind Maps. A root is used to describe a main concept that defines the focus of the Mind Map. Starting from the root, branches can be used to describe concepts that are related to the main concept. Branches can in turn be supplemented by further branches to associate the corresponding concepts with further concepts. Together, the root and branches of Mind Maps form a tree-like structure. Cross references can be used to relate two different branches and thus their concepts. In order to specify the relationship between two concepts more precisely, a cross reference is provided with a concept and a direction. Roots,

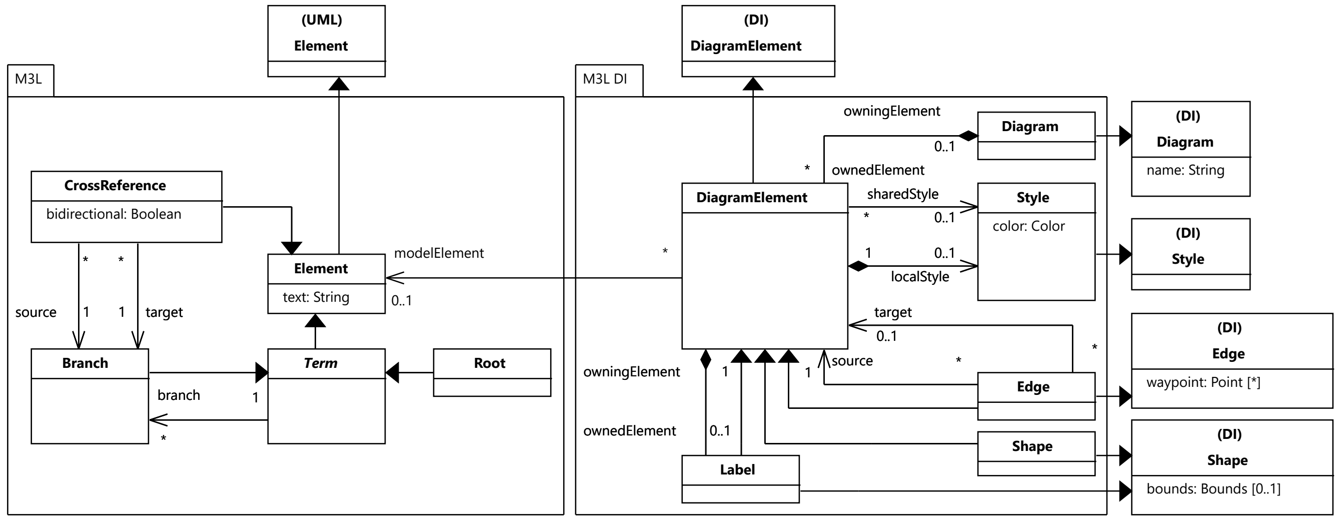

Fig. 3. Abstract syntax (package M3L) and diagram syntax (package M3L DI) embedded into UML and DD.

branches, cross references and their combination describe the connection of several concepts in a structured way. These modeling elements form the abstract syntax (cf. package M3L in figure 3) of Mind Maps.

b) Concrete syntax of Mind Maps: The root of a Mind Map is represented by an ellipse. Inside the ellipse, the main concept described by the root is rendered as text. Branches are rendered as rounded rectangles, connected by lines to the root or to another branch. The concepts of the branches are rendered as text inside the rectangle. Cross references are rendered as lines that are placed between two branches. Their direction is indicated with an arrowhead. Along the representing line of a cross reference, the (optional) concept of that cross reference is rendered as text. The placement of individual elements in a representation of a Mind Map is not predetermined and can be freely chosen by the collaborators. It is possible to move or enlarge the ellipse and texts. Ellipses, lines, arrowheads, texts and their placement form the contrete syntax (cf. figure 4) of Mind Maps.

In MOF-based modeling languages [16], such as UML [17], the modeling concepts and their combination, called abstract syntax, are described by MOF conforming models, called meta-models. The Diagram Definition (DD) standard [18] of the OMG offers a MOF compatible framework which integrates representation aspects into meta-model. This makes it possible to describe the representation of a model and to exchange it in the same way as models. The DD standard differentiates between layout information, called diagram syntax, and the representation of individual elements, called concrete syntax. When describing a MOF-based modeling language according to the DD standard, the diagram syntax is associated with the abstract syntax of the modeling language. By mapping the diagram syntax associated with the abstract syntax to graphical elements, the concrete syntax is described. The framework given by the DD standard uses three packages: Diagram Common (DC), Diagram Interchange (DI) and Diagram Graphics (DG). The DC package includes elements shared by the DI and DG packages. For example, the DC package includes the description of points with an X and Y coordinate. In the DI package, elements describing a diagram syntax are included and associated with each other. A modeling language can refer to the elements in the DI package and specialize them according to the required diagram syntax. The DG package describes graphical elements such as ellipses that can be referenced by a mapping to define the concrete syntax of a modeling language. Due to the possibility to exchange representations of models like models and the separation between abstract syntax, diagram syntax and concrete syntax, the modeling language of *CMCM* is described according to the DD standard.

On the left side in figure 3 the package M3L is used to describe the abstract syntax of Mind Maps. The classes Root, Branch and CrossReference are representing the equivalent modeling elements of Mind Maps. Each modeling element of a Mind Map can be provided with a text. From a root as well as from a branch there can be any number of branches. Branches can be connected with cross references. A cross reference can be unidirectional or bidirectional.

On the right side in figure 3 the package M3L DI is used to describe the diagram syntax of Mind Maps. A Mind Map is composed of shapes and edges. Shapes are diagram elements whose position and size can be adjusted. Thus, the position and size of a root can be described by a shape. Edges are diagram elements that form a line and are positioned by specifying multiple points. Edges always have a diagram element as a source and can connect two diagram elements with the specification of a target. Thus, edges are suitable for positioning branches and cross references. Shapes and edges can be labeled to display text. The position and size of the labels can also be adjusted, as with shapes. All diagram

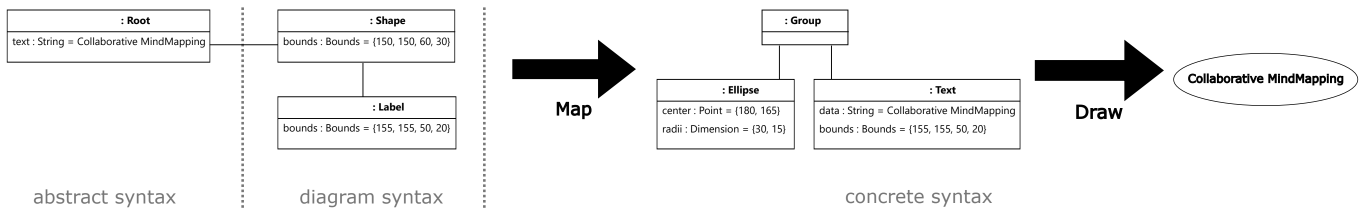
: Root
text : String = Collaborative MindMapping
: Shape
bounds : Bounds = {150, 150, 60, 30}
: Label
bounds : Bounds = {155, 155, 50, 20}
Map
: Group
: Ellipse
center : Point = {180, 165}
radii : Dimension = {30, 15}
: Text
data : String = Collaborative MindMapping
bounds : Bounds = {155, 155, 50, 20}
Draw
Collaborative MindMapping
abstract syntax
diagram syntax
concrete syntax

Fig. 4. Abstract syntax, diagram syntax, and concrete syntax for the root concept in the use case (without color).

elements are color-able by a local and / or shared style. If a diagram element does not have a local style, then it uses the shared style of itself or its owner.

Figure 4 shows the interrelations between abstract syntax, diagram syntax and concrete syntax, using the root element as an example. The concept “Collaborative MindMapping” from the model in figure 1 is of type `Root` (abstract syntax). It has a `Shape` and a `Label` (diagram syntax) with given coordinates, but the concrete form of that shape is not yet specified. This is achieved by mapping the concept to the `Ellipse` and `Text` types (concrete syntax), respectively. Ellipse and text specify the exact form and shape of the graphical element, and combined with the layout data the element can now be drawn. For other elements of a Mind Map, a simultaneous approach is used. From the combination of an element of the diagram syntax and an element of the abstract syntax, whose connection is given by an association, the concrete syntax of this combination can be uniquely derived. A mapping realizes the concrete syntax by instantiating elements of the package *DG* of *DD*.

In *CMCM*, changes to the model of the abstract as well as the diagram syntax are exchanged. The concrete syntax of Mind Maps is fixed and therefore does not need to be exchanged. It is ensured that all collaborators who are familiar with Mind Maps can interpret the Mind Maps in *CMCM*. In addition, the volume of data that has to be exchanged in case of model changes is reduced. It is only necessary to exchange changes to models describing the abstract syntax and the diagram syntax.

B. Difference Language (DL)

Difference Language (DL) is a family of operation-based domain-specific languages dedicated to represent model changes in the form of modeling deltas. Specific DLs for representing modeling deltas within a particular modeling domain are derived from the meta-models of these modeling languages. In the application in this paper, a specific DL is generated from the abstract and diagram syntax as described in figure 3 for describing changes in the Mind Map modeling language. The DL generator is language-agnostic, i.e., it does not rely on a particular modeling language. The DL-based collaborative modeling approach can be applied to a wide range of modeling languages by generating specific DLs.

Figure 3 (package `M3L`) depicts the content part (i.e. abstract syntax) of the Mind Map modeling language. In graphical modeling, each modeling object has design information such as ratio, size, and position, also called layout information. Figure 3 (package `M3L DI`) portrays the layout part (i.e. diagram syntax) for the content part, which is changeable by collaborators. The layout part of the meta-model depicts a subset of the Graphical Modeling Framework (GMF) notation [19], which supports notations for developing visual modeling editors. The meta-model in figure 3 includes all content (`M3L`) and layout (`M3L DL`) aspects that can be manipulated by collaborators. Thus, model differences occurring during collaboration refer only to the data specified in figure 3. The specific DL for the *CMCM* meta-model is generated by the DL generator service explained in [14]. While generating the specific DLs, the DL generator inspects all meta-classes. For each of these meta-classes, it iterates over the attributes and collects those that are changeable and not marked as persistent identifiers. For each meta-class, the DL generator generates creation and deletion operations. For each attribute of these meta-classes, the DL generator generates a change operation. In general terms, the DL generator applies three basic operations: *create* and *delete* for each meta-class and *change* for each attribute of a given meta-model. Relationships are treated as attributes of meta-classes.

The concrete DL used in *CMCM* is derived directly from the *CMCM* meta-model (cf. figure 3), covering content, graphical representation and the corresponding mapping. Each concept (classes, attributes, etc.) from the meta-model is associated with the appropriate operations (e.g., add, change, or delete) [14].

Creating a new branch “Visualization”, moving it to the top-right of the root, and then changes the color to “green” can be represented as a set of operators like the following:

```
<r-id>.createBranch(Title ‘‘Visualization’’)
<b-id>.change(X ‘‘395.3333435058594’’,
              Y ‘‘60.333343505859375’’)
<b-id>.change(Color ‘‘0x8fbc8fff’’)
```

The first line creates a new branch under the root. The root is identified by its ID, shown here simply as `<r-id>`. The next two lines change attributes of the new branch (represented by its ID shown as `<b-id>`): both the x- and y-coordinates are changed when the new branch is moved to a new location. In the last line, the color attribute of the branch is changed to a value representing “green”.

The result is shown in figure 1 in the top-right quarter.

C. Micro and Macro Versions

During synchronous collaboration, model changes are detected by a listener service in the modeling client and are described by micro versions using the DL operator syntax. In asynchronous collaboration, model changes are calculated by model differencing tools [14] and stored as macro versions by DL operations.

In case of synchronous collaboration, modeling deltas consist of a small set of model changes, whereas they represent a larger set of model differences in asynchronous collaboration. Modeling deltas consisting of small changes in synchronous collaboration are referred to as *micro-versions*, whereas they are referred to as *macro-versions* in asynchronous collaborative designing. The *CMCM* tool enables synchronous collaboration use case by micro-versioning, whereas asynchronous collaboration is built on macro-versioning. Since micro and macro versions are exchanged through the same DL notation, model differences in synchronous and asynchronous collaboration can be handled using the same techniques.

D. Main Components

The *CMCM* tool uses several independent services and orchestrations of these services. Generally, the tool is built based on a repository-style client server architecture pattern. Specifically, the server consists of a synchronizer service and a modeling repository. For synchronous collaboration, the client side of the tool is made of a change listener, a change applier and an editor service. Asynchronous collaboration requires a difference calculator service and a synchronizer [14, p. 125ff].

- **Modeling Editor.** A modeling editor is the main feature of the *CMCM* tool. It is used to design system models by end users. The editors open models from the server providing an option to select which model users want to develop.
- **Change Listener.** A change listener service is a part of each client instance of the tool. It listens for changes users make in the modeling editor of their tool instances. The listener detects every single change made by users on their editors. The detected changes are constantly sent to the server in order to synchronize them with other parallel instances of the model under development. The model changes are contained in deltas and synchronized using the operations of the Difference Language.
- **Calculator.** Comparison between model versions is done on the server. In asynchronous collaboration a new model variant is compared to the current version. Model differences are manifested by the Difference Language and synchronized to the resulting merged model version.
- **Applier.** Like the change listener service, a change applier service is also a part of each client instance of the tool. While the model changes are detected by the change listener service and synchronized among the several model instances by a synchronizer server, these changes are applied to other model instances by this change applier service once they are delivered to other clients.
- **Synchronizer.** A synchronizer service is located at the server. It synchronizes changes in modeling deltas among all collaborators of a particular model under development.
- **Model Repository.** A model repository is located on the server side. It serves to store models and their histories. The model repository consists of only modeling deltas namely *active deltas* and *difference deltas* [14]. The active deltas describe the active revisions of model under development, whereas difference deltas describe changes between subsequent revisions of system models.

V. Hands-on Session

During the practical session, we plan to model a Mind Map about collaborative modeling. All participants should have at least a passing familiarity with the subject, so we anticipate no content-related difficulties. Anyone who installs the *CMCM* tool can therefore participate. For installation instructions, see section III-A.

As a result, we expect a well thought-out Mind Map about the topics of the workshop, re-capturing the information learned by all participants. We also expect an in-depth test of our approach and our tooling. Identifying bugs and missing features is the first part of this test, followed by identifying features that work well but could be improved upon. This includes extending our experiences concerning modeling conflicts: are they rare enough to be irrelevant for practical applications, or do they occur occasionally for larger groups of participants?

Parallel to the technical evaluation, we are also interested in best practices for collaborative modeling:

- What is the best *mode of communication*?
  Possible solutions include audio only, audio and video, text chat, and breakout rooms. Depending on the number of workshop participants, different communication modes can be used.
- What is the best way to deal with *alternative modeling ideas*, where one person wants to model something in a different way than another?
  Possible solutions include discussion, majority vote, and split/fork of the model.
- How can we deal with headstrong people who do not adhere to the group's decisions? How do we deal with actual *sabotage*?
  Possible solutions include banning, restricting rights, and social influence.

The outcome of the hands-on session will be an initial Mind Map of the workshop content, including some ideas and strategies for improving the *CMCM* tool and the associated collaborative modeling process.

VI. Conclusion

This paper presented the *CMCM* tool for collaborative modeling of Mind Maps. The tool allows for synchronous and asynchronous modeling. It uses model differences, encoded via the operators of Delta Language, to transmit model changes from a client to the server, which in turn sends the changes to all other clients. The server acts as the single point of truth.

Micro versions, resulting during synchronous collaborations, are sufficiently small to be shared simultaneously, without interfering with the common work.

While theoretically possible, no conflicting change information has been observed on the server during our extensive tests. Tools following the DL-principle have been used experimentally by more than ten users located over long distances (Africa, Asia and Europe), all connecting to the same server located in Germany. During these experiments, the tool has shown sufficiently high performance by synchronization of small DL-based modeling deltas [7].

Larger sets of delta language operators (macro versions), contain the entire model, which e.g. is sent to each newly connected client. They also support model exchange during asynchronous collaborations and serve as interim safe states that can be restored at a later date. In theory, any state of the model can be restored at any time by backtracking the micro versions, but in practice macro versions are more convenient because they can be applied more efficiently when explicitly marking a tagged branch.

In *CMCM*, any Mind Map can currently be edited by any registered user simultaneously. We plan to add a more comprehensive access-management, including asynchronous collaboration support. For asynchronous collaboration, a merging-mechanism for models is also required. In case of accidental deletes or erroneous edits, the entire diagram history is stored on the server, so any change made to a diagram by any user can be reversed later. The general DL-based approach can be integrated into open source tools, as shown for the no longer supported UML Designer [6]. Further activities will migrate these ideas to further UML-tools.

Currently, *CMCM* allows a – conceptually limitless – group of collaborators to develop and discuss Mind Maps. It can be deployed on premise, thus satisfying regulatory demands of the GDPR, if used with a trusted server. Users can model all relevant aspects of Mind Maps, including branches and colors. They can undo and redo changes, and track the authors of each Mind Map branch. The strict separation between abstract, diagram and concrete syntax allows for efficient synchronization of changes and provides easy adaptability of our approach to other modeling languages.

References

[1] C. A. Ellis, S. J. Gibbs, and G. Rein, "Groupware: some issues and experiences," *Communications of the ACM*, vol. 34, no. 1, pp. 39–58, 1991.

[2] G. Booch and A. W. Brown, "Collaborative development environments," *Adv. Comput.*, vol. 59, no. 1, pp. 1–27, 2003.

[3] W. Zhou, E. Simpson, and D. P. Domizi, "Google docs in an out-of-class collaborative writing activity." *International Journal of Teaching and Learning in Higher Education*, vol. 24, no. 3, pp. 359–375, 2012.

[4] R. A. Calvo, S. T. O'Rourke, J. Jones, K. Yacef, and P. Reimann, "Collaborative writing support tools on the cloud," *IEEE Transactions on Learning Technologies*, vol. 4, no. 1, pp. 88–97, 2010.

[5] "Overleaf: Collaborative document writing," http://overleaf.com, last accessed 2021.

[6] M. Appeldorn, D. Kuryazov, and A. Winter, "Delta-driven collaborative modeling." in *MODELS Workshops*, 2018, pp. 293–302.

[7] D. Kuryazov, A. Winter, and R. Reussner, "Collaborative modeling enabled by version control," *Modellierung 2018*, 2018.

[8] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato, *Version control with subversion.* " O'Reilly Media, Inc.", 2004.

[9] T. Swicegood, *Pragmatic version control using Git.* Pragmatic Bookshelf, 2008.

[10] T. Buzan, *Mind Map Mastery: The Complete Guide to Learning and Using the Most Powerful Thinking Tool in the Universe.* Watkins Publishing, 2018.

[11] S. Edwards and N. Cooper, "Mind mapping as a teaching resource," *The clinical teacher*, vol. 7, no. 4, pp. 236–239, 2010.

[12] K. Meyer, "Kollaboratives Mind- und Concept-Mapping," Bachelor's Thesis, University of Oldenburg, Germany, 2021.

[13] BigBlueButton Inc., "BigBlueButton," https://bigbluebutton.org/, 2021.

[14] D. Kuryazov, "Model difference representation," Ph.D. dissertation, University of Oldenburg, 2019. [Online]. Available: http://oops.uni-oldenburg.de/3938/1/kurmod19.pdf

[15] T. Sprock, "Repository Managementsystem für Delta Language," Master's thesis, University of Oldenburg, Germany, 2020.

[16] Object Management Group, "Meta Object Facility (MOF) 2.5.1 Core Specification," https://www.omg.org/spec/MOF/2.5.1, 2019.

[17] ——, "OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1," https://www.omg.org/spec/UML/2.5.1, 2017.

[18] ——, "Diagram Definition (DD), Version 1.1," https://www.omg.org/spec/DD/1.1, 2015.

[19] Eclipse Foundation, "Graphical Modeling Project (GMP)," https://www.eclipse.org/modeling/gmp/, 2018.