Unifying Model- and Screen Sharing

Yentl Van Tendeloo
University of Antwerp, Belgium
Yentl.VanTendeloo@uantwerpen.be

Hans Vangheluwe
University of Antwerp, Belgium
Flanders Make vzw, Belgium
McGill University, Canada
Hans.Vangheluwe@uantwerpen.be

***Abstract*—The complexity of engineered systems is ever increasing, resulting in a plethora of larger and more diverse models. This increase in complexity can be addressed by collaborative model development, also known as Concurrent Engineering. We distinguish two distinct types of collaboration, based on the different collaboration needs between modellers: screenshare and modelshare. Screenshare allows users to collaborate –often at the same time– using exactly the same visualization. This implies that even the most trivial model modifications, even semantics-preserving ones, are replicated for all users. Modelshare allows users to share the same model, albeit with different visualizations, offering different views on the model, as standardized in ISO/IEC/IEEE 42010:2011. Both types of collaboration are currently in use. We present a unifying framework integrating both approaches. This unifying framework is similar to our existing framework for concrete syntax, reusing existing modelling tool infrastructure. This allows for different types of collaboration to be intertwined: screensharing with some users, while modelsharing with other users.**

I. Introduction

The complexity of engineered systems is ever increasing, resulting in a plethora of larger and more diverse models. This increase in complexity can be addressed by collaborative model development, also known as Concurrent Engineering.

We distinguish multiple types of sharing, at different levels of access to model elements. In particular, we focus on screenshare and modelshare, terms used in AToMPM [1], a collaborative, web-based (meta-)modelling environment. With screenshare, different modellers share the exact same representation of a model, offering the experience of working on a shared modelling canvas. With modelshare, different modellers share the same model, but have different views (with different visualizations) on this model. Whereas both types of collaboration have their advantages and disadvantages, they require significantly different implementations.

These two types of collaboration are mostly orthogonal to collaboration algorithms focussing on consistency/conflict resolution, such as (partial) model locking [2], token-based control [3], and model versioning [4], [5]. Implementations of these algorithms must take into account the different types of sharing (i.e., modelshare or screenshare). For example, in screensharing it is important that the visual position of model elements is locked while a user drags it. This is not required with modelsharing. Groupware, which collaboration is a part of, is gaining in importance [6], and is since long considered as difficult to create [7]. While several advancements have been made, implementing a collaborative environment still requires much thought about corner cases [8].

We present a unifying framework for the above two types of sharing. This allows for a unified implementation, which can be combined with all other aspects of collaboration, such as conflict resolution algorithms. Additionally, both types of sharing can be used within the same tool, but also allows for combinations, where some users perform screensharing with other users using modelsharing, all on the same model.

The unifying framework is based on our existing framework for dealing with the concrete syntax [9] of models in a Multi-Paradigm Modelling (MPM) [10], [11] environment. This existing framework has additional advantages, some of which are also relevant in the context of collaboration, as we will present later. It is possible that this framework is already supported in a tool, in which case our approach to model collaboration comes at minimal cost.

We have implemented this approach in our Multi-Paradigm Modelling environment, named the Modelverse [12]. Throughout this paper, we present the running example of a Causal Block Diagrams (CBD) (also known as Synchronous Data Flow) model. The CBD language is a simple yet realistic language, used to model complex mathematical equations. A well known CBD language is Simulink®. Models in a CBD language consist of blocks with inputs and outputs. Connections between these blocks carry a signal, which the blocks manipulate. The types of blocks include simple algebraic operator blocks, such as addition blocks, but also more advanced blocks, such as integration blocks, which operate on entire signals (functions of time).

The remainder of this paper is organized as follows. Section II presents the necessary background on model syntax and our Multi-Paradigm Modelling approach to it, which is used extensively throughout this paper. Section III and Section IV elaborate on screensharing and modelsharing, respectively, and presents how this is facilitated by our framework. Section V presents the unification of both screensharing and modelsharing and discusses the advantages. Section VI explores related work and Section VII concludes the paper.

II. Background

This paper heavily relies on language engineering concepts. To ensure that this terminology is clear, be briefly recap these terms here. Subsequently, we briefly present our concrete syntax framework, forming the foundation of our approach.

A. Modelling Syntax

Modelling *languages* in general are defined by their abstract and concrete syntax [13], [14]. The abstract syntax defines the concepts of a language and how they may be combined. These concepts can be instantiated and used as the building blocks of models (the sentences in the language). For example, the abstract syntax of CBDs defines concepts such as *Addition*, *Constant*, and *Signal*. The concrete syntax defines the visualization, or rendering, of these abstract syntax concepts. For example, the concrete syntax of CBDs defines the mapping of a *Constant* to a circle with the name of the constant its value.

A modelling *formalism* additionally has a notion of semantics, describing the meaning of sentences in the language. Semantics is given by providing both a semantic domain and a semantic mapping function. The semantic domain is language, for which semantics is assumed known, to which all sentences in the original language are mapped. For example, the semantic domain of CBDs can be, for each signal in the CBD, a Real function of the Real time base, representing the values of the signals over time. The semantic mapping specifies how elements of the language are mapped to instances in the semantic domain. For example, a semantic mapping of CBDs can be given in the form of an operational simulation specification.

In the remainder of this paper, we will mostly concern ourselves with the language aspect: abstract and concrete syntax. Indeed, we only consider collaboration in modelling, which might even occur on models in languages that have no behavioural semantics, such as UML Class Diagrams.

B. Multi-Paradigm Modelling Approach to Concrete Syntax

Previous work [9] introduced an MPM approach to concrete syntax, addressing the limitations of existing modelling language engineering approaches to concrete syntax. Other approaches had limited portability (e.g., only a single front-end exists), limited perceptualization (e.g., only visual representation), limited mappings (e.g., only a single type of visualization), limited lay-outing possibilities (e.g., only general-purpose lay-out), and limited definitions (e.g., only a single "icon" for an abstract syntax concept). These are especially limiting in the context of Multi-Paradigm Modelling, which proposes to model all relevant aspects of a system at the most appropriate level(s) of abstraction, using the most appropriate formalism(s), while explicitly modelling the process.

To circumvent these restrictions, we defined an MPM framework for concrete syntax in which all aspects of concrete syntax are modelled explicitly. The concrete syntax language is modelled explicitly, defining concepts such as a rectangle and a circle. Perceptualization of models can then also be modelled explicitly through the use of activities, mapping from the language directly to the concrete syntax language. The concrete syntax language is then used as a description of the data format exchanged with the front-end. The front-end is what the user interacts with. Instances of this concrete syntax language can then directly be exchanged with any type of conforming front-end. Essentially all perceptualization is handled in the back-end, instead of the front-end: the front-end never directly accesses the abstract syntax of the model. The front-end then maps the concepts of the data format (e.g., *Rectangle*) to equivalent operations on the platform (e.g., `create_rectangle(...)` in *TkInter*).

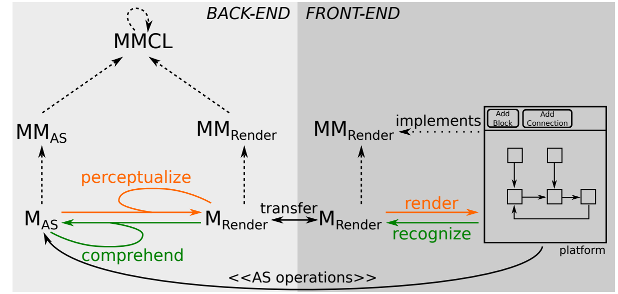

Fig. 1: Overview of the approach, taken from [9].

Figure 1 presents this framework. *MM*$_{Render}$ denotes the concrete syntax metamodel, present in the back-end (as an explicit model) and the front-end (possibly hardcoded and only implicitly accessible, through an API). Perceptualization happens entirely in the back-end, where an M_{Render} is constructed, describing how M_{AS} is presented. This M_{Render}, for example, defines where to put rectangles and circles, and is completely unrelated to the CBD domain. Any front-end that has knowledge of the same *MM*$_{Render}$ (e.g., with concepts such as *Rectangle* and *Circle*) can then visualize this model effortlessly. The visual *MM*$_{Render}$ described above is only an example, and could just as well be for sound or text concrete syntaxes.

III. Screen Sharing

The first type of collaboration considered is termed "screen sharing". As the name suggests, it seems to users as if they are sharing the same screen or canvas. That is, users share the same concrete syntax rendering of the model, including details of the visualization that are not present in the abstract syntax such as the visual location of elements. For example, with CBDs, users performing screen sharing see the exact same visualization, and even dragging visual elements is immediately reflected in all users' views.

This type of collaboration is useful if there is a close interaction between different modellers. An example is "pair modelling", where multiple users are simultaneously reasoning about the model and its construction. In this case, all types of changes are considered to be relevant as the placement of elements can already be indicative of modelling intentions. In essence, all concrete syntax changes are always propagated, meaning that models are shared at the concrete syntax level.

In our concrete syntax framework, this indicates that multiple users share the same M_{Render} model. As this is the perceptualized representation of the model, shared information includes concrete syntax concepts, such as concrete syntax element coordinates. Note that users do not directly share the abstract syntax, but instead share a perceptualization of it. While changes to the concrete syntax can still have an impact

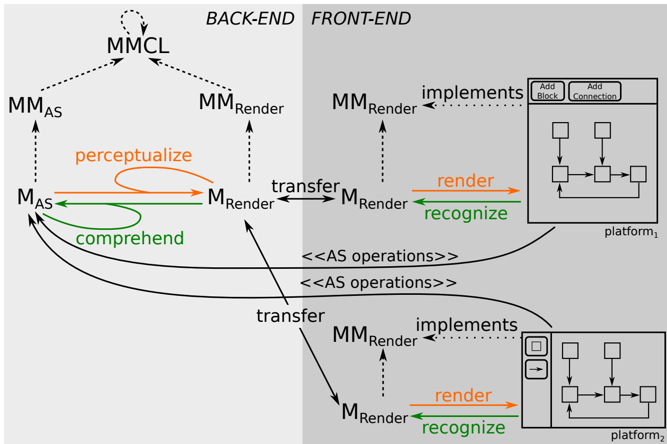

Fig. 2: Screenshare in our concrete syntax framework.

on abstract syntax, and vice versa, screensharing assumes that sufficient information is present in the concrete syntax.

This is shown in Figure 2, where two interfaces, one for each collaborating user, use the same M_{Render}. At the right, we see two front-ends, which might be different implementations on entirely different software and hardware platforms. The only restriction is that they share the same MM_{Render}, such that they are able to understand the same visualization, making screenshare possible. Both interfaces have their own copy of M_{Render}, which they synchronize through the back-end, immediately propagating changes. These three versions of M_{Render} are kept synchronized, and conflicts are resolved, through the use of existing techniques. Since both front-ends are merely visualizers of the M_{Render}, this guarantees that all front-ends present the exact same information. Any change on M_{Render} can have an impact on the abstract syntax model M_{AS} as well. Therefore, all changes are *comprehended* and potentially result in changes to the abstract syntax model, in turn causing a new *perceptualization*. For example, dragging an element inside of another element is a concrete syntax operation. It is however comprehended as an abstract syntax operation as well, where a containment link is created between both elements. Subsequently, perceptualization might require the container to be enlarged to completely contain the containee, thereby updating M_{Render} again. After the new M_{Render} is generated, it is propagated to all front-ends.

Our framework therefore natively handles screensharing, as this is collaboration on the model M_{Render}.

IV. Model Sharing

The second type of collaboration considered, is termed “model sharing”. As the name suggests, users share the same model, but not to the same extent as screen sharing. That is, users share the same model, but might have different visualizations of that model. This resembles multiple views on a single model. For example, with CBDs, users that perform model sharing operate on the same CBD model, but may have different visualizations (e.g., different model “icons”, different perceptualization formats, or just different element locations). Their visualizations are independent of one another. That is, even if they have the same visualization method, the visualized artefacts are different, meaning that concrete syntax operations such as dragging are not replicated, unless these operations also have an effect on abstract syntax.

This type of collaboration is useful if multiple users wish to collaborate, but have different backgrounds or preferences. For example, some users might prefer a model textual notation, while other users prefer a visual one. Neither is vastly superior to the other, and they both have their use [15]. Note that this approach can also be used outside of the context of collaboration, where a single user opens multiple editors on the same model. For example, a music composition model can be visualized using standard musical symbols, in a musical score, but can also be *sonified*, where the composition is played for the user. While this is not technically sharing, as only a single user is involved, there are multiple views on the same model.

This approach is also useful for performance improvements with many collaborating users. Indeed, screen sharing pushes many (small) changes in all directions: every drag operation is repeated by all connected clients. And while this can be beneficial to know what people are working on, it can easily turn into a performance bottleneck. Cognitively, users might become distracted by the changes made by all other users. Sharing only the abstract syntax changes reduces the amount of exchanged data and additionally limits the changes that happen concurrently on the same screen.

In our concrete syntax framework, multiple users share the same M_{AS} model, although they have different M_{Render} models, possibly even conforming to different metamodels MM_{Render}. As this is the unperceptualized model, only the abstract syntax model is shared. Changes still happen through the concrete syntax, but now there is no collaboration at that level anymore: the M_{Render} is specific to the user. Only when changes are comprehended to changes on the M_{AS}, actual collaboration occurs. Collaboration therefore happens on the M_{AS} model, although indirectly: users do not directly modify this model. This is similar to views, where users modify elements in their view, instead of directly modifying the underlying abstract syntax.

As was the case with screensharing, this framework is orthogonal to conflict resolution, relying on existing algorithms. Modelsharing implies collaboration at the level of the M_{AS}, where conflicts must be solved at that level. In contrast to screensharing, edit operations on M_{AS} come from a model transformation, although this has no influence.

This is shown in Figure 3, where two interfaces, one for each collaborating user, use the same M_{AS}. At the right side, we see the two different front-ends, which in this case even have a different MM_{Render}. In our example, they visualize the model in different ways: one uses a symbol-based visualization of the CBD trace, while the other uses a plot-based visualization of this exact same trace.

As was the case with screensharing, all changes on M_{Render} can potentially result in changes to M_{AS} as well. When M_{AS} is modified, these changes must now not only be perceptualized to the M_{Render} that caused the change, but should be

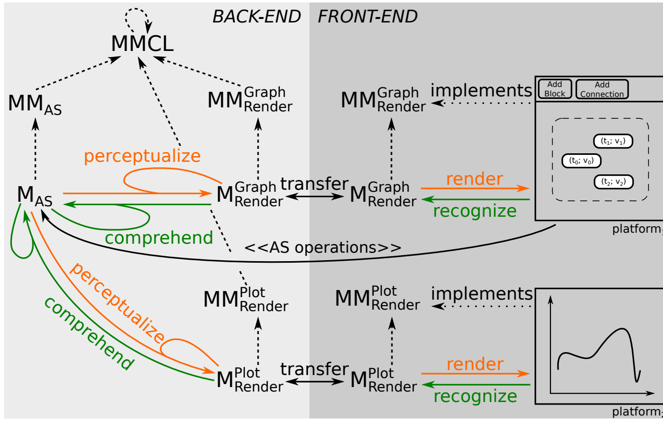

Fig. 3: Modelshare in our concrete syntax framework.

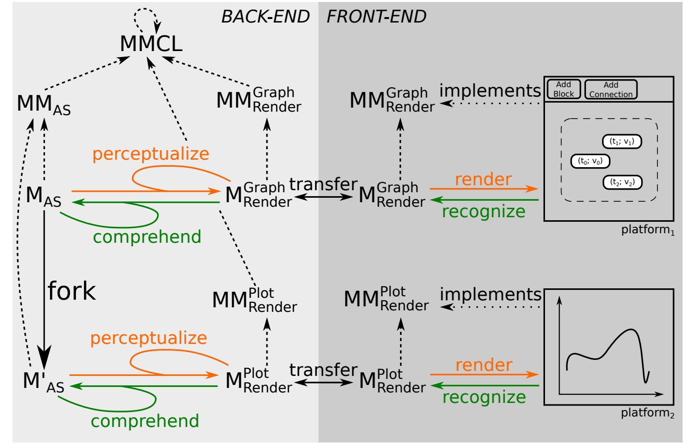

Fig. 4: Overview of no sharing of the same model, requiring a fork of the abstract syntax model.

pushed to all perceptualized models. As such, changes to one M_{Render} can result in changes to other M_{Render} models. Since the changes are coordinated by the M_{AS} and perceptualized with specific activities, all of this can happen transparently to the users. To the users of the other M_{Render}, the change will simply be as if someone directly altered their concrete syntax.

Our framework therefore natively handles modelsharing, as this is collaboration on the model M_{AS}. Note that screensharing automatically implies that modelsharing is also being done, as the same changes on concrete syntax will result in the same changes in abstract syntax.

V. Unification

We have indicated how both screensharing and modelsharing can be implemented through the use of our explicitly modelled framework for concrete syntax. Both approaches have already been implemented in tools such as AToMPM [1]. Our approach has additional benefits, however, which are thanks to its unifying nature. As our approach mostly reuses the existing concrete syntax framework for its operation, an implementation doesn't have to start from scratch.

A. No Sharing

Up to now, we have not yet mentioned how modellers would go about *not* collaborating. Indeed, while we unify two approaches to collaboration, our approach is also applicable when there is no sharing at all. Instead of sharing at the concrete syntax or the abstract syntax level, users then first create a copy of M_{AS}, on which they then operate. As they do not collaborate on the same model, this causes a divergence, which could later on be resolved with techniques such as model versioning [4] or model merging [16]. Both techniques also allow one to deal efficiently with the seemingly inefficient model copying. Since this is not related to collaboration in modelling, we do not elaborate on this topic.

An overview of our framework, this time without sharing, is shown in Figure 4. In this case, the abstract syntax model was duplicated, or "forked", to ensure that no changes are propagated between different users. Whatever changes are made by one user, nothing will change for the other users.

B. Combining Types of Sharing

With these three types of sharing unified, the next logical step is to allow combinations simultaneously. For example, some users might be model sharing, whereas other users are screen sharing, and some might be working on different versions of the abstract syntax model.

This is shown in Figure 5, where there are seven users that are (partially) collaborating. In this case, there are two versions of the abstract syntax model, between which there is no collaboration. There might be future collaboration due to potential model merges or through the use of model versioning. Users A, B, and C collaborate on the same model M_{AS}, and users D, E, F, and G collaborate on another model M'_{AS}, a forked version of M_{AS}. At the next level, there is sharing of the abstract syntax between these users within their group, meaning they perform model sharing with one another. In one branch, users A, B, and C perform model sharing, and in the other users D, E, F, and G perform model sharing as well. Additionally, some users have a stronger form of sharing between them: screensharing. Indeed, users A and B share the same concrete syntax as well, and are thus using screensharing. Similarly, users D, E, and F use screensharing as well.

As a result, there are different levels of sharing and change propagation, as shown in Figure 6. First, we note two types of groups: there are screenshare groups (green: w, x, y, and z) and modelshare groups (red: u and v). Note hat a screenshare implies a modelshare, as this includes all the same, and some more changes that are to be propagated. Following the overview in Figure 5, we see the various groups of collaboration. Using a CBD model as an example, we show exactly the same model for each client. Although the model is semantically (and in abstract syntax) exactly the same, the four screensharing collaborations each have different visualizations: group w uses the previously introduced notation; group x has a different location of the elements; group y uses a Σ instead of a $+$ symbol; and group z uses a grayscale notation.

If user A makes a modification α, where the constant block is dragged slightly to the right, this change is propagated to the

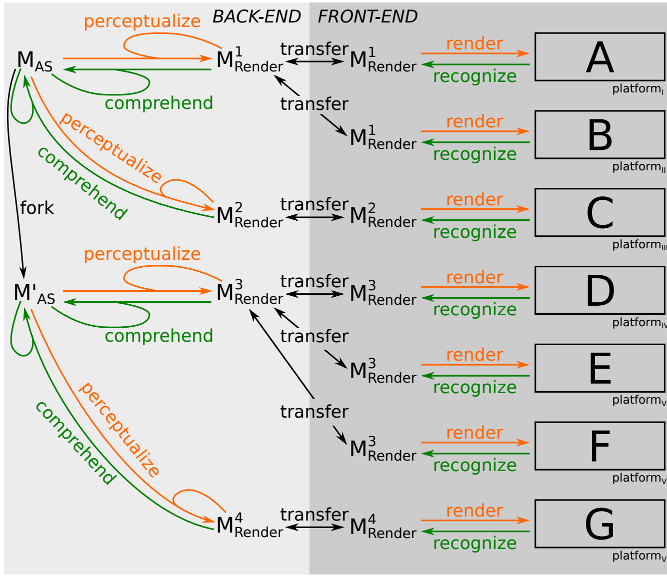

Fig. 5: Collaboration at different levels for seven users.

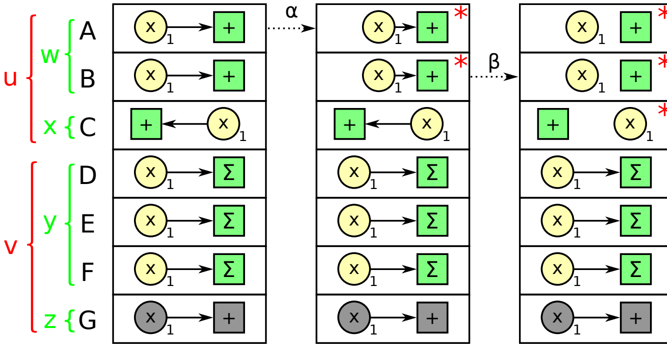

Fig. 6: Different degrees of change propagation, depending on the type of collaboration.

other users. Since the comprehension phase ignores changes to the location of blocks, the changes are limited to M_{Render}. As such, the change is only noticed by the users in the same screencast collaboration (group w), more specifically users A and B. This is then shown in the next column, where both users have a modified representation of the model, while all other users have their representation untouched.

Afterwards user B makes a modification β, deleting the link from the constant block to the addition block. This change has an effect on the concrete syntax model (i.e., the link is not drawn anymore), but is also comprehended as a change on the abstract syntax model M_{AS} (i.e., the association of type *Link* is removed). This change is propagated to all users in the same modelshare group (u), more specifically to users A, B, and C. While this was normal for user A (using screenshare) and user B (initiating the modification), also user C sees a modification. The modification at user C happens due to the removal at M_{AS}, which causes perceptualization to happen again, thereby altering the concrete syntax of user C. Users in modelshare group v have forked from M_{AS} to M'_{AS} and are therefore not notified of the changes to the abstract syntax.

Removals are often easier to process than additions. In the case of an addition, the element is added in abstract syntax, but there is no information on where to put the element. This information was there for the screenshare users, namely the location of the mouse click, but this information cannot be used with modelsharing. As such, perceptualization has to add information to come up with a location. This can depend on the specific modelling domain.

Another situation, not shown, is where user A performs a concrete syntax change, such as dragging an element inside another element. User A and B immediately see the change in concrete syntax (the dragging), but comprehension indicates that a new association be created between the container and containee. After the next phase of perceptualization, this abstract syntax change is also propagated to the concrete syntax of user C, meaning that in this case a concrete syntax change resulted in concrete syntax changes for users that are only model sharing. Again, users in a different modelshare group are not notified of the changes.

These three different scenario's highlight that our approach transparently handles different users and various combinations of screensharing, modelsharing, and no sharing at all.

VI. Related Work

The terminology used in this paper is based on the two types of collaboration provided by AToMPM [1], a collaborative web-based meta-modelling environment. AToMPM provided both modelshare and screenshare, although this was not unified, as presented here. Additionally, this earlier work was purely an implementation feature, and certainly not explicitly modelled as presented in this paper. The way in which operations are executed is also different: whereas in our framework it is possible to start screensharing or modelsharing with any accessible model (i.e., taking into account access control), AToMPM actually required the owning user to *invite* other users for modelsharing or screensharing.

Several other collaborative tools exist, each with their own distinct approach to collaboration. Mostly, however, tools support either screensharing or modelsharing. Screensharing, sometimes termed whiteboarding [3], is supported by several tools and is a popular model collaboration approach. For example, WebGME [5] and a tool by Gallardo *et al.* [3] support screensharing, as each user can see the same change immediately. Modelsharing is similar to multi-view modelling [17], where different visualizations of the same model are manipulated concurrently. Each view can be considered as having a different type of visualization, or concrete syntax.

Another type of collaboration is achieved through model repositories, such as MDEForge [18]. While they support a browsable repository of models, where modifications can be made on models, they do not consider the split between abstract and concrete syntax as deeply as in our work. MDEForge is primarily focussed on reusing existing models (as a model repository) and performing model management operations on them (e.g., through model transformations [19]). Although a programmatic interface exists, to the best of our knowledge, the tool does not focus on the model editing part.

Orthogonal to our approach are different techniques to actually make collaboration work in a safe way. That is, to define strategies for when conflicting changes occur. One such strategy, as implemented by WebGME [5], is to branch off from the model when changes are made. This relies on model versioning, where each modification is stored explicitly and creates a new branch starting from the original model. This impedes collaboration, as users enter different branches and later have to perform model merging to resolve potential conflicts. Another strategy is to use locking, as implemented by Gallardo *et al.* [3], where users can request permission to modify the model. Only a single user can modify the model at the same time, thereby preventing conflicting operations. All these approaches are orthogonal to our approach, as we consider the level at which sharing happens, and where we actually perform the collaboration. For example, when modelsharing, the collaboration happens at the level of M_{AS}, which is where these other techniques come into the picture.

Related work on the concrete syntax framework, particularly to projectional editing [20], was presented elsewhere [9].

VII. Conclusion

The complexity of engineered systems is ever increasing, resulting in a plethora of larger and more diverse models. This increase in complexity can be addressed by collaborative model development, also known as Concurrent Engineering. We distinguished two types of collaboration, depending on which artefacts are shared between users: screenshare and modelshare. With screenshare, different modellers share the exact same representation of the model, including, for example, the location of elements. With modelshare, different modellers share the same model, but have different views (or visualizations) on this model. We implemented both approaches based on our framework for concrete syntax, thereby enabling the unification of both approaches and reusing several operations. Since both are unified, it is possible to combine these approaches, such that different combinations of screensharing and modelsharing become possible.

In future work, a compromise between modelshare and screenshare could be made, where only "relevant" concrete syntax operations are shared. For example, if the size of elements is relevant, these should still be propagated, while location information should not. Another direction of future work is the extension of this modelling environment to a simulation and debugging environment, for example using existing techniques [21]. The future work of our concrete syntax framework is also applicable in this context, in particular the use of an interaction model [9], which can interact with resolution strategies (e.g., locking).

Acknowledgements

This work was partly funded by a PhD fellowship from the Research Foundation - Flanders (FWO). This research was also partially supported by Flanders Make vzw, Flanders' strategic research centre for the manufacturing industry.

References

[1] E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. Van Mierlo, and H. Ergin, "AToMPM: A web-based modeling environment," in *Proceedings of MoDELS'13 Demonstration Session*, 2013, pp. 21–25.

[2] C. Debreceni, G. Bergmann, I. Ráth, and D. Varró, "Property-based locking in collaborative modeling," in *Proc. MoDELS*, 2017, pp. 199–209.

[3] J. Gallardo, C. Bravo, and M. Redondo, "A model-driven development method for collaborative modeling tools," *Journal of Network and Computer Applications*, vol. 35, pp. 1086–1105, 2012.

[4] P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, and M. Wimmer, "An introduction to model versioning," in *Formal Methods for Model-Driven Engineering - International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM)*, 2012, pp. 336 – 398.

[5] M. Maróti, T. Kecskés, R. Kereskényi, B. Broll, P. Völgyesi, L. Jurácz, T. Levendovszky, and A. Lédeczi, "Next generation (meta)modeling: web- and cloud-based collaborative tool infrastructure," in *Proceedings of the Workshop on MPM*, 2014, pp. 41 – 60.

[6] M. Gerosa, M. Pimentel, H. Fuks, and C. de Lucena, "Towards an engineering approach for groupware development: learning from the AulaNet LMS development," in *Proceedings of the 9th International Conference on CSCW in Design*, 2005, pp. 329–333.

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

[8] S.-L. Beatteay, N. Savant, and E. Olivares, "Building Conclave: a decentralized real-time, collaborative text editor," 2018, https://hackernoon.com/building-conclave-a-decentralized-real-time-collaborative-text-editor-a6ab438fe79f.

[9] Y. Van Tendeloo, S. Van Mierlo, B. Meyers, and H. Vangheluwe, "Concrete syntax: A multi-paradigm modelling approach," in *Proc. SLE*. ACM, Oct. 2017, pp. 182 – 193.

[10] P. J. Mosterman and H. Vangheluwe, "Computer automated multi-paradigm modeling: An introduction," *SIMULATION*, vol. 80, no. 9, pp. 433–450, 2004.

[11] H. Vangheluwe, J. de Lara, and P. J. Mosterman, "An introduction to Multi-Paradigm Modelling and Simulation," in *Proceedings of the AIS'2002 Conference (AI, Simulation and Planning in High Autonomy Systems)*, 2002, pp. 9 – 20.

[12] Y. Van Tendeloo and H. Vangheluwe, "The Modelverse: a tool for multi-paradigm modelling and simulation," in *Proceedings of the 2017 Winter Simulation Conference*. IEEE, Dec. 2017, pp. 944 – 955.

[13] A. Kleppe, "A language description is more than a metamodel," in *Fourth International Workshop on Software Language Engineering*, 2007.

[14] S. Van Mierlo, Y. Van Tendeloo, B. Meyers, and H. Vangheluwe, "Domain-specific modelling for human-computer interaction," in *The Handbook of Formal Methods in Human-Computer Interaction*. Springer, 2017.

[15] M. Petre, "Why looking isn't always seeing: Readership skills and graphical programming," *CACM*, vol. 38, no. 6, pp. 33–44, 1995.

[16] G. Brunet, M. Chechik, S. Easterbrook, S. Nejati, N. Niu, and M. Sabetzadeh, "A manifesto for model merging," in *Proceedings of the 2006 International Workshop on Global Integrated Model Management*, ser. GaMMa '06. New York, NY, USA: ACM, 2006, pp. 5–12.

[17] International Organization for Standardization, "ISO/IEC/IEEE 42010:2011, systems and software engineering — architecture description," 2017, https://www.iso.org/standard/50508.html.

[18] F. Basciani, J. Di Rocco, D. Di Ruscio, A. Di Salle, L. Iovino, and A. Pierantonio, "MDEForge: an extensible web-based modeling platform," in *Proceedings of the Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE)*, 2014, pp. 66 – 75.

[19] J. Di Rucco, D. Di Ruscio, A. Pierantoini, J. Sánchez Cuadrado, J. de Lara, and E. Guerra, "Using ATL transformation services in the MDEForge collaborative modeling platform," in *Proceedings of the International Conference on Model Transformation (ICMT)*, 2016.

[20] M. Voelter, J. Siegmund, T. Berger, and B. Kolb, "Towards user-friendly projectional editors," in *Proceedings of the International Conference on Software Language Engineering*, 2014, pp. 41–61.

[21] S. Van Mierlo, "A multi-paradigm modelling approach for engineering model debugging environments," Ph.D. dissertation, University of Antwerp, 2018.