Real-Time Collaborative Modeling across Language Workbenches – a Case on Jetbrains MPS and Eclipse Spoofax

Samuël Noah Voogd
Vrije Universiteit Amsterdam
The Netherlands
s.n.voogd@student.vu.nl

Kousar Aslam
Vrije Universiteit Amsterdam
The Netherlands
k.aslam@vu.nl

Louis van Gool
Canon Production Printing
The Netherlands
louis.vangool@cpp.canon

Bart Theelen
Canon Production Printing
The Netherlands
bart.theelen@cpp.canon

Ivano Malavolta
Vrije Universiteit Amsterdam
The Netherlands
i.malavolta@vu.nl

***Abstract*—Software tools known as language workbenches are used to define and deploy custom (domain-specific) languages for the purpose of modeling (specific parts) of a system of interest. Because system modeling is a practice that stands to benefit from real-time collaboration, technologies offering real-time collaborative mechanisms for language workbenches are starting to make an appearance. However, these collaboration technologies are generally limited to providing collaboration among clients of a single designated workbench. If the collaborating engineers wish to use different workbenches to work on the model, cross-platform support for collaborative modeling becomes a necessity.**

In this paper we propose Parsafix, a tool-based approach for achieving real-time collaboration between different language workbenches for users collaborating on models conforming to the same domain-specific language. We propose the main components and mechanisms that make up Parsafix, as well as the implementation of a prototype tool supporting those mechanisms. The prototype tool allows for collaboration between users of JetBrains MPS and Spoofax (within the Eclipse IDE), by making use of the IDEs' respective real-time collaboration technologies Modelix and Saros. A hands-on session is proposed to showcase the feasibility of having collaborative modeling across different language workbenches through Parsafix.

***Index Terms*—Model-driven development, Computer languages, Collaboration**

I. Introduction

Language workbenches are tools that allow for defining and designing (domain-specific) languages and their development environments [1]. Over the years several of such tools have appeared throughout the software industry, each with their own advantages and disadvantages. There exist language workbenches that include their own Integrated Development Environment (IDE) and form a more or less standalone product, as well as those that can be installed as plugins for existing IDEs [2]. The choice of workbench will ultimately boil down to a language or model engineer's specific situation and preference.

Defining a Domain-Specific Language (DSL) can be an arduous and complex task. The same goes for utilizing a DSL, once it has been realized, to model (parts of) a system of interest [3]. Due to their potential complexity, these language and model engineering tasks may benefit from the involvement of more than a single engineer [4] [5] [6]. A trivial collaboration strategy is to share files among engineers, either manually or by using version-control systems such as Git. This method is effective, albeit somewhat inefficient due to the inevitable delay that is introduced by having to wait for files to be sent and received, as well as the time it takes to merge independent changes [7]. Real-time collaboration eliminates this inefficiency by allowing engineers to work on the same model(s) at the same time, thanks to automatic synchronization of model edit operations between users [8].

Owing to the benefits of such a feature, a number of language workbenches, or the IDEs they reside in, are already in possession of some kind of real-time collaboration technology. Examples of such technologies are Modelix [9] for JetBrains MPS [10] and Visual Studio Live Share for Visual Studio Code's modeling SDK [11]. The authors expect that more and more language workbenches will adopt some form of real-time collaboration in the future.

Although the contemporary development of real-time collaboration techniques appears promising, the collection of language workbenches from which to choose begs the following question: What if the parties who wish to collaborate make use of different language workbenches?

When it comes to language engineering, this may not be a very pressing issue. Each language workbench is built differently, due to differences in implementation choices and the definitions that creators adhere to concerning what exactly a language workbench should be. Because of these differences between language workbenches, the inner structure, syntax and general aesthetic of a language implementation can differ dramatically from workbench to workbench [2]. Since language definitions are language-workbench specific, it is not feasible to develop a single language definition using multiple language workbenches.

This research is funded by the Rijksdienst voor Ondernemend Nederland (RVO) through the ITEA3 BUMBLE project (18006).

However, once a language has been implemented independently inside multiple language workbenches, collaboration among these workbenches starts to make more sense. Although the implementation and exact syntax of a language can and will differ per workbench, the workbenches should ultimately produce the same result, i.e., a model defined in terms of a single predefined language [12].

At the time of writing, research on collaboration across different language workbenches is very limited. The collection of publicly available research regarding language workbenches appears to remain confined to research that examines language workbenches independently of one another, either for the purpose of evaluating a workbench or comparing different workbenches [2] [13] [14].

In our work, we fill this research gap by presenting **an approach to enable *real-time collaboration* among engineers working on models conforming to the same DSL, but in different language workbenches**.

Our work closely relates to the ideas of model transformation as well [15] [16] [17]. The workbenches on each side of collaboration have similar syntax which ensures that the information is preserved both ways. To provide a practical realization of our approach, we developed a prototype that provides the aforementioned real-time collaboration functionality for a specific use-case. This research was conducted in collaboration with Canon Production Printing, the Canon division for professional digital printers and accompanying workflow management. The technologies chosen as the use-case for the prototype results from the ongoing development in the field of model-driven engineering and language engineering performed at Canon Production Printing.

The language workbenches used for the development of the prototype are JetBrains MPS and Spoofax (within the Eclipse IDE) [18], along with the collaboration plugins Modelix for MPS and Saros for Eclipse [19]. The DSL that was used for the purpose of modeling in the use-case is a custom DSL developed at Canon Production Printing. This DSL was implemented inside both language workbenches, compliant with the native implementation style of each workbench.

In summary, the main contributions of this paper are: (i) an approach for real-time collaboration across different language workbenches and (ii) a prototype implementing the proposed approach and (iii) a description of how the prototype can be used in practice.

Section II discusses the application of the approach proposed in this work. Section III gives a high-level description of the approach. Section IV describes how the approach is used by discussing a prototype implementation. Section V outlines a hands-on session to be performed during HoWCoM 2021 using the prototype tool. Finally, section VI concludes this work and discusses the potential future directions for the enhancement of the approach and prototype tool.

II. Case

The need for cross-platform modeling across language workbenches stems from the fact that there are several language workbenches to choose from. Popular examples include MetaEdit+ [20], JetBrains MPS, Rascal [21], Spoofax and Xtext [22]. In addition to this, the field of custom language engineering is as alive as ever and the use of model-driven engineering through the use of DSLs is widespread throughout the academic and commercial domain [3] [23]. The ongoing popularity of model-driven engineering is likely to stimulate the further advancement of existing language workbenches as well as the potential introduction of new workbenches [24]. Specifically, in the context of model-driven engineering, system modeling is a fundamental discipline that poses its own challenges [25]. Performing this task collaboratively can yield improved efficiency and higher model quality over modeling performed individually [26] [27]. The emergence of technologies providing real-time collaboration functionality for model engineering further supports the claim that collaboration is a desirable feature in this field [14] [9] [28] [29].

Cross-platform support becomes relevant as soon as engineers, working on the same modeling project, have a preference for, or are required to work with different language workbenches. For example, such a scenario could be expected to occur when multiple modeling teams from different organisations or companies are required to work together. In such a case, it would be beneficial in terms of efficiency, if both parties could remain working in the language workbench of their choice, instead of forcing one side to adopt the other's workbench.

Imagine the following scenario. Two teams of software engineers, *Team A* and *Team B*, are tasked to collaborate with each other in developing software for a machine that is part of a big cyber-physical system [30]. The machine is composed of complex mechanical components as well as sophisticated software in order to perform its required function. As a first step, both teams are required to engineer a high-level model of the machine in order to decide on how to build it. To do this, both teams make use of the same DSL.

Team A is assigned the task of designing interfaces between software components. For this team, it is important to know which parts of the machine will require interaction with one another in order to determine their connection methods. *Team B* is responsible for implementing the services to be provided by each software component. For this team, it is important to know the limitations of the hardware their software is to run on. In other words, the design choices of one team are dependent on those of the other.

In this scenario, *Team A* makes use of the DSL in an IDE that supports the description of interactions among software components. *Team B* makes use of a different IDE that splits the models written in the DSL up into different textual representations. In their IDE, the software engineers can model the software in great detail, and also view the high level specifications. Both IDEs already support methods for allowing real-time collaboration between their respective clients.

Through the use of a tool that links the two IDEs together, the two teams could collaborate with one another on the engineering of the model as it is being developed. This has the potential to speed up the modeling process and make it more efficient, as both parties can see relevant information appear in their own IDEs while the other party is modeling it.

Naturally, such a tool could also be beneficial for far less complex cases. Imagine two system engineers want to model a system together using the same DSL, but both of them prefer a different language workbench. In that case, the tool would allow them to stick to their own workbench, but still collaborate on the same project.

In light of the above discussion, it is evident that cross-platform collaboration among language workbenches is desirable. Therefore, the approach proposed in this work has been designed to provide collaboration across different language workbenches between users who are modeling using the same DSL. This form of cross-platform collaboration is completely independent from the considered DSL or modeling use-case; rather, the proposed approach has the potential to support any DSL and consequently, any model.

III. The Parsafix Approach

The following section gives a high-level overview of the Parsafix approach and how it accomplishes its intended purpose.

A. Name and GitHub Repository

The working title given to the tool-based approach is *Parsafix*, a play on the words *parse*, *Saros*, *Spoofax* and *Modelix*. This name alludes to the fact that the initial intended purpose of the approach, hereinafter referred to as Parsafix, was to parse data received from MPS and Spoofax by means of the respective collaboration plugins, Modelix and Saros. As is described later in this section, the prototype implementation of the approach supports this exact use-case.

Parsafix is available in the following GitHub repository: https://github.com/blended-modeling/parsafix. The repository contains the source code of the current implementation of Parsafix and a guide on how to setup, run, and use the prototype tool.

B. Purpose

The purpose of Parsafix is to facilitate real-time collaboration among different users for engineering a model using the same DSL but different language workbenches. Parsafix forms a link between pre-existing collaboration tools, and thus an indirect link between language workbenches. Parsafix should therefore be seen as a parsing pipeline from one collaboration tool to another. For this reason, Parsafix requires that the language workbenches it links already possess some form of real-time collaboration functionality.

Parsafix sits in between the workbenches' real-time collaboration functionality. Parsafix exists completely independent of the other components in the collaboration architecture, that is, neither the language workbenches nor their collaboration technologies are aware of its existence.

This approach was chosen deliberately in order to make Parsafix generic. In this way all other components can continue being developed independently and their implementations will not have to be altered in order for Parsafix to work.

C. Collaboration Architecture

Figure 1 shows the collaboration architecture of a Parsafix-enabled collaboration setup. The setup comprises two language workbenches, their respective real-time collaboration technologies, and Parsafix itself. For the sake of clarity, the two different language workbenches and their respective real-time collaboration technologies shall be referred to as collaboration systems A and B. The figure is symmetrical in the sense that Parsafix treats collaboration systems A and B in exactly the same way from a conceptual point of view, although the exact implementations of the communication and data-handling methods will differ per case.

The figure shows that each collaboration system contains a real-time collaboration technology, which is particular to a specific language workbench. In addition, each collaboration system contains clients that are instances of the language workbench. The number of potential clients is specific to the collaboration technology, but by definition it should allow for at least two homogeneous clients to collaborate on a project. The arrows in the figure represent the bidirectional flow of project data and edit data that the real-time collaboration technologies use to synchronize their workbench clients.

Collaboration systems A and B are independent of one another. Individually, the collaboration systems facilitate real-time collaboration between clients of a single language workbench. This could be any language workbench, although not all language workbenches currently possess a technology that offers real-time collaboration. As mentioned in Section III-B, language workbenches lacking real-time collaboration technologies are not compatible with Parsafix, as these technologies are central to Parsafix's functioning.

As shown in Figure 1, Parsafix is able to treat both *projects* and *models* within the language workbenches. Not all workbenches treat their project structure in the same way, but generally speaking a workbench allows its users to create projects and define models within the projects. Conceptually, Parsafix is independent of whether the models are simply files within a directory or are part of a more complex project structure. The models within a project are written using DSLs. The DSL used may differ per model, but any single model can only be defined in terms of a single DSL (or a predefined integration of DSLs, ultimately forming a single DSL).

Every form of real-time collaboration among users of a language workbench will require data to be communicated between the clients that describes the state of the project and the models it contains. Without this, a workbench client would not know what its peers are doing and what the collaborative project looks like. In order for Parsafix to have access to

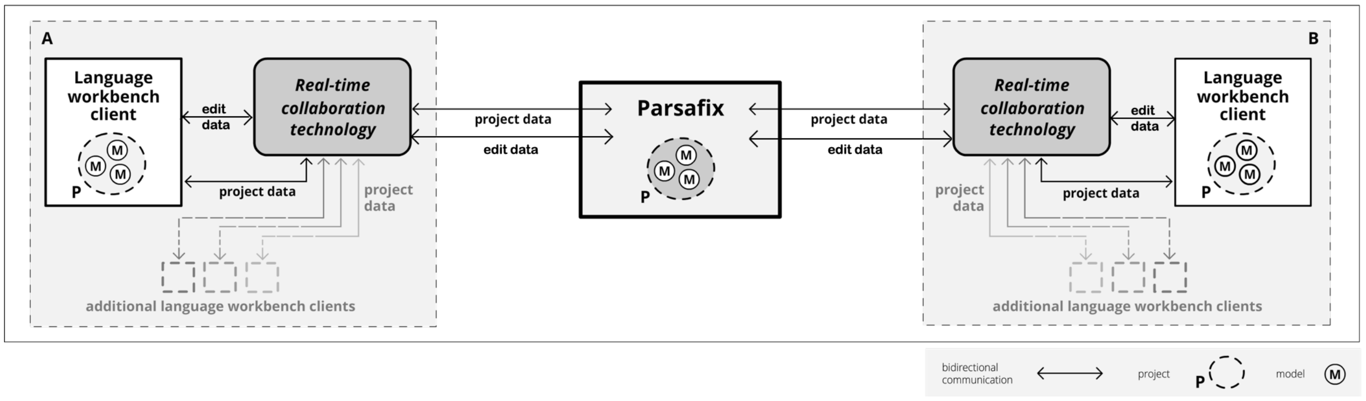

A
Language workbench client
edit data
Real-time collaboration technology
project data
P
M
project data
additional language workbench clients
project data
edit data
Parsafix
B
bidirectional communication
project
model

Fig. 1. Collaboration between two different language workbenches through Parsafix

the project data, it acts as a language workbench client, asking the workbench's respective collaboration technology for the required information. It does this for both workbenches, receiving data from both sides. In this way, both collaboration technologies facilitate communication with Parsafix, but remain unaware of the fact that it is not just another language workbench client. The flow of data in the figure is bidirectional. This is because Parsafix may also wish to push data into the collaboration systems, altering the models within their respective projects. This push of data is generated by Parsafix as it performs its parsing function between the data of the two collaboration systems. Parsafix's internal logic is described in the following section.

D. Internal Structure

Figure 2 describes the internal structure of Parsafix. It is a zoomed in version of the box representing Parsafix in Figure 1. It shows Parsafix as consisting of three main parts that each perform a specific function. On the left there is a part that handles the communication with the collaboration technology of workbench A. On the right is the part performing the same function for collaboration technology B. Finally, in the center is a part that takes care of parsing the incoming data. Once again, the figure is symmetrical, i.e., Parsafix treats sides A and B in exactly the same way from a conceptual point of view, although the exact implementations of their communication and data handling methods will differ per case.

Each of the three parts of Parsafix's internals contains a data model. The two communication parts contain data models that describe the projects of the side they communicate with. Parsafix receives project data from both sides A and B and uses it to construct their data models.

The parsing part of Parsafix also contains a data model, which shall be referred to as the Parsafix data model. The function of the parsing part is to ultimately unify data models A and B into the Parsafix data model. In this way, the Parsafix data model comes to describe the cross-platform project as a whole.

In the following we describe the internal steps performed by Parsafix during a collaborative modeling session. Parsafix takes the data it receives from sides A and B and uses it to create a data model for each side, representing the models inside the workbench projects. It then maps that information onto the Parsafix data model, essentially combining data from the other two data models. The Parsafix data model comes to contain all the information necessary to reconstruct the project in either workbench, or, more interestingly, in both at the same time.

Because Parsafix is treated as a regular collaboration client by both sides, it receives new project data whenever an actual client of either workbench makes a change to the project data. When this happens, Parsafix updates the internal data model corresponding with that workbench's project. It then maps the change onto the Parsafix data model. Because the Parsafix data model has a unified version of both projects A and B, the change that was applied on one side is automatically applied to the other side as well. The change in the Parsafix data model is then taken and used to generate an edit command, which is sent to the opposing side. Upon receiving the edit command, the opposing collaboration technology will pass it on to its workbench clients, which will incorporate the change as if it had been performed by one of their peers. In order to stay up to date, Parsafix also updates the data model of the opposing workbench to correspond with the change that was made. In this way, the two workbenches are made to ultimately contain the same models, and thus, cross-platform collaboration between language workbenches is achieved.

E. Data Parsing

The aforementioned Parsafix data model takes the form of multiple Abstract Syntax Trees (ASTs). An AST describes a model's elements and how they relate to one another in a hierarchical fashion. It contains all the information necessary to recreate a model using any desired concrete syntax [31] [32]. It does not include syntax- or representation-specific elements such as punctuation and delimiters, and thus provides a generic representation of a model. Parsafix aims to link the projects of two language workbenches that potentially make

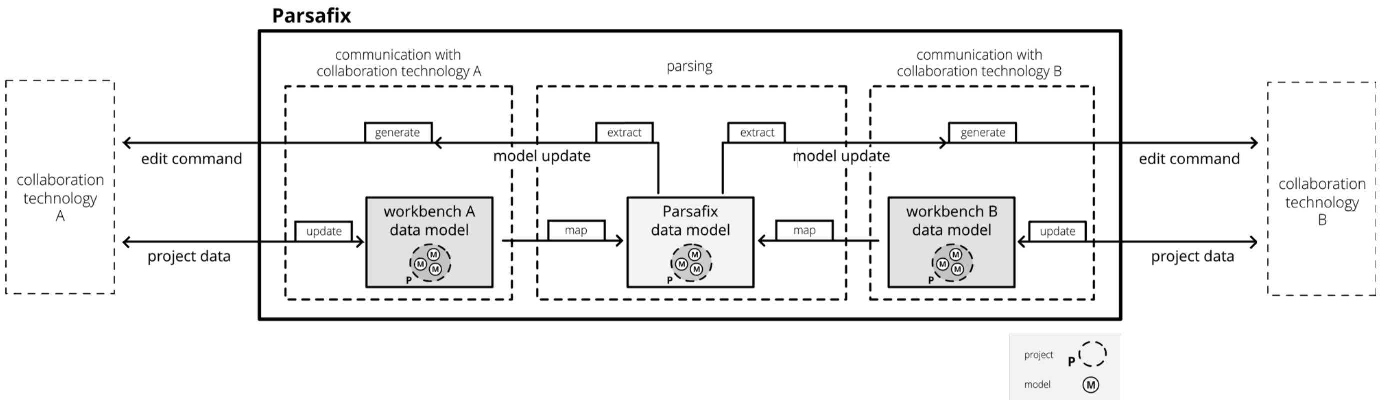


Fig. 2. Internal structure of Parsafix showing how data communicated between the tool and two different collaboration technologies is processed

use of vastly different syntaxes to describe the same models. Parsafix approaches this problem by constructing an AST for each model inside the Parsafix data model.

As can be seen in Figure 2, this means that the project represented in the Parsafix data model and the projects of the workbenches come to contain the same number of models. This is the precise aim of Parsafix, to ensure that both sides are working on the same project and thus on the same models. The only additional information that the Parsafix project contains compared to those of the workbenches, is identifiers pointing back to each workbench project. This is necessary in order to keep track of how the workbench projects correspond with the collaborative project as a whole.

In what form the project data from the language workbenches arrives at Parsafix is dependent on the language workbench and its corresponding collaboration technology. Figure 3 illustrates how Parsafix receives project data from a collaboration technology and ultimately maps it to the Parsafix data model. Below we describe this process as seen in the Figure 3 for the two representations that project data can take, namely AST data or plain text data.

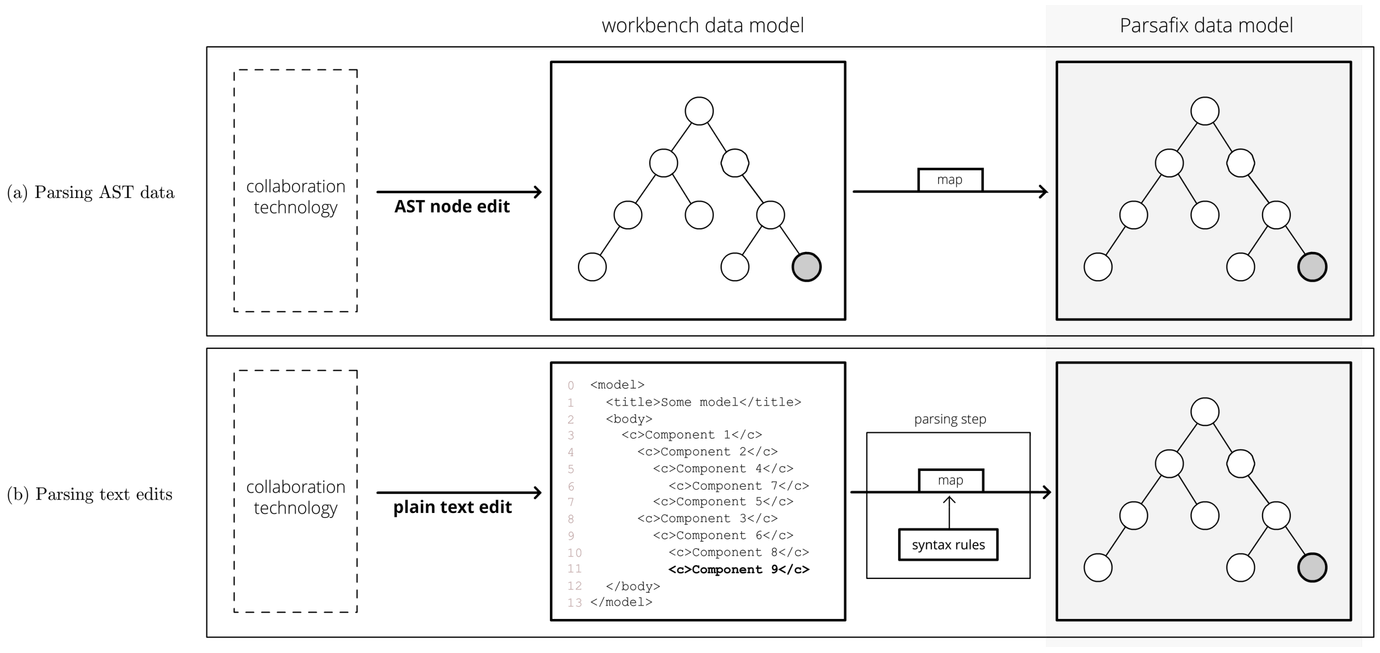
1) Parsing AST data: Ideally speaking, Parsafix would receive project data from all workbenches in the form of ASTs or something similar. Some aspects of an AST will inevitably be implementation- and thus language workbench-specific, such as the exact syntax of type and property names. However, aside from this, no real parsing is necessary for Parsafix to map an AST onto the Parsafix data model, because the data model itself is made up of ASTs. Only a small segment of the mapping step needs to be tailored specifically for the workbench in question.

Figure 3a shows the steps that take place when Parsafix receives project data from a collaboration technology in the form of an AST node edit. Such a node edit is either the addition, removal or property change of a node in the AST of a model. Upon receiving the edit, Parsafix updates its corresponding workbench data model to coincide with the edit and stay synchronized with the workbench's project. Then it simply maps the change onto the Parsafix data model. All it has to do is find the node in question in the model's corresponding AST and delete or update it, or add the node as a new child to its parent.

2) Parsing text edits: Not all language workbenches are made to collaborate by sharing data describing the ASTs of models. Collaboration can also be achieved by means of sharing the changes in the concrete syntax of a model, for example by propagating the plain-text changes in the source code. In such a case, more sophisticated parsing steps must be implemented to allow for a precise reconstruction of the data model in Parsafix, most of which will likely be language workbench and DSL-specific.

Figure 3b shows the steps that Parsafix takes in order to go from a plain-text edit of a model's source code, to parsing it to the Parsafix data model. Once again Parsafix starts by updating the workbench data model when it receives an edit. However, when edit data comes from a purely text-based language workbench (e.g., Spoofax), the workbench data model is now plain text. Parsafix must then parse this plain text in order to properly map it onto a corresponding AST inside the Parsafix data model. This parsing step consists of going over the parts of the text relevant to the edit and using syntax rules to determine whether the syntax is valid and what the new text means for the model it represents. If a new line was added, this could mean the addition of a new element, and the removal of a line could signify a deletion. New characters in a certain position could indicate a property change.

We now discuss how Parsafix sends project data back to a collaboration technology in the form of an edit command. As shown in Figure 2, the parsing step inside Parsafix also includes extracting model updates from the Parsafix data model. This happens after a change made on one side has been mapped to the Parsafix data model. In order to keep both sides synchronized with one another, the opposing side must be made aware of the change that occurred. As mentioned earlier, the project inside the Parsafix data model has identifiers pointing to the projects of the workbenches. The model update consists of the identifiers and the properties of the actual change. This model update is then used to generate an edit command that is sent to the opposing side in order to synchronize it with the collaborative Parsafix project.



```
<model>
  <title>Some model</title>
  <body>
    <c>Component 1</c>
      <c>Component 2</c>
        <c>Component 4</c>
          <c>Component 7</c>
        <c>Component 5</c>
      <c>Component 3</c>
        <c>Component 6</c>
          <c>Component 8</c>
          <c>Component 9</c>
  </body>
</model>
```

Fig. 3. Data parsing performed by Parsafix on project data received from a collaboration technology

F. Limitations and Assumptions

1) Parsing: Parsafix's main responsibility is parsing the project data it receives and mapping it onto the Parsafix data model. In doing so for project data from two language workbenches, the data model comes to describe a cross-platform collaborative project.

In order to do this, Parsafix needs to know the following information about the data it is receiving: whether it is AST data or text data and how to parse and map the data. For now the prototype tool expects a near to perfect input and cannot handle discrepancies and unexpected inputs.

a. The form of the incoming data depends on the language workbench and its collaboration technology. As already mentioned, this will either be data pertaining to a data model such as an AST, or data describing a plain text edit in the source code. If Parsafix is to link collaboration technologies that do not provide AST data, then workbench-specific and to some degree DSL-specific implementations will have to be written. There is no universal protocol specifying how AST data or text edits should be communicated, so even though the form of the data is conceptually limited to these two options, the precise form of the incoming data cannot be predicted. For this reason Parsafix needs to know explicitly per language workbench which of the two options it is dealing with, as such information cannot be reliably deduced from the data itself.

Although the requirement to make the form of the incoming data explicit can be seen as a limitation to Parsafix in terms of generalizability, it poses no significant threat in this regard, because there exist only a limited number of language workbenches [2].

b. After Parsafix has been made aware of what form the incoming model data is in, it requires one more piece of vital information in order to work. It needs to know how to actually parse the data and map it onto the Parsafix data model.

As mentioned in the previous section, this is relatively straightforward in the case of AST data. ASTs are data models that encompass both the structure of a model as well as its properties. In other words, if we have a model's AST, we have no need for the DSL definition, because all the data necessary to deduce the DSL elements that make up the model is already present in the AST. This is why ASTs are so generic and the reason the Parsafix data model is made up of ASTs. When Parsafix receives AST data from a collaboration technology, it can map this data more or less directly onto the ASTs inside the Parsafix data model. The current version of Parsafix does not map AST-based edits directly to text-based edits and vice-versa. The ASTs are not identifiable for now, this may be handled in future by using ID-management system

In contrast to AST data, the parsing step is more complicated when it comes to plain text data. Parsafix cannot parse and map model data if it does not know what elements make up the model and how they relate to one another, that is, information that could otherwise be found in the model's AST. In such a case, Parsafix requires the implementation of a parser that pulls the plain text data apart using the DSL definition of the language the model is written in. Any efficient parser implementation could be used for this, but unfortunately the DSL definition cannot be generalized.

The limitation in terms of generalizability due to the aforementioned parsing step presents no true threat to Parsafix's utility. There are two reasons for this. Firstly, a great deal of research has already been performed in the field of text parsing. Therefore, there is no need to reinvent the wheel when developing custom parsers for those language workbenches that collaborate by means of textual edits. In future, the existing research could be employed through the use of a plugin system within Parsafix, where the textual data is given as input to a parser plugin and it spits out the desired data model.

Secondly, the described parsing step is also performed by language workbenches that allow their users to edit models in free-form text. In order to be considered a true language workbench, a workbench must be able to reason about the models inside its projects [1]. This is why language workbenches that work with free-form text editors generate ASTs from their users' input. In other words, even the language workbenches that might be made to collaborate by sharing textual edits, also have an AST of their models at their disposal. Because this is the case, any collaborative technology designed for language workbenches could ultimately be made to share AST data. If this is not currently the case for a language workbench of interest, its collaboration technology could come to support AST data sharing at some point in the future.

2) DSL Symmetry: It should be noted that Parsafix can only perform its intended function if the language workbenches it connects contain implementations of the same DSL. If this is not the case, Parsafix will likely ask a workbench to apply changes that it cannot understand. For example, imagine that two language workbenches are connected through Parsafix and the intent is to model using the Unified Modeling Language (UML), but only one side has UML implemented. When the workbench with the UML implementation adds a UML class, Parsafix will try to send the opposing side an edit command telling it to do the same using its implementation of UML. Seeing as there is no UML implementation present, the workbench will not know what to do and (at best) produce an error or (at worst) crash. Parsafix exists independently of the workbenches and their collaboration technologies and thus cannot solve this problem. It could be made to detect whether a DSL implementation is present or not and consequently decide not to send edit commands to parties lacking an implementation. In any case, absence of the DSL implementation on either side will cause the collaboration between the workbenches to halt. In future, Parsafix can be enhanced to be able to map DSL elements in one workbench to DSL elements of the other workbench (and vice versa), which will facilitate the teams collaborating with each other using different workbenches by creating an overlap of elements among workbenches.

Although the requirement for symmetry between language workbenches in terms of DSL implementation can be seen as a limitation, it forms an inherent part of the main concept behind Parsafix. Parsafix aims to facilitate collaboration between users working on a model in the same DSL.

A language workbench only offers support for models written in DSLs that it is familiar with. This is true even without taking Parsafix into account. Collaboration technologies of individual language workbenches cannot provide collaboration between their workbench clients if the clients do not already possess a means of interpreting the collaborative models. This interpretation must either take the form of a matching DSL definition, or some workaround such as a fallback general-purpose language. For the same reason, Parsafix should not offer a collaboration technology data that its workbench clients will not know how to interpret.

Language workbenches with collaboration technologies that share source code, i.e. plain text data, form an exception to this rule. These workbenches could be made to simply display the model data they receive as plain text. All Parsafix would need to do is pick a text parser for the desired DSL and employ it in parsing project data on the side of the language workbench lacking the DSL implementation.

Of course, if collaboration technologies of language workbenches have found other ways to circumvent asymmetry in terms of DSL implementation, such as the use of a fallback general-purpose language, Parsafix could be made to make use of this as well.

3) Collaboration Technology Communication: In order for Parsafix to communicate with the collaboration technologies of language workbenches, it must be instructed on how to do so. Every collaborative technology is likely to communicate with its clients at least somewhat differently and thus generalization is simply not possible. In other words, Parsafix requires custom implementations for communicating with each of the collaboration technologies it wishes to link. However, once a means of communication has been implemented for a given language workbench, then it can be used every time Parsafix is used to link such workbench, independently of the DSL being used.

IV. Prototype

In this section we describe the technical details of the current prototype implementing Parsafix. For the sake of clarity, we will focus on a specific use-case involving two language workbenches based on the MPS and Eclipse modeling platforms.

A. Technologies

The workbenches chosen as a use-case for the prototype are MPS (version 2020.1.1) [10] and Eclipse Spoofax (version 2.5.13) [18]. For MPS there exists the plugin Modelix (as pulled from https://github.com/modelix/modelix on 01/04/2021) [9] and for Eclipse the plugin Saros (version 16.0.1) [19], both of which provide real-time collaboration between clients of their corresponding IDEs. The DSL utilized in the use-case is developed at Canon Production Printing, the Open Interface Language as exemplified in [33], which allows describing interfaces and their behaviour through state-machines.

B. Collaboration Architecture

Figure 4 shows how the prototype implementation instantiates the collaboration systems of the generalized architecture in Figure 1.

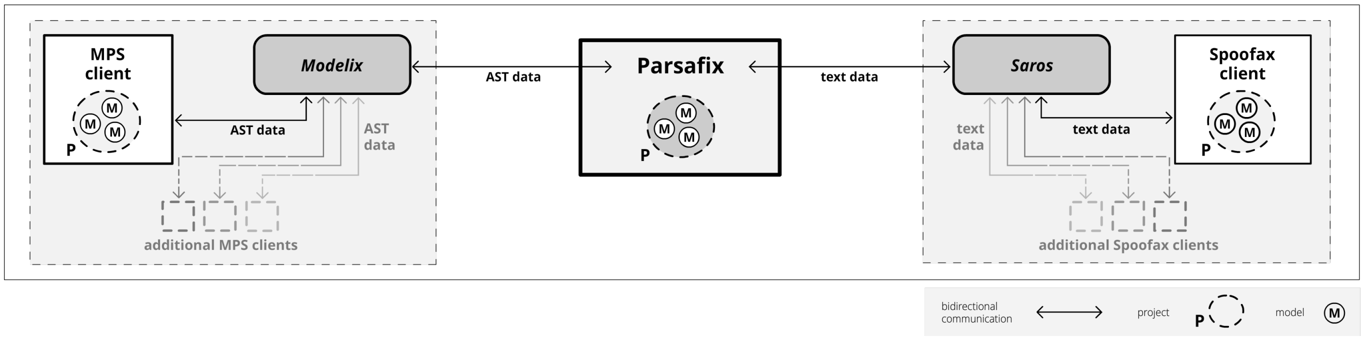


Fig. 4. Collaboration architecture showing the collaboration between MPS and Spoofax through Parsafix

On the left we find the language workbench MPS, the clients of which are collaborating on a modeling project using the Modelix plugin. Modelix achieves collaboration by sharing AST data among clients. For this reason, while posing as an MPS client, Parsafix also receives this AST data.

The right side of the figure shows the language workbench Spoofax. Spoofax itself is a plugin that can be installed and used inside the Eclipse IDE. Here Spoofax controls all the language workbench-related functionalities and Eclipse provides the environment in which to define and utilize DSLs made using Spoofax. Eclipse achieves collaboration using the Saros plugin. Saros was designed for Eclipse rather than Spoofax and can thus be used by Eclipse users to collaborate on all kinds of projects, including those unrelated to custom languages. This is why Saros does not achieve collaboration by sharing AST data, but by sharing source code text edits. Hence, the data arriving at Parsafix as it poses as an Eclipse client, is text data.

C. Internal Structure

As described in Section III-D, Figure 2 shows the inner workings of Parsafix. Inside Parsafix there exists a data model for each language workbench as well as the Parsafix data model, which merges them.

Figure 5 shows what a simplified version of these data models might look like in the case of a simple model written in the use-case DSL, which describes a printer that can be turned on and off. Figure 5a shows the AST constructed in MPS to represent the model, Figure 5c shows the Spoofax source code for the same model and Figure 5b shows how the two come together in the Parsafix data model.

All three versions of the model contain the same elements. The root of the model is an element of type “module”, which has a child of type “interface”, which has two children of type “method”. Inside the MPS AST, the elements have IDs. These IDs are stored with the corresponding elements inside the Parsafix data model. In this way the MPS AST and the Parsafix data model point to one another. The Spoofax source code is plain text, so the modules found within it have no IDs. As a solution to this, Parsafix assigns IDs to the extracted elements when it parses the text. In the case of this example, the line index and offset of an element in the text is used as ID.

As a result, the elements of all three data models can be traced back to one another. When a change is made on the MPS side, the corresponding node in the MPS AST is updated, followed by the corresponding node in the Parsafix data model, followed by the corresponding line of text in the Spoofax source code. If a change arrives from Spoofax, the same process takes place but in reverse.

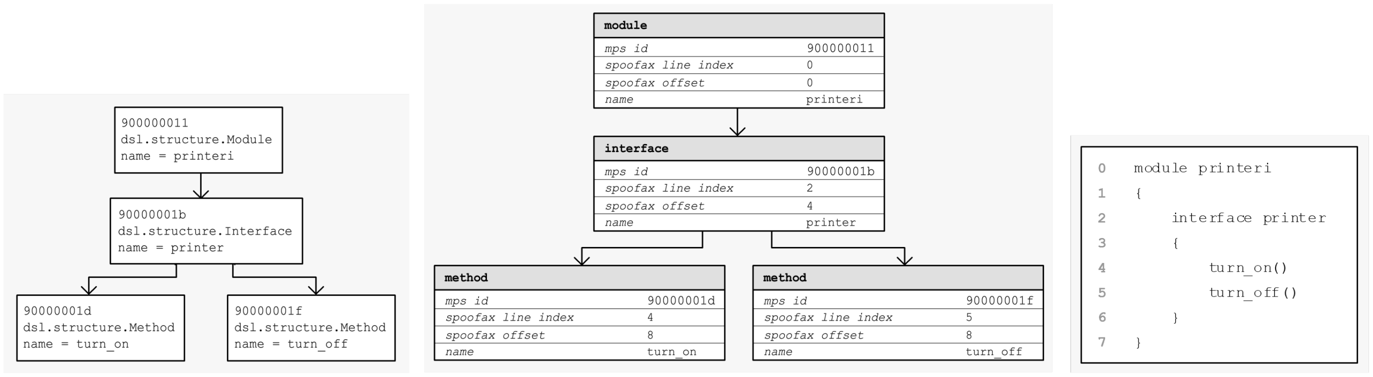
Figures 6 and 7 give an impression of what a model engineer might see when working on the example model within MPS or Spoofax respectively.

D. Data Parsing

1) MPS: Parsing and mapping MPS data onto the Parsafix data model is a simple process. This is because the two data models are practically identical, save for some minor details, as can be seen in Figure 5.

In the example illustrated in the figure, the MPS AST elements are shown to have very specific type names, all starting with “dsl.structure”. These example type names are based on the naming convention that MPS uses for types in its language definitions. Parsafix will need to know which element types on the MPS side correspond with which types on the Spoofax side. Otherwise it will erroneously produce duplicate elements in the project when unifying the MPS and Spoofax data models, because it will see two elements of seemingly different types. For this reason a small parsing step needs to take place for the MPS AST data, where naming conventions from MPS are translated to some standardized *Parsafix names* used for the DSL. In the figure, “dsl.structure.Module” becomes “module” in the Parsafix data model. In turn “dsl.structure.Interface” becomes “interface”, etc. Performing this step can be as simple as a lookup in a table to find the corresponding Parsafix type name. The same goes for workbench-specific naming conventions in other parts of the AST, such as property names.

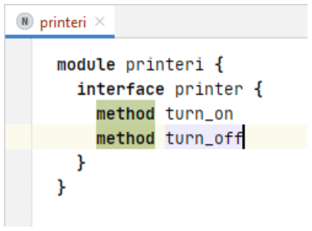
2) Spoofax: In order to parse and map Spoofax data to the Parsafix data model, a parser must be employed. The general working of such a parser is already touched upon in Section III-E. For the current prototype of Parsafix, a minimalistic parser is used that takes the line of text that is altered and uses regular expressions to check what kind of element the line



```
0  module printeri
1  {
2      interface printer
3      {
4          turn_on()
5          turn_off()
6      }
7  }
```

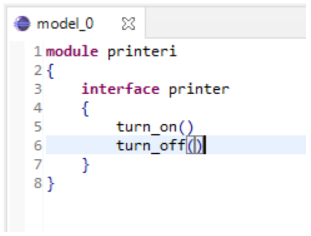
(a) MPS AST (b) Parsafix data model (c) Spoofax source code

Fig. 5. Prototype Data Models



```
module printeri {
  interface printer {
    method turn_on
    method turn_off
  }
}
```

Fig. 6. Simple model definition in MPS



```
module printeri
{
    interface printer
    {
        turn_on()
        turn_off()
    }
}
```

Fig. 7. Simple model definition in Spoofax

contains. The regular expressions correspond with the syntax rules implemented in the language definition of the use-case DSL inside Spoofax.

To unify the data models, the names given to the types and properties of the parsed Spoofax elements are made to coincide with the names of the *Parsafix names* for the use-case DSL, as described in the previous paragraph.

3) Edit Commands: In the case of MPS's collaboration technology Modelix, an edit command consists of an AST node update. Either a node property is altered, or a parent node is told to add or remove a child. In the case of Spoofax, an edit command consists of a text edit. This will either be the addition or deletion of characters. In order to generate these edit commands, all Parsafix needs to know is the ID of the model element that must be updated and the type of change that is being applied. For Modelix this is the ID of the AST node and an indication about which kind of operation is performed (i.e., add, delete or update).

For Saros, an edit command is represented by the line index and offset of the model element in the text, or the line index and offset of one of the properties of a model element, and whether characters must be added or removed.

V. Hands-on Session

The authors would like for Parsafix to be featured at the hands-on session that will take place during the HoWCoM workshop in October 2021. The aim is to gain insights to answer the following research question: **How feasible is it to use Parsafix for multi-user real-time collaborative modeling across different language workbenches?** The observations made during the session will be relevant in terms of validation of the approach that Parsafix takes, as well as uncovering performance and user-experience issues in the early stages of the tool's development. The session will also give the participants a notion of Parsafix's potential.

For the purpose of the hands-on session, a public Modelix and Saros server will be deployed, along with a public version of Parsafix. Participants will receive a virtual machine image containing a version of MPS and Eclipse, with the necessary Modelix and Saros plugins installed, as well as a small generic DSL to be used during the hands-on session. The participants will receive instructions on how to connect to the Modelix or Saros server using either the MPS or Eclipse client contained in the Docker image. Once connected, they can start collaboratively modeling using the given DSL.

The modeling task performed by the participants will result in a simple model written in the use-case DSL described in Section IV. Each participant's individual task will consist of creating a model, adding new elements to the model, as well as altering and removing elements added by other participants. The candidates will walk through the following steps:

1) The candidates are assigned to one of two collaborating teams. One team will be working in JetBrains MPS and the other team will be working in Spoofax (Eclipse).

2) One team will create an initial model and will define some specific elements in it (i.e., some edit operations will be performed locally).
3) The teams are given instructions on how to connect their language workbench client to Parsafix and will be asked to begin the collaborative modeling session.
4) Once the two language workbench clients are connected to the Parsafix prototype, the teams will begin to carry out edit operations on the previously-created model in parallel. They will see the work of their collaborating parties appear in real-time and can in turn make changes to the model.
5) The session is over once each of the teams has completed their tasks to the best of their ability.

The hands-on session will also provide an opportunity to analyze the scalability of the prototype tool, both in terms of number of users and the cost Parsafix incurs on whole modeling process. Because the current implementation of Parsafix is just a prototype, it does not yet possess any advanced form of error handling. For this reason the candidates shall be instructed to limit their interaction with the prototype tool to a specific set of operations. As stated in Section III-F, operations such as defining a model in a DSL unknown to Parsafix will cause the cross-platform collaboration to halt.

VI. Conclusion

In our work, we have presented **an approach to enable real-time collaboration among engineers working on models conforming to the same DSL, but in different language workbenches**.

Parsafix is a tool-based approach for providing real-time cross-platform collaboration for users modeling using the same DSL, but in different language workbenches. It achieves this collaborative functionality by employing preexisting collaboration technologies of the workbenches and linking them together. Parsafix is essentially positioned between the technologies, posing as a client for each and passing change data from one side to the other. The edit operations are translated to and from Parsafix's internal data model in the form of abstract syntax trees. The abstract syntax trees describe the collaborative project as a whole and how the workbenches' projects relate to each other.

The current prototype of Parsafix supports real-time collaboration for JetBrains MPS and Spoofax, using the plugins Modelix and Saros, respectively.

The Parsafix prototype is implemented as a preliminary proof of concept and therefore has much room for improvement in both a scientific as well as technical sense. Future scientific research on Parsafix will focus on concretizing the generalization of its internal mechanisms, such as the mapping of different ASTs onto each another and how existing research in the field of text parsing can be exploited for the purpose of parsing textual models.

From a technical perspective, future improvements of Parsafix will focus on stabilizing the prototype tool and improving its performance, as well as implementing the aforementioned plugin system and implementing more advanced error handling to handle the many problems that can occur during the real-time synchronization of data, such as data loss and inconsistencies.

Currently, Parsafix communicates with two collaborative workbenches. Different collaborative technologies are likely to communicate with their clients in a certain specific way. This challenge can be overcome by implementing workbench-specific methods of communication. Therefore, to better serve its purpose, future versions of the Parsafix prototype will be made to support additional language workbenches and their respective collaboration technologies.

Acknowledgment

This research is funded by the Rijksdienst voor Ondernemend Nederland (RVO) through the ITEA3 BUMBLE project (18006).

References

[1] M. Fowler, "Language workbenches: The killer-app for domain specific languages?" Jun 2005. [Online]. Available: http://martinfowler.com/articles/languageWorkbench.html
[2] S. Erdweg, T. van der Storm, M. Völter, M. Boersma, R. Bosman, W. Cook, A. Gerritsen, A. Hulshout, S. Kelly, A. Loh, G. Konat, P. Molina, M. Palatnik, R. Pohjonen, E. Schindler, K. Schindler, R. Solmi, V. Vergu, E. Visser, and J. Woning, "The state of the art in language workbenches. conclusions from the language workbench challenge," vol. 8225, 10 2013.
[3] E. Vacchi, W. Cazzola, S. Pillay, and B. Combemale, "Variability support in domain-specific language development," vol. 8225, 10 2013.
[4] L. Hattori, "Enhancing collaboration of multi-developer projects with synchronous changes," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 2, 2010, pp. 377–380.
[5] P. de Lange, P. Nicolaescu, R. Klamma, and M. Jarke, "Engineering web applications using real-time collaborative modeling," 08 2017, pp. 213–228.
[6] C. Sun, S. Xia, D. Sun, D. Chen, H. Shen, and W. Cai, "Transparent adaptation of single-user applications for multi-user real-time collaboration," *ACM Trans. Comput.-Hum. Interact.*, vol. 13, no. 4, p. 531–582, Dec. 2006. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/1188816.1188821
[7] C. Jaspan, M. Jorde, A. Knight, C. Sadowski, E. Smith, C. Winter, and E. Murphy-Hill, "Advantages and disadvantages of a monolithic repository: a case study at google," 05 2018, pp. 225–234.
[8] P. Nicolaescu, M. Derntl, and R. Klamma, "Browser-based collaborative modeling in near real-time," in *9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2013, pp. 335–344.
[9] M. Völter, "Modelix and the future of language engineering," Feb 2021. [Online]. Available: https://blogs.itemis.com/en/modelix-and-the-future-of-language-engineering
[10] M. Völter and K. Solomatov, "Language modularization and composition with projectional language workbenches illustrated with mps," 01 2010.
[11] "Visual studio live share: Visual studio," May 2021. [Online]. Available: https://visualstudio.microsoft.com/services/live-share/
[12] T. Kosar, P. Martínez López, P. Barrientos, and M. Mernik, "A preliminary study on various implementation approaches of domain-specific language," *Information and Software Technology*, vol. 50, pp. 390–405, 04 2008.
[13] S. Erdweg, T. van der Storm, M. Völter, L. Tratt, R. Bosman, W. Cook, A. Gerritsen, A. Hulshout, S. Kelly, A. Loh, G. Konat, P. Molina, M. Palatnik, R. Pohjonen, E. Schindler, K. Schindler, R. Solmi, V. Vergu, E. Visser, and J. Woning, "Evaluating and comparing language workbenches: Existing results and benchmarks for the future," *Computer Languages, Systems and Structures*, vol. 44, 08 2015.

[14] J.-P. Tolvanen, "Metaedit+ for collaborative language engineering and language use (tool demo)," in *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*, ser. SLE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 41–45. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/2997364.2997379

[15] H. Ehrig, K. Ehrig, C. Ermel, F. Hermann, and G. Taentzer, "Information preserving bidirectional model transformations," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2007, pp. 72–86.

[16] G. Bergmann, I. Ráth, G. Varró, and D. Varró, "Change-driven model transformations," *Software & Systems Modeling*, vol. 11, no. 3, pp. 431–461, 2012.

[17] H. Giese and R. Wagner, "From model transformation to incremental bidirectional model synchronization," *Software & Systems Modeling*, vol. 8, no. 1, pp. 21–43, 2009.

[18] L. Kats and E. Visser, "The spoofax language workbench rules for declarative specification of languages and ides," vol. 45, 10 2010, pp. 444–463.

[19] S. Salinger, C. Oezbek, K. Beecher, and J. Schenk, "Saros: An eclipse plug-in for distributed party programming," in *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*, ser. CHASE '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 48–55. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/1833310.1833319

[20] J.-P. Tolvanen and S. Kelly, "Metaedit+: defining and using integrated domain-specific modeling languages," 10 2009, pp. 819–820.

[21] S. T, "The rascal language workbench," *Communications of The ACM - CACM*, 01 2011.

[22] L. Bettini, *Implementing Domain Specific Languages with Xtext and Xtend - Second Edition*, 2nd ed. Packt Publishing, 2016.

[23] L. Shen, X. Chen, R. Liu, H. Wang, and G. Ji, "Domain-specific language techniques for visual computing: A comprehensive study," *Archives of Computational Methods in Engineering*, 10 2020.

[24] T. Kosar, S. Bohra, and M. Mernik, "Domain-specific languages: A systematic mapping study," *Information and Software Technology*, vol. 71, 11 2015.

[25] A. Bucchiarone, J. Cabot, R. Paige, and A. Pierantonio, "Grand challenges in model-driven engineering: an analysis of the state of the research," *Software and Systems Modeling*, vol. 19, pp. 1–9, 01 2020.

[26] M. Stephan, "Emerging concepts and trends in collaborative modeling: A survey," in *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development*, ser. MODELSWARD 2019. Setubal, PRT: SCITEPRESS - Science and Technology Publications, Lda, 2019, p. 240–247. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.5220/0007255502400247

[27] M. Renger, G. Kolfschoten, and G.-J. de Vreede, "Challenges in collaborative modeling: A literature review," vol. 10, 01 2008, pp. 61–77.

[28] S. Kelly and J.-P. Tolvanen, "Collaborative creation and versioning of modeling languages with metaedit+," in *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 37–41. [Online]. Available: https://doi-org.vu-nl.idm.oclc.org/10.1145/3270112.3270132

[29] I. David, K. Aslam, S. Faridmoayer, I. Malavolta, E. Syriyani, and P. Lago, "Collaborative model-driven software engineering:a systematic update," SANER 2019.

[30] R. Baheti and H. Gill, "Cyber-physical systems," *The impact of control technology*, vol. 12, no. 1, pp. 161–166, 2011.

[31] B. Langlois, C.-E. Jitia, and E. Jouenne, "Dsl classification," 2007.

[32] G. Fischer, J. Lusiardi, and J. Wolff von Gudenberg, "Abstract syntax trees - and their role in model driven software development," in *International Conference on Software Engineering Advances (ICSEA 2007)*, 2007, pp. 38–38.

[33] E. Schindler, H. Moneva, J. van Pinxten, L. van Gool, B. van der Meulen, N. Stotz, and B. Theelen, *JetBrains MPS as Core DSL Technology for Developing Professional Digital Printers*. Cham: Springer International Publishing, 2021, pp. 53–91. [Online]. Available: https://doi.org/10.1007/978-3-030-73758-0_3