Challenges for reuse in collaborative modeling environments

Omar Alam
Trent University
Peterborough, ON, Canada
omaralam@trentu.ca

Jonathan Corley
University of West Georgia
Carrollton, GA, USA
jcorley@westga.edu

Constantin Masson and Eugene Syriani
University of Montreal
Montreal, QC, Canada
constantin.masson,syriani@iro.umontreal.ca

ABSTRACT

Software systems are not developed by individuals, and have become increasingly collaborative endeavors between stakeholders with complimentary expertise. Model-driven engineering (MDE) facilitates such collaboration by enabling developers to work on models (often domain-specific) rather than application code directly. Collaborative modeling environments help promote the integration of models. The issue of model reuse is crucial in a collaborative environment to foster the practice of modeling. Relying on a version control system as the sole means of reuse limits the usefulness of collaborative environments. This article discusses a set of challenges that must be addressed in a collaborative environment to support model reuse. Our investigation of the state of practice in existing environments shows that they do not provide sufficient support for reuse yet.

KEYWORDS

Model-driven engineering, Collaborative computing, Reusable Software, Reuse models.

1 INTRODUCTION

The development of complex software-intensive systems requires stakeholders from diverse domains to work in a coordinated manner on different aspects of the system. Often times, developers do not just communicate and exchange information among themselves, but also reuse each other's work. Various tools have emerged that serve different kinds of collaboration. For example, a version control system (VCS) is an offline collaboration tool that allows developers to reuse each other's code. A collaborator who uses a VCS does not have real-time updates on other collaborators' work. Other tools offer real-time collaboration, by making each collaborator aware of other's activity through different mediums of communication. Online editing tools (e.g., Google Docs and Microsoft Online Office) allow a collaborator to visualize the changes by others in real-time. Regardless of the specific collaboration tool, reuse of each other's work is a major outcome of collaboration.

The success of reuse in software development facilitates collaboration and coordination during software development activities, as exemplified by class libraries, services, and components. A developer can reuse the artifacts developed by other developers through the reuse interfaces of these artifacts. Modern software developers typically use VCS, such as Git repositories [1], to collaborate and track development activities throughout their projects. These systems allow developers to organize and distribute the development effort among themselves, keep track of issues and bugs, and schedule the delivery of releases. Together with VCS and repositories, advances in reuse allowed software development activities to be more organized and collaborative. For example, a group of developers can use a Git repository to collaborate in coding the classes of a software program and another developer can reuse these coded classes. However, the concern of reuse in collaborative modeling has not been a focus of study yet.

In this paper, we present a set of challenges that needs be addressed in a framework that enables reuse in collaborative modeling. Though there are emerging collaborative modeling tools, support for reuse in these tools is limited [19]. Commercial tools such as Rational Rhapsody [11], Visual Paradigm [13], MagicDraw [8], and Enterprise Architect [6] are modeling tools that are used in industry and offer collaborative support. Some MDE technologies, such as Eclipse CDO [2] and EMFStore [5], provide some support for VCS. Rocco et al. [19] and Franzago et. al. [22] provide an overview of these tools and discuss their potentials and shortcomings. They acknowledge that support for reuse and discovering reusable artifacts is limited, increasing the upfront development cost for many model-based projects. Therefore, the potential benefits of collaborative modeling in these approaches is limited. Existing collaborative environments do not provide sufficient support for reuse, and we outline the challenges that we have identified as significant to addressing this concern. The paper lists some well-known collaborative environments and tools and investigates how they address the identified challenges.

In the remainder of this paper, we first discuss the potential for reuse in collaborative modeling environments along with an example scenario exploring cases of reuse in Section 2. In Section 3, we discuss challenges for reuse in these environments. In Section 4, we examine the support for reuse in some popular modeling environments. Finally, we conclude in Section 5.

2 POTENTIAL FOR REUSE IN COLLABORATIVE ENVIRONMENTS

2.1 Model-driven engineering for collaboration

Model-driven engineering (MDE) [21] helps in reducing the gap between heterogeneous domains using principles of separation of concerns, automatic generation, and domain-specific languages (DSL). In MDE, stakeholders work on models in order to design, transform, simulate, and analyze systems. MDE advocates using the most appropriate modeling formalism that expresses the relevant properties of the system under development at each level of

abstraction, for a given stakeholder group. A formalism used at the requirement level for scientists is different from the formalism used at the design level for developers. Through model transformations, models of higher abstraction levels are integrated with lower-level models that are closer to the solution space, such as algorithms, data structures, networking, etc. This process continues until an executable model (which can be code) is generated [24].

MDE is a potential solution to help develop systems collaboratively [31]. Stakeholders from diverse backgrounds who work on the system under development using different notations can collaborate and reuse each other's work. Furthermore, in a complex system, models can quickly grow in size, deeming the efforts of a single modeler insufficient to maintain and evolve the system. In such projects, collaboration is a necessity to cope with the growing size of models. Therefore, there is a need for collaborative platforms that allow teams of stakeholders with varying expertise to work together to produce a coherent and complete system [17]. In particular, there is a need for collaborative environments that support different modeling formalisms and allow stakeholders to reuse each other's models.

2.2 Reuse in Collaborative Environments

In general programming language (GPL) environments, reuse of libraries and components is common practice. Most programs are created by mixing several existing libraries ranging from standard libraries to custom modules created in-house. Powerful collaborative platforms, such as Github, and specific language features, such as polymorphism, facilitate code reuse in those environments. Most software developers rely on VCS, libraries, or repositories for collaboration. Unfortunately, this is not the norm in MDE projects.

Though some new environments are supporting more complex sharing systems (e.g., GenMyModel [7]), ease and efficiency of reuse is still far from the programming equivalent. For example, VCS is mostly used to track and collaborate in the development of non-reusable models [19]. Unlike VCS repositories for GPL environments, model reuse in repositories through VCS is still a challenge in MDE [26]. Enforcing consistent reuse is necessary to cope with the growing complexity of software systems. In MDE, modelers typically create models from scratch because modeling languages offer limited support to reuse existing models and modeling environments tend not to ship with any reusable models. For example, when a modeler wants to implement a new DSL, it is common to build it from scratch instead of reusing existing language artifacts [22]. Lack of support for reuse limits the potential of modeling environments in MDE, as checking out, committing, and updating models that cannot be reused will not be very useful for collaborative environments.

When collaborating, modelers may work on the same artifact, different parts of the same artifact or distinct artifacts that are part of the whole system [18]. Modelers may need to reuse each other's models, or reuse models that are external to the project, i.e., imported from a different project. Support for reuse in collaborative MDE can be facilitated with reuse mechanisms in modeling languages and their environments. In this paper, we focus only on environmental concerns and support, describing challenges and summarizing the current state of practice of collaborative MDE environments. Supporting reuse in modeling languages is critical to supporting reuse of models but is not the within the scope of this paper.

2.3 Modeling Reusable Components of a Modern Vehicle

To better understand how modelers collaborate to reuse each other's model, we provide an example of reusing components of a modern vehicle. We refer to this example to explain the challenges for supporting reuse in Section 3. Fig. 1 illustrates an example of collaboration in a vehicle manufacturing project. Initially, two modelers (Modeler 1 and 2) were working on the project. Modeler 1 developed the self-parking feature (version 1) to support parallel parking, while Modeler2 worked on a model of the car that uses that feature. At some later time, Modelers 1 and 3 collaborated directly on extending the self-parking feature to support perpendicular parking as well. Modeler 2 used version 2 of self-parking in Car 2, but only for parallel parking support. Eventually, Modeler 4 wanted to reuse all of the self-parking feature version 2 in Car 3. All models (components and cars) are assumed to be stored in some available repository.

As stated in Section 2.2, we focus on environmental support for reuse in collaborative MDE, e.g., through storage or search mechanisms. It is certainly possible that the modeling language used when developing the self-parking component has support for reuse, e.g., through reuse interfaces [25, 30]. However, such reuse mechanisms are outside the scope of this paper.

3 CHALLENGES AND SUPPORT FOR REUSE

As we consider the potential of collaborative environments, we first explore the various challenges to reuse and examine how collaborative environments might be able to address or mitigate these challenges. We identified these challenges based on our experiences in building and using collaborative modeling environments [14, 32]. This list also builds on the requirements for collaborative environments presented in [28].

3.1 Supporting Communication

As we seek to support collaborative development environments, a primary consideration is communication between developers. We consider here two types of communication, direct and indirect communication between developers who reuse the same models.

3.1.1 Direct Communication. The modeling environment can support direct communication from one developer to another in two ways. First, the environment can provide synchronous (e.g., text, voice, or video chat options) or asynchronous (e.g., messaging systems) communication facilities. These tools have obvious benefits, but are not essential to be provided within the environment, as external solutions are plentiful (e.g., Slack [12]). However, a more essential concern within the environment is the concept of authorship. Environments, such as VCS provide facilities to track the contributors to a project. This enables other developers to identify those who are actively working on a system or who have contributed in the past. In other words, the subject of discussion (i.e., the artifact) is defined and traceable in discussion threads. If

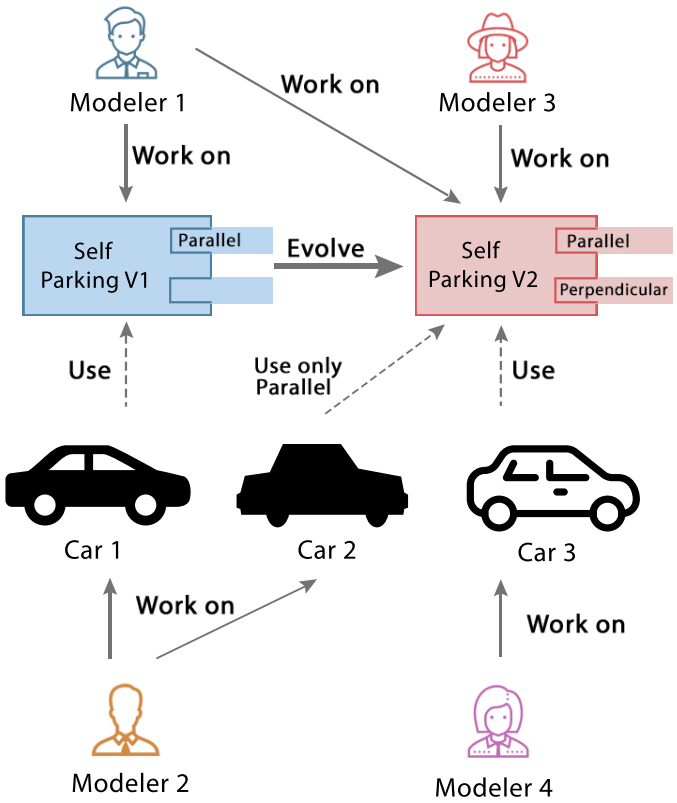

Figure 1: Example for Collaboration in a Vehicle Manufacturing Project

modelers use an external communication solution such as Slack, it will be difficult to point out which artifact is being discussed, generating confusion among collaborating modelers. Collaborative environments could capture similar authorship information providing the facility to identify the contributions of other collaborators; i.e., track the authors of a given model as well as there contributions over time. Thereby, authorship knowledge within the system can be used to guide collaborators direct communication. Considering the example from Section 2.3, Modeler 3 might identify the existing Self Parking model, but require adding the additional parking feature for their own use cases. If the environment supports tracking authorship, Modeler 3 could then identify Modeler 1 as the most appropriate individual to contact, and request a direct collaboration to update the Self Parking model with the new features.

3.1.2 Indirect Communication. Indirect communication encompasses the practices of documentation including the use of comments and external sources, such as wikis. This communication informs future modelers of basic concerns, such as structure and use of the software artifacts. It can also convey less direct details, such as intent and rationale for implementation decisions. This information is essential to guide new collaborators through the more mundane details of how the system is used. Although it may not be essential for modelers experienced with the systems, the documentation of these basic concerns facilitates new collaborators, and eliminates often non-trivial barriers to reuse. A software system designed in Java would not be considered complete without javadocs describing the various interfaces. Yet, a model might be offered without any concern for describing the use and purposes of that model. Thus, the design decisions made when constructing the models is completely lost, and new collaborators may violate these decisions when seeking to extend and reuse existing models. Consider the example scenario detailed in Section 2.3. As Modelers 2 and 4 are working on the car, they must understand the usage and limitations of the self-parking model being reused. In the future, Modeler2 might even work on evolving the Car1 model to an updated version of that vehicle. Having the documentation of the two versions will help support the decision to either use V2 of self-parking or stay with V1. If the V2 of the self-parking model assumes more accurate control of the braking system than V1, it may not be an option for some vehicles. However, this information could be very difficult to identify without some sort of documentation declaring it.

The use of comments in general-purpose languages such as Java has proved to be a great benefit. This practice could easily be supported within collaborative modeling environments. Although comments are important components of a specific modeling language, we think they should be supported generically at the environment level. Much like a view can be maintained separate from the underlying modeled elements, a comment could be linked to those elements without needing to disrupt the focus and intent of the modeling formalism. The value of the environment providing this

generic concern is particularly noteworthy when considering the prevalence of domain-specific languages (DSLs). Support at the environment level, would de-facto offer support to all DSLs within the environment.

3.2 Versioning of Models

VCS are tools that trace all changes for a system and have facilities for managing the history of these changes. Ideally, all basic VCS features available in a GPL environment may be present in collaborative MDE environments.

Branching is widely used in VCS and requires support for branch merging. The strong semantics of models add a layer of complexity: a simple textual comparison of their serialized format (in XMI) may be insufficient in order to detect all conflicts. Instead, syntactic and semantic similarity tests would require a more complex process to detect conflicts. In GPL environments, projects often reuse a specific version of an external dependency. Since dependencies themselves make use of version control, we may have a direct link to the specific version of the dependency. For instance, a CMake build system can download and compile a specific tag from a branch of the required dependency. Though tools like GenMyModel or WebGME [27] introduce model repositories with VCS support, linking to specific versions is not currently possible. Linking to an external model is not enough. A system to select specific version, tag, and branch is required. Consider Fig. 1: Car 1 uses self-parking version 1, which has now evolved to version 2. Without a proper tag system and support for versioning, Car 1 would be automatically updated to use the self-parking version 2 feature with unforeseen consequences from the forced shift. In this way, indirect collaborators reusing components from an ongoing project can manage the migration to newer versions of their dependencies on their own timetable.

GPL developers find the ability to reuse the same language elements across a wide variety of development environments, but similar interoperability has long been a source of frustration for modelers. Modern support for XMI formats presents a step in the right direction. However, when considering the prominence of domain-specific models, complete support for tool interoperability is a daunting challenge at best. However, modern trends in developing new languages that compile to a common base (e.g., CoffeeScript [3]) could present a way forward.

3.3 Model Repositories

Nowadays, software developers extensively use repositories, such as Github, to collaborate and track development activities throughout their projects. Coupled with version control facilities, these systems allow developers to organize and distribute the development effort among themselves, keep track of issues and bugs, and schedule the delivery of releases, as well as reuse each other's code. To facilitate collaboration between modelers, a repository should provide a storage facility that is equipped with VCS features (e.g., commit - update versioned history, browse - manually explore models). Modelers can use these features to reuse each other's models, document the changes made to the models, and schedule their activities. When working with models in a project, the storage facility should use the meta-information about the models to organize them and to record the relationships that may exist between them. The collaboration scenario in Fig. 1 would be possible when Modeler1 can store the self-parking component in a repository that has VCS facilities, to allow Modeler 3 to collaborate and introduce the second version for that component after making her updates.

3.4 Searching for Models

One of the powerful features of VCS is the ability to search artifacts in the repository. The search activity identifies relevant models based on their metadata: VCS allows searching elements based on their change metadata. The search feature is important for collaboration and reuse of model elements. In Fig. 1, Modelers 2 and 4 would be able to search and find the component based on its meta-information and reuse it. In addition to VCS, Integrated Development Environments (IDEs) allow programmers to search for code snippets in large code bases or libraries. Collaborative modeling environments will benefit from advanced search capabilities of programming IDEs to reuse from potentially large number of projects. Searching can be performed at different levels of granularity: searching for whole models, parts of the model, actions performed on models, authors of model manipulations. Searching can be performed by browsing through a dynamic list of artifacts or by querying the repository. It is important that the result of the search is provided quickly for effective collaboration.

3.5 Granularity of Model Reuse

There are different scenarios that motivate the reuse of a model [18]. The most trivial reuse mechanism is copying a model from one project to another. In this case, the copied model and the original evolve separately which may lead to inconsistency. A model can be imported to a local project as a reference. In this case, the imported model is a proxy of the original and changes in the latter are transposed to the former. When reusing a model, it is possible to reuse the whole model, as a black box, in which case the reusing entity communicates with a dedicated interface of the reused model. For example, in Fig. 1, Car 1 reuses self-parking version 1 as a whole.

Alternatively, in a white box approach, one model may reuse specific content from another model, as it is the case with Car 2 and Car 3. The model can expose all elements, the data they encapsulate, or the relationships between them. Further information may be hidden with the use of views encapsulating parts of a model (sub-model or aggregation) [18]. Like when reusing a library, the dependency of a reused model must be meticulously analyzed.

3.6 Relationships between Models

To fully reap the benefits of repositories, the stored models should define relationships among them. At the modeling language level, models should define interfaces that facilitate reuse. Reuse in software engineering is enabled by modular interfaces between artifacts [25]. Having an explicit model interface makes it possible to apply proper information hiding principles [29] by concealing internal design details of the model from the rest of the application.

Unlike programmers, modelers often create models from scratch due to lack of model interfaces that facilitate reuse. Popular reuse units (e.g., code, components, and services) provide interfaces that

Tool	Comm.	Versioning	Repository	Search	Reuse granularity	Relationships
WebGME [27]	None	History, Branch	Storage, Tracking	Browse elements	None	None
MDEForge [9]	Indirect	History	Storage	Browse models	None	Yes
MetaEdit+ [23]	None	History, Branch	Storage	Query elements	None	None
OBEO [10]	None	History	Storage, Tracking	None	View	None
AToMPM [32]	Direct	None	Storage	Browse models	Model, View, Element	None
GenMyModel [7]	Indirect	History, Tags	Storage, Tracking	Query elements	Model, Element	None
CDO [2]	None	History, Branch	Storage, Tracking	None	None	None
Visual Paradigm [13]	Indirect	History	Storage, Tracking	Browse elements	Model, Element	None
EMFStore [5]	None	History, Branch	Storage, Tracking	None	None	None
EMFCollab [4]	None	History,Branch	Storage, Tracking	None	None	None

Table 1: Support for reuse in popular collaborative modeling environments

enable selecting/choosing the reused unit, coordinated customization of the unit and using the customized unit in the context of it reusing artifact. These interfaces are important to specify what structures and behavior the reused model provides to the reusing model, and how to adapt reused model to the specific needs of the reusing model [15].

Furthermore, the modeling environment should use the relationships between models to support traceability. Modelers should be able to trace models that are involved in their project to understand their functionality, read their documentation, and access their development history. Models can be traced based on the relationships defined by the modeling language as discussed above, as well as relationships that are defined by the environment. If there is a modeling environment that supports multiple modeling languages, it can define relationships between related models that serve the same purpose as in the case of megamodels that define relationships between different kinds of models [20]. Relationships in a megamodel can range from input/output of model transformations, global constraints across models, to conformance between a model and a type model.

For example, if a modeling environment allows modeling requirements and design models, and the modelers want to create models for security, they can define traceability links between security requirement and design models.

Finally, modelers may visualize the traces and relationships, whether they are defined by the language or by the environment. This enables them to get the big picture of the project, visualize changes made to related models, as well as plan and coordinate future developments based on those changes.

In Fig. 1, Modeler 3 would need traceability support to update the self-parking component. She would need to trace back and visualize her changes to version 1 of the component, and this trace between the versions would be useful to Modeler 2 when moving from one version to the next. In addition, model interfaces of the car models are necessary to allow Modelers 2 and 4 to reuse the self-parking component and map its elements to elements in the car models.

4 SUPPORT FOR REUSE IN POPULAR MODELING ENVIRONMENTS

Table 1 overviews the support for reuse in 10 existing popular collaborative modeling environments. We selected the tools based on the set compared in [28] and surveyed in [19]. We summarize how each of these ten tools addresses the challenges described previously. As mentioned earlier, we only investigate whether the modeling environment address the challenges. It is possible that some of the challenges are addressed at the modeling language level, which is not the focus of this paper.

4.1 Reuse dimensions

We explain the categories and tags used in Table 1.

Communication category describes support for communication within the environment.

- Direct - environment supports direct communication (e.g., messaging)
- Indirect - environment supports documenting the models at some level, including documenting use, intention, and/or authorship
- None - environment provides no clear support for communication

Versioning category describes support for managing versions of models.

- History - environment provides support for maintaining a history of changes made to a model
- Branch - environment provides support for managing/merging distinct branches of a model
- Tags - environment provides support for identifying significant versions of a model for reuse
- None - environment provides no clear support for managing versions of models

Repository category describes support for storage, retrieval, or tracking of models.

- Storage - environment provides support for storing models
- Tracking - environment provides support for tracking features/bugs for models
- None - environment provides no clear support for storage, retrieval, or tracking of models

Search category describes support for searching models. The tags can be any combination of the following.

- Browse - environment provides support to browse artifacts
- Query - environment provides support to query artifacts
- Models - Search is performed on whole models
- Elements - Search is performed on parts of models or their elements
- None - environment does not provide support for searching

Granularity category describes the granularity of model that may be reused in the environment (this category considers only environment support and not language level support).

- Model - environment provides support for reusing models as an atomic unit
- View - environment provides support for reusing views of a model(s) as an atomic unit
- Element - environment provides support for reusing an element of a model as an atomic unit
- None - environment provides no clear support for reusing existing models, views, or elements in new models

Relationships - category describes support for analyzing relationships of models to support, for example, understanding of existing systems or impact of proposed changes.

- None - there are no such relationships
- Yes - there is support for such relationships

4.2 Discussion

Communication: AToMPM supports direct communication via a chat system. MDEForge and GenMyModel support indirect communication through documentation (though not comments). Additionally, some modeling languages support the concept of comments within the language, but we feel this should be an environment concern and not contained within the language.

Versioning: History is usually an ordered list of changes. Tools like WebGME, EMFStore and EMFCollab add a Git-like feature, like commit hash for each change and a branching system with merging. Tools, such as MetaEdit+, rely on an external VCS in the backend to implement these features. However, history is generally used only for error recovery inside the project (i.e., revert back to a working version). We have not identified a tool that supports linking to an older version of a model for reuse. GenMyModel does have a tag system to identify a specific version, but this feature seems to be intended only for internal use.

Repository: All tools we reviewed have some form of storage system, but some method of discovering new models (Search in Table 1) is not always present. Tools like WebGME, EMFStore, EMFCollab and GenMyModel offer a Github-like repository that allows users, among others, to browse models and see other users. Other tools, such as, OBEO and AToMPM use folder path structure. Several environments support tracking features and bugs in models developed within the environment.

Search: MetaEdit+ and GenMyModel provide the most advanced searching mechanism as they allow to query the repository to find model elements that match the query expression. The query is based on the information provided by the metamodel (e.g., type names or meta-language name). WebGME and Visual Paradigm allow to browse for models, but also model elements, whereas MDEForge and AToMPM only allow the user to browse through model file names. OBEO, CDO, EMFStore and EMFCollab do not directly support searching, however they rely on the Eclipse IDE for that functionality.

Granularity of Reuse: AToMPM, Visual Paradigm, GenMyModel, and OBEO provide explicit environmental support for reuse. Visual Paradigm and GenMyModel enables linking between UML formalisms, e.g., linking objects in sequence diagrams to classes defined in a class diagram.

AToMPM and OBEO are the only tools where specific views of a large model can be created and reused.

Relationships: Most environments could support some forms of traceability (especially those with versioning support), but higher level analysis (e.g., tracing dependencies to understand the impact of proposed changes) is not clearly defined by the environments. Furthermore, while most environments could support either heterogeneous models (e.g., via megamodeling) or linking through generative transformations (e.g., converting platform-independent models to platform specific models), most environments provide little or no support to analyze and manage these relationships. MDEForge is the only exception as it provides support for megamodel-based relationships between artifacts and allows modelers to track those relationships.

5 CONCLUSION

Building a software system is typically a collaborative endeavour. Different stakeholders, from potentially diverse domains, coordinate with each other to fulfill the goals of the software project. Reuse is a key facet of software system development that increases both productivity and maintainability, as well as reducing both production cost and time-to-market. Increasingly, software practitioners rely on collaboration tools, such as Github, that allow them to both document and manage their own work as well as finding and reusing each other's work.

In MDE, developers collaborate to create, update, and analyze models of a system under development. However, despite the success stories in GPLs as exemplified by the proliferation of reuse through libraries, services, and components; model reuse is a challenge in MDE. Over a decade ago, Bran Selic stated clearly that mature tool support is necessary to support adoption and growth of MDE [16]. In this paper, we identified challenges facing reuse in collaborative MDE environments and investigated the support provided by existing environments. Our study of the state of practice in existing environments shows that their support for reuse is not sufficient. We believe that the identified challenges in this paper will help guide the development and improvement of modeling tools to provide better support for reuse. In particular, we hope the categorization demonstrated on the 10 identified environments in Section 4 will prove useful in analyzing MDE tools for support of these challenges. As we look forward to the future of MDE, we see a clear place for collaborative modeling environments as well as a great need to support reuse in those environments.

REFERENCES

[1] Last accessed: 2017. Git. (Last accessed: 2017). https://git-scm.com/

[2] Last accessed: 2018. CDO Model Repository. (Last accessed: 2018). http://www.eclipse.org/cdo/.

[3] Last accessed 2018. CoffeeScript. (Last accessed 2018). https://coffeescript.org/.
[4] Last accessed: 2018. EMFCollab. (Last accessed: 2018). http://qgears.com/products/emfcollab/.
[5] Last accessed: 2018. EMFStore. (Last accessed: 2018). http://www.eclipse.org/emfstore/.
[6] Last accessed: 2018. Enterprise Architect. (Last accessed: 2018). http://www.sparxsystems.com/products/ea/.
[7] Last accessed 2018. GenMyModel. (Last accessed 2018). https://www.genmymodel.com/.
[8] Last accessed: 2018. MagicDraw. (Last accessed: 2018). https://www.nomagic.com/.
[9] Last accessed 2018. MDEForge. (Last accessed 2018). http://www.mdeforge.org/.
[10] Last accessed: 2018. OBEO Designer. (Last accessed: 2018). https://www.obeodesigner.com/.
[11] Last accessed: 2018. Rational Rhapsody Designer Manager. (Last accessed: 2018). www-03.ibm.com/software/products/en/ibmratirhapdesimana.
[12] Last accessed: 2018. Slack. (Last accessed: 2018). https://slack.com/
[13] Last accessed: 2018. Visual Paradigm. (Last accessed: 2018). https://www.visual-paradigm.com/.
[14] Omar Alam. 2016. *Concern-Oriented Reuse: A Software Reuse Paradigm.* Ph.D. Dissertation. McGill University.
[15] Omar Alam, Jörg Kienzle, and Gunter Mussbacher. 2013. Concern-Oriented Software Design. In *Model-Driven Engineering Languages and Systems - 16th International Conference, MODELS 2013, Miami, FL, USA, September 29 - October 4, 2013. Proceedings*. 604–621.
[16] Bran Selic. 2003. The Pragmatics of Model-Driven Development. *IEEE Software* 20, 5 (2003), 19–25.
[17] Benoit Combemale, Julien DeAntoni, Benoit Baudry, Robert B. France, Jean-Marc Jézéquel, and Jeff Gray. 2014. Globalizing Modeling Languages. *Computer* 47, 6 (2014), 68–71.
[18] Jonathan Corley, Eugene Syriani, Huseyin Ergin, and Simon Van Mierlo. 2016. *Modern Software Engineering Methodologies for Mobile and Cloud Environments*. Number 7. IGI Global, Book section Cloud-based Multi-View Modeling Environments, 120–139.
[19] Juri di Rocco, Davide di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. 2015. Collaborative Repositories in Model-Driven Engineering. *IEEE Software* 32, 3 (2015), 28–34.
[20] Jean-Marie Favre. 2006. Megamodelling and Etymology. In *Transformation Techniques in Software Engineering (Dagstuhl Seminar Proceedings)*, Vol. 05161.
[21] Robert France and Bernhard Rumpe. 2007. Model-driven Development of Complex Software: A Research Roadmap. In *Future of Software Engineering (FOSE '07)*. IEEE, 37–54.
[22] M. Franzago, D. D. Ruscio, I. Malavolta, and H. Muccini. 2018. Collaborative Model-Driven Software Engineering: a Classification Framework and a Research Map. *IEEE Transactions on Software Engineering* (2018). DOI : http://dx.doi.org/10.1109/TSE.2017.2755039
[23] Steven Kelly, Kalle Lyytinen, and Matti Rossi. 1996. MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. In *CAiSE*. Springer-Verlag, London, UK, UK, 1–21.
[24] Steven Kelly and Juha-Pekka Tolvanen. 2008. *Domain-Specific Modeling: Enabling Full Code Generation.* John Wiley & Sons.
[25] Jörg Kienzle, Gunter Mussbacher, Omar Alam, Matthias Schöttle, Nicolas Belloir, Philippe Collet, Benoît Combemale, Julien DeAntoni, Jacques Klein, and Bernhard Rumpe. 2016. VCU: The Three Dimensions of Reuse. In *Software Reuse: Bridging with Social-Awareness (ICSR)*. 122–137.
[26] Dimitrios S. Kolovos, Louis M. Rose, Nicholas Matragkas, Richard F. Paige, Esther Guerra, Jesús Sánchez Cuadrado, Juan De Lara, István Ráth, Dániel Varró, Massimo Tisi, and Jordi Cabot. 2013. A Research Roadmap Towards Achieving Scalability in Model Driven Engineering. In *BigMDE Workshop*. ACM, 2:1–2:10.
[27] Miklós Maróti, Tamás Kecskés, Róbert Kereskényi, Brian Broll, Péter Völgyesi, László Jurácz, Tihamer Levendovszky, and Ákos Lédeczi. 2014. Next Generation (Meta)Modeling: Web- and Cloud-based Collaborative Tool Infrastructure. In *Proceedings of the 8th Workshop on Multi-Paradigm Modeling*. 41–60.
[28] Constantin Masson, Jonathan Corley, and Eugene Syriani. 2017. Feature Model for Collaborative Modeling Environments. In *International Workshop on Collaborative Modelling in MDE*, Vol. 2019. CEUR-WS.org, 164–173.
[29] David Lorge Parnas. 1972. On the Criteria to Be Used in Decomposing Systems into Modules. *Commun. ACM* 15, 12 (1972), 1053–1058.
[30] Daniel Strüber, Stefan Jurack, Tim Schäfer, Stefan Schulz, and Gabriele Taentzer. 2016. Managing Model and Meta-Model Components with Export and Import Interfaces. In *Proceedings of the 4rd Workshop on Scalable Model Driven Engineering part of the Software Technologies: Applications and Foundations (STAF 2016) federation of conferences, Vienna, Austria, July 8, 2016*. 31–36.
[31] Eugene Syriani. 2016. Framework to Model Collaboratively. In *International Workshop on Collaborative Modelling in MDE*, Vol. 1717. CEUR-WS.org, 4.
[32] Eugene Syriani, Hans Vangheluwe, Raphael Mannadiar, Conner Hansen, Simon Van Mierlo, and Hüseyin Ergin. 2013. AToMPM: A Web-based Modeling Environment. In *MODELS 2013 Demonstration*. 21–25.