IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 12 April 2023, accepted 6 May 2023, date of publication 16 May 2023, date of current version 24 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3276872

RESEARCH ARTICLE

Cross-Platform Real-Time Collaborative Modeling: An Architecture and a Prototype Implementation via EMF.Cloud

KOUSAR ASLAM, (Member, IEEE), YU CHEN, MUHAMMAD BUTT, AND IVANO MALAVOLTA, (Member, IEEE)
Department of Computer Science, Vrije Universiteit Amsterdam, 1081 HV Amsterdam, The Netherlands
Corresponding author: Kousar Aslam (k.aslam@vu.nl)

This work was supported in part by the Rijksdienst voor Ondernemend Nederland (RVO) through the ITEA3 BUMBLE Project 18006.

ABSTRACT Real-time collaboration in model-driven software engineering is gaining increasing attention from both the research and industrial community. This is due to its potential adverse effects on the efficiency of software modeling process. However, current approaches for real-time collaboration are tightly coupled to modeling platforms and language workbenches. To address this issue, we present BUMBLE-CE, the first extensible approach for cross-platform real-time collaborative modeling which is independent of both the modeling platforms and the domain-specific modeling language used by the modelers. One of the main characteristics of BUMBLE-CE is that it allows modelers to work on the head revision of their models as usual and, when necessary, they can start and terminate on-demand real-time collaborative modeling sessions. This paper reports on the requirements driving the design of BUMBLE-CE, its architecture and underlying design decisions, implementation of BUMBLE-CE using EMF.Cloud technologies and an example application of BUMBLE-CE to state machine models realized in Eclipse EMF and Jetbrains MPS.

INDEX TERMS Collaborative modeling, eclipse EMF, jetbrains MPS, model-driven software engineering.

I. INTRODUCTION

Model-Driven Software Engineering (MDSE) uses models as the main artefacts during software design and development, thus increasing the abstraction level of the software development process [1]. MDSE-based software is therefore expected to be easier to understand, to facilitate better communication among software engineers, and to improve software maintenance [2]. Similarly, with the growing trend of geographically distributed teams, adoption of agile methodologies, and open-source development, the need for increased collaboration within software development teams is more and more pressing [3]. Combining the positives of both MDSE and Collaborative software engineering, *Collaborative MDSE* deals with the development of methods and tools to increase communication and collaboration among technical and non-technical stakeholders working together on large models for building complex software systems [4].

Collaborative MDSE is gaining popularity in academia [5], [6] and industry [7]. This is due to the fact that the arduous and complex task of designing models for complex systems can benefit from the brain power of more than one modeler, in many cases involving different expertise in different disciplines [8]. Various modeling platforms are available for defining and designing models, such as the Eclipse Modeling Framework [9], Jetbrains MPS [10], MetaEdit+ [11]. The choice of modeling platform can depend on either the chosen domain-specific modeling language (DSML) or organization-specific constraints and preferences. While working on these modelling platforms, modelers can collaborate either in an offline mode (*i.e.,* asynchronous collaboration [5]) or a real-time one (*i.e.,* synchronous collaboration [5]). In offline mode, collaborators check out a modeling artefact from a version control system (VCS), such as Git, and commit local changes to the repository asynchronously.

The associate editor coordinating the review of this manuscript and approving it for publication was Claudia Raibulet.

This method is effective, though not fast 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 [12]. In real-time collaboration, modelers can simultaneously edit a model and these changes are immediately propagated to all other modelers (similarly to what Google Docs does). The inefficiency of offline collaboration is eliminated by real-time collaboration through automatic synchronization of model edit operations between modelers [13]. We focus on real-time collaboration in this work.

Several modeling platforms being used nowadays are integrated with real-time collaboration tools. Examples of such technologies are Modelix [14] for JetBrains MPS [15], Spoofax [16] for Eclipse, and Visual Studio Live Share for Visual Studio Code's modeling SDK [17]. The need for real-time collaboration in the context of modeling across *different* platforms stems from the same reason as discussed above - there are several modeling platforms available to choose from. Currently less attention is being paid to enable *cross-platform real-time collaboration*. At the time of writing, the MDSE research community is primarily interested on intra-platform collaboration [6].

In our work, we aim to fill this research gap by presenting **a novel generic architecture and its prototype realization for real-time collaboration which is independent of both the modeling platforms and the domain-specific languages used by the modelers**. The architecture has been designed with the aim to provide flexibility to the modelers to enable real-time collaboration within their preferred editors with minimal effort. This is achieved via a set of extension points for external plugins where third-party developers can extend the default real-time collaboration mechanisms with their own business logic (*e.g.,* by implementing their own model validators, resolvers for tracing model elements across modeling environments, and more). Our approach allows modelers to work on different platforms by allowing them to collaborate on the same model represented by different concrete syntaxes, yet conforming to the same metamodel. This is different from multi-view modeling which refers to decomposing the models into multiple views, that are concerned with specific aspects of the whole system. Each of these views is generally represented using completely different modeling languages, each with their own abstract syntax [18]. The work has been done as part of the BUMBLE ITEA3 project [19]. BUMBLE is a European research project centered around blended modeling [20] and collaborative modeling [5]. As such, we name our collaboration engine as BUMBLE-Collaboration Engine (BUMBLE-CE). In brief, the main contributions of this study are the following:

1) Elaboration of the main requirements for cross-platform real-time collaborative modeling engines.
2) The description of the software architecture and key design decisions for building our extensible cross-platform real-time collaborative modeling engine.
3) The prototype implementation of the above-mentioned architecture using EMF.Cloud.
4) The description of an example application of the prototype implementation of BUMBLE-CE, in the context of a simple DSML for modeling state machines.

It is to be mentioned that in this work we focus on designing and implementing BUMBLE-CE to fulfill the requirements elicited in Section IV, not on providing an evaluation or conducting a case study to assess BUMBLE-CE. In the future, we will perform extensive evaluation, both in academic and industrial contexts.

The **target audience** of this study is composed of researchers and practitioners. Specifically, with this study we provide evidence that collaborative modeling that is both real-time and cross-platform is feasible, thus providing a foundation for researchers active in the collaborative modeling research area; researchers can reuse/expand our generic architecture for carrying out their own research, *e.g.,* by (i) implementing (and empirically evaluating) their own algorithms for change propagation across different modeling platforms, (ii) integrating their own cross-platform tracing mechanisms, (iii) studying how different concrete syntaxes for the same DSML can be suitably integrated in a real-time collaboration network, or (iv) studying how synchronous and asynchronous collaborative modeling paradigms can be suitably integrated. Practitioners, specially tool vendors, can (i) use our generic architecture as a blueprint for implementing their own collaborative modeling engines or (ii) expand how EMF.cloud-based prototype into a production-ready collaboration engine and use it in their own industrial projects.

The rest of the paper is organized as follows: Section II explains the essential concept of real-time collaboration and introduces the technologies needed to understand the paper; Section III draws a comparison between our working and existing approaches/tools for real-time collaboration; Section IV presents the requirements elicited from academic and industrial partners for BUMBLE-CE; Section V explains the architecture and underlying design decisions for BUMBLE-CE; Section VI describes prototype implementation of BUMBLE-CE; an example application of our collaboration engine is presented in Section VII. A reflection on limitations and areas of improvement for BUMBLE-CE is provided in Section VIII and finally we conclude in Section IX. We suggest readers interested in the research context of the work to focus on Sections II and III, the readers interested in the principles and design decisions for BUMBLE-CE to focus on Sections IV and V, while the readers interested in technical aspects of BUMBLE-CE to focus on Sections VI and VII.

II. BACKGROUND

In this section, we provide the basic concepts for understanding the remainder of this paper. Specifically, we first provide a definition for real-time collaboration and collaborative modeling (Section II-A); then, we present the technologies

used for implementing our prototype (EMF and EMF.Cloud, Section II-B); finally we present Jetbrains MPS since it is a relevant part of our example application on the DSML for modeling state machines (Section II-C).

A. REAL-TIME COLLABORATION

Real-time collaboration is defined as the collaborative editing of data by multiple modelers over a network, in real-time [21]. A well-known example of this type of real-time collaboration is the collaboration functionality offered by *Google Docs*, where multiple modelers can work on a single document at once, seeing the input of others appear as it is typed [22]. In this context, real-time refers to be within a time frame that is considered sufficiently small for acceptable timeliness [23]. Instantaneous communication over a network is physically not possible, due to factors such as travel and processing time, which is why there will always be a delay involved to some degree. However, as Google Docs exemplifies, when operating optimally, modern technology allows for collaboration that appears near-instantaneous to the collaborating modelers.

B. ECLIPSE EMF AND EMF.CLOUD

The Eclipse Modeling Framework (EMF) is a popular development environment comprising a set of Eclipse plugins that provide tools for MDSE in Eclipse [24]. EMF is an implementation of the well-known MOF four-layers metamodeling stack [25]. The MOF stack is composed of the following layers: M0 (Modeled System) layer represents the software system to be modeled, M1 (Model) represents a model edited by a modeler, M2 (Metamodel) defines the modelling language used to create the model in M1, and M3 (Metametamodel) defines a language for the definition of modeling languages. In essence, the metamodel describes the structure of the model and a model is a concrete instance of this metamodel. Modelers are able to define metamodels in EMF using the Ecore metamodeling language. Ecore is an implementation of the Essential MOF (EMOF) language, which is a subset of MOF [26]. A minimal set of elements required to specify metamodels is provided by EMOF. EMF models are programmatically manipulable through a Java-based API, and EMF offers a converter for serializing and deserializing models to/from XMI.

EMF.cloud [27] is the umbrella project for components and technologies that make the EMF and its features accessible on the web and in the cloud. The current EMF implementation natively supports loading, model manipulation, and serialization. To connect web clients, the EMF.cloud model server builds a foundation on top of current technologies. It can control the state of loaded models in a shared editing domain at run-time. It permits applying modifications using a command pattern and registering for modifications. It offers a REST API that supports various formats for models and model edits (*e.g.,* JSON and XMI) [28]. The features listed above make EMF.Cloud a good candidate for our prototype implementation of BUMBLE-CE.

C. JETBRAINS MPS

The MPS IDE[1] is an open-source language workbench developed by JetBrains [10]. It is used within as well as outside of JetBrains to aid in the development of a variety of modeling tools. MPS makes use of *projectional editing*. This means that users of a language defined within MPS edit the language elements of a model in a fixed layout determined by the language developers. Behind the scenes, MPS is keeping track of a data model that describes all aspects of the model being edited. The projection that the modeler sees is generated from this data model. A projection within MPS can be textual, symbolic and/or tabular. This list can be expanded further upon through the implementation or use of plugins (*e.g.,* for providing graphical editors). One of the most recurrently-proclaimed features of projectional editing is its user-friendliness, as it does not require the modelers to edit any source code and largely eliminates the possibility for modelers to make syntactic errors while working on the models.

III. RELATED WORK

Real-time collaboration has been identified as a popular research topic [5], [6], aligning well with its 95% industry need [7]. Several approaches and methods have been developed to deal with the challenging task of facilitating modelers to make changes to the same model simultaneously. Table 1 presents existing popular tools/approaches for real-time collaborative modeling. We classify the identified tools along the following features: *Technical Space,* specifies technology that the tool is being built on, such as Eclipse, Jetbrains MPS, or any other custom framework; *Cross-platform*, whether the tool is able to incorporate another modeling platform as a collaborating party; *Change propagation,* whether the tool provides flexibility to the user to choose a particular mechanism for propagation of changes during real-time collaboration session; *Language independent,* does the tool allows modelers to collaborate on model developed in any modeling language or only supports a specific language; *Conflict resolution,* whether the tool provides some mechanism to resolve the conflicts occurring during collaboration; and finally the *Workspace awareness,* the ability of tool to allow users to see what other collaborators are doing in real-time, such as the cursor location of others in a document.

MONDO collaboration framework enables offline and real-time collaboration for models hosted in a version control system (VCS), currently SVN repository [29]. MONDO provides a secure collaborative framework for VCSes, but it is not real-time. The real-time collaboration is handled by offering a web based modeling front-end (Eclipse RAP-based web application). In contract to BUMBLE-CE, this web user interface is language dependent and, therefore, needs to be provided separately for each modeling domain supported by

[1] https://www.jetbrains.com/mps/

TABLE 1. Comparison of BUMBLE-CE with existing real-time collaboration tools/approaches.

Tool/Approach	Technical space	Cross platform	Change propagation	Language independent	Conflict resolution	Workspace awareness
MONDO	EMF	No	Fixed	No	semi-automated	Yes
AToMPM	AtoMPM	No	Fixed	Yes	No	Yes
WebGME	WebGME	No	Fixed	Yes	No	Yes
Collaboro	EMF	No	Fixed	Yes	semi-automated	Yes
Parsafix	MPS	Yes	Fixed	No	No	No
FlexiSketch	Custom	No	Fixed	Yes	semi-automated	Yes
BUMBLE-CE	EMF	Yes	Flexible	Yes	No	No

MONDO collaborative modeling tool. Workspace awareness is mainly provided by the versioning system (*e.g.,* commit messages, update notifications, etc.).

AToMPM is a cloud-based multi-user tool that provides an in-browser interface for different modeling activities [30]. Each client of AToMPM has its own view of the same model, using its own concrete syntax. The real-time collaboration is supported either (i) by sharing the same model and canvas or (ii) by sharing the abstract syntax only. In both cases, changes in the abstract syntax are shared; in the first case, changes in the representation are shared as well. AToMPM does not offer any locking or conflict resolution mechanism. Similar to BUMBLE-CE, the first change seen by the server always wins. AToMPM provides workspace awareness by making updates from other users directly visible in the editor.

WebGME [31] is a web-based collaborative (meta) modeling tool that supports online collaboration and model versioning. Shared data are protected by applying user access control. WebGME uses lightweight branching and allows users to create commit objects and send branch update messages to the server. The user is responsible for manual creation of branches and there is no mechanism available for automatic conflict resolution, or locking to prevent conflicts. WebGME also provides workspace awareness by making updates from other users directly visible in the editor and providing general notifications, such as a popup messages.

Collaboro, based on EMF, provides real-time collaboration facilitation for creation of domain-specific languages [32]. Its metamodel is generic and can be applied to various group decision-making problems. Collaboro supports both static (*e.g.,* change proposals) and dynamic (*e.g.,* voting) aspects of collaboration. Although Collaboro enables team work between stakeholders at different levels of abstraction and decision making, all the stakeholders involved in collaboration need to agree on all of the proposals as Collaboro only adopts a consensus based policy.

In our previous work, we presented a language dependent tool, Parsafix, which enables real-time collaboration among engineers working on Modelix and Saros [33]. Modelix is a real-time collaboration tool for JetBrains MPS, and Saros is a real-time collaboration tool for the Eclipse IDE (as well as IntelliJ). Parsafix achieved its goal by facilitating indirect communication between existing real-time collaboration technologies that the modeling platforms possess, posing as a client for each and translating model data from one side to the other, ultimately forming a collaborative project. From all the tools listed in Table 1, only Parsafix supported cross-platform collaboration, however, the tool is not extensible. This implies that we could not incorporate more editors as collaboration parties in Parsafix.

The recent capabilities of mobile devices have enabled their use for modeling. FlexiSketch facilitates domain-specific modeling on the mobile devices, mainly for the purposes of requirements elicitation [34]. FlexiSketch allows modeling and metamodeling to be done in any order. For instance, a user can sketch a model informally, introduce new sketched elements, and then upgrade some of the elements as concepts of a language. One of the problems with FlexiSketch is an ever increasing metamodel size with every new modeling project. For conflict resolution, FlexiSketch uses a non-optimistic locking mechanism to prevent conflicts by preventing the modification of the same element by more than one user. In such cases, the element is shown with a red background and becomes non-reactive to the inputs of other users.

The comparison presented in Table 1 shows that BUMBLE-CE mainly distinguishes from other real-time collaboration engines in (i) facilitating modelers working in different platforms to work together (*Cross-platform*) and, (ii) allowing the modelers to incorporate a mechanism of their choice for propagation of edit operations (*Change propagation*) among collaboration parties during collaboration sessions. The latter point is achieved by making BUMBLE-CE extensible by providing extension points so that engineers can plugin their custom logic in BUMBLE-CE.

IV. REQUIREMENTS FOR BUMBLE-CE

The requirements for BUMBLE-CE are elicited from the use cases provided by both the academic and industrial partners in the BUMBLE project. The purpose of these requirements is to clarify what functionality must be supported by BUMBLE-CE. The starting point for the design of BUMBLE-CE is that, given a DSML and its corresponding editors, it shall provide a collaboration mechanism that allows multiple modelers to collaboratively edit the models in real-time across different modeling platforms. Within BUMBLE-CE, we identify the following stakeholders and related responsibilities:

- BUMBLE-CE admin: The technical person administrating the front- and back-end of BUMBLE-CE. The admin will be responsible for setting up the collaboration engine for a specific use case, *e.g.,* they add the languages and their supported editors for which

collaboration is enabled, they integrate the various plugins to be used for collaborating, they assign roles to modelers, etc.
- MDSE engineer: A person developing the plugins (*e.g., Consistency Checker, GitHub Driver*) for our BUMBLE-CE engine.
- modeler: A modeler is an MDSE practitioner collaborating on models with other modelers through BUMBLE-CE.

Below we present the requirements we identified for supporting cross-platform real-time collaborative modeling in BUMBLE-CE.

R1: *Independence from the used modeling platform.* Agnostic of a particular platform the modelers have chosen (*e.g.,* EMF or MPS) – changes made in one platform shall be propagated to all other editors in the other modeling platforms. The system shall be fully compatible with the existing platforms such that it shall fully preserve all existing functionalities and features, user interfaces and workflows for modeling and non-real-time collaboration. In this way modelers would effectively reuse their knowledge, skills, and experience in a familiar environment along with real-time collaboration functionalities provided by the collaboration engine.

R2: *Freedom of using a preferred concrete syntax.* Modelers shall be able to choose the editor/view to be used to edit/view (elements of) the models they are working on. This means that independently from the concrete syntax the modelers have chosen, changes by an individual modeler are instantly visible to all other modelers that have viewing/reading and/or editing/writing rights to the considered (collection of) models.

R3: *Authentication of modelers.* The modelers shall be identified by means of an authentication step (*e.g.,* with a login) when accessing the modeling environment. For this purpose, modelers will be assigned with username and password to be used as login credentials.

R4: *Access rights to models.* Modelers shall only be able to collaborate on models for which they have access rights. For a collaboration session, the *initiator* (a modeler who started the collaboration session) will be able to invite other modelers (collaborators) to the collaboration session.

R5: *Undo functionality for modeler's own actions.* By default, a modeler shall at least have full access rights to model elements that they modified. In particular, while editing a model, a modeler must at least be able to perform undo actions for modification that they made and is (by default) not able to undo modifications performed by other modelers.

R6: *Supporting real-time collaboration.* Changes made by a modeler in one editor should be visible to the modelers in real-time in other editors. This implies that during a real-time collaboration session, BUMBLE-CE should ensure high local responsiveness such that each local editing operation performed on the models must immediately take effect without noticeable delay.

R7: *Supporting model validation.* It shall be possible to view errors/notifications on the results of model validation in the editor/view for any concrete syntax that represents (elements of) the corresponding model. Model validation is therefore to be realized at the level of (elements of) the relevant model while the interaction with the modeler is to be performed via all of the available concrete syntaxes.

R8: *Independent cross-referencing of model elements.* Cross-referencing between elements of the same model must be agnostic of the specific syntax that a modeler may have selected to edit/view such model.

R9: *Supporting custom modeler's preferred actions.* Modelers must be able to enable extra functionalities preferred by them to fulfill their needs for performing specific actions on models *e.g.,* generating code, checking consistency between models, etc.

V. THE SOFTWARE ARCHITECTURE OF BUMBLE-CE AND ITS DESIGN DECISIONS

In this section, we present the generic architecture of BUMBLE-CE which enables modelers to collaborate in real-time across different platforms by fulfilling the requirements illustrated in the previous section. Our cross-platform real-time collaboration engine implies a star topology where the BUMBLE-CE will be deployed on a server at the center of the star and various collaboration parties will run on third-party modeling environments locally on the modelers' machines.

Figure 1 shows an example scenario where four modelers are collaborating on the same model of a state machine across different modeling platforms. Specifically, modeler A is working in an Eclipse[2] environment where the state machine is modeled textually via an editor implemented in xText,[3] modeler B is also working in an Eclipse environment but in this case the model is edited via a GMF-based graphical editor, modeler C is working in an MPS environment with a textual editor, and finally modeler D is working on a custom web app implemented in Angular[4] where the state machine is edited via a custom graphical editor.

The main responsibilities of BUMBLE-CE is to bring up, terminate, and enable real-time collaboration sessions among modelers working on the same model. Intuitively, when two or more modelers are collaborating on the same model, every edit operation on the client side is propagated to the BUMBLE-CE (that is, the center of the star topology), properly remapped to the concrete representations and formats of the other modelers participating in the collaboration session, and then propagated to all the other involved modeling environments.

The example scenario already shows some key design decisions we took for BUMBLE-CE.

[2]https://www.eclipse.org/
[3]https://www.eclipse.org/Xtext/
[4]https://angular.io/

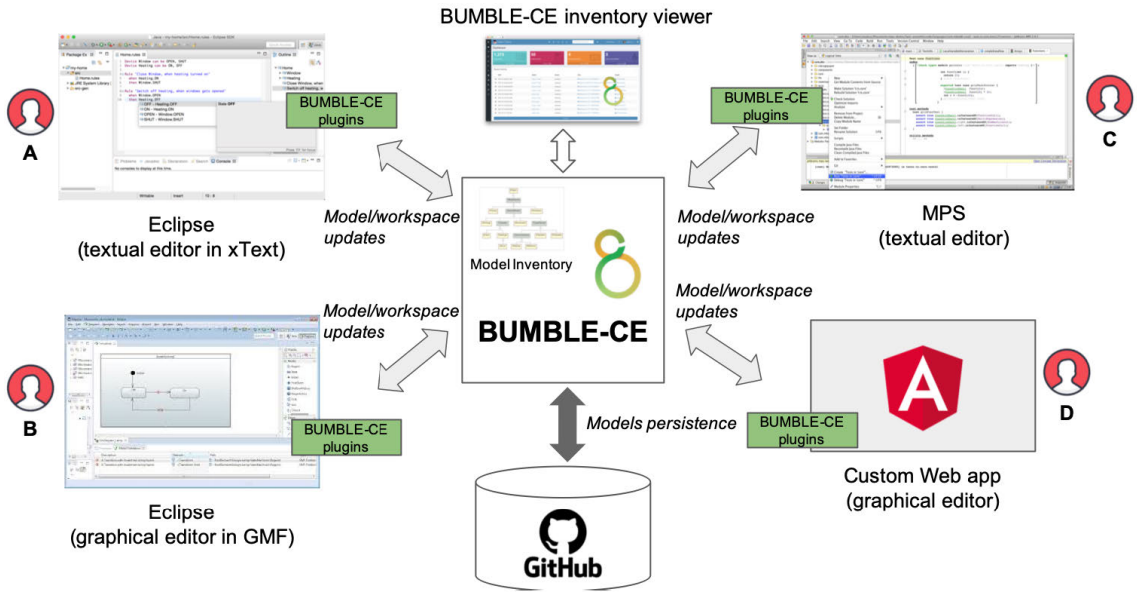


FIGURE 1. Example of cross-platform real-time collaborative modeling enabled by BUMBLE-CE.

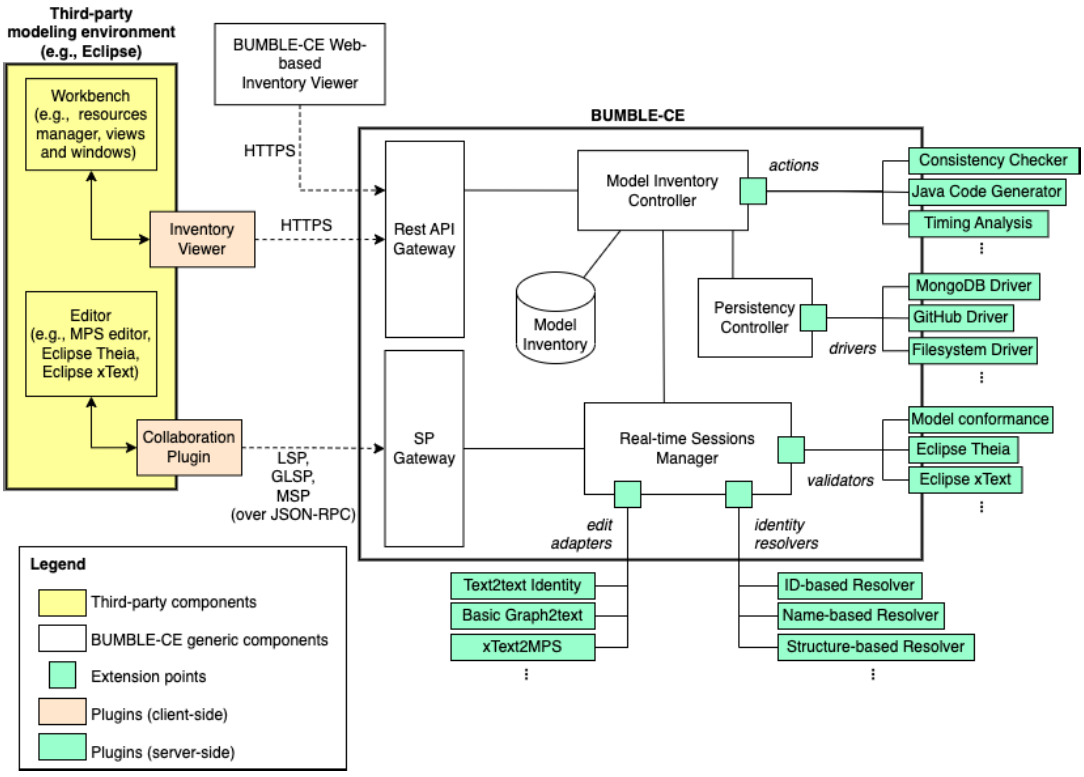


FIGURE 2. Software architecture of BUMBLE-CE.

- We design BUMBLE-CE with minimal impact on the tools and modeling environments used by the modelers. Real-time collaboration is realized via a set of dedicated plugins that are suitably loaded in the (already-existing) modeling environment of the modeler. The BUMBLE-CE plugins have two main responsibilities: (1) to establish a continuous communication channel to the central BUMBLE-CE for sending/receiving edit operations in real-time and (2) to provide an always-updated inventory viewer, which allows the modelers to get an overview about all models managed within the project, the current collaborative sessions, the

status of the models (*e.g.,* persisted on Git, valid/not valid etc.).
- A track record of all the models, relationships between the models, modelers, collaboration sessions and DSML definitions used in the project are kept by means of a so-called *Model Inventory*. The *Model Inventory* stores various relationships between the models (which can be freely customized by its modelers) and the DSMLs definitions. The *Model Inventory* keeps track and provide information about the currently active real-time collaboration sessions, models that are being collaborated on, and modelers currently involved in collaboration sessions. The inventory viewers on client side is developed once and for all for each modeling environment (*e.g.,* Eclipse, MPS), whereas collaboration plugins are developed once and for all for each type of editor (*e.g.,* Eclipse xText, MPS textual editor, Eclipse Sirius).
- During a real-time collaboration session, models are not persisted and all the edit operations are performed in memory and internally stored in a stack of commands. Model persistency is managed externally by BUMBLE-CE by means of persistency drivers. A default implementation for the persistency driver is provided in our work. In the scenario described above, we show a basic GitHub driver which performs a ''commit and push'' operation on a selected GitHub repository as soon as the modelers collaborating on the models agree that their model reached a certain level of stability; this operation also resets the collaboration session and its corresponding stack of commands.
- During a real-time collaboration session, the edit operations is stored in memory.
- The contents of the *Model Inventory* can be inspected at any time via a web-based viewer, which is independent of the modeling platforms used by the modelers for collaboration.

A more detailed description of the architecture of BUMBLE-CE is shown in Figure 2. In the remainder of this section we describe each component of the architecture and relate it to the main responsibilities, technical choices, and its relationship with the requirements reported in previous section.

A. MODEL INVENTORY

The responsibility of the *Model Inventory* is to represent and store the metadata of all modelers, models, DSML definitions (*i.e.,* the metamodels underlying the used DSMLs), their relationships and the ongoing collaboration sessions. Internally, the *Model Inventory* is implemented as a megamodel [35]. Megamodeling has been proposed with the aim of supporting modeling in the large, that is, dealing with models, metamodels, and their properties and relations. Intuitively, a megamodel is a model in which the atomic units of information are other models (or metamodels). While a metamodel specifies properties and rules governing models construction, a megamodel specifies properties and rules governing MDSE artefacts construction, and among them, models and metamodels. Megamodeling offers the possibility to specify relationships between models (and metamodels) and to navigate among them. This is fundamental in the BUMBLE-CE since it allows us to abstract from specific technical spaces of collaborating parties, attach collaboration-specific metadata to every model/DSML and keep track of the real-time collaborative sessions and their associated metadata.

By following the models@runtime principles [36], the megamodel stored in the *Model Inventory* is continuously updated, so as to keep in sync the workspaces of the clients' modeling environments.

B. MODEL INVENTORY CONTROLLER

The *Model Inventory Controller* has the main responsibility of providing standard functionalities for accessing and manipulating the *Model Inventory*, such as CRUD operations on all its contained information about models, collaborators, languages, supported editors. *Model Inventory* already contain information about the access rights of modelers encoded in the corresponding roles associated to them (*i.e.,* owner, initiator and collaborator). The BUMBLE-CE admin assigns these roles via the *BUMBLE-CE Web-based Model Inventory Viewer*. These access-based roles capture the relation between the modeler and the model under collaboration. Different authentication and authorization mechanisms are included in the *Model Inventory Controller* which make use of information about modeler roles to enable the satisfaction of requirements **R3** and **R4**.

Model Inventory Controller provides an extension point for third-party plugins which allows third-party actions to be called at specific moments in time, for instance, when a new model is created, when a real-time collaborative session is terminated, when a modeler pushes a certain button on their modeling editor, etc.). Examples of providers of such external actions include consistency checkers for specific DSMLs, code generators, timing analysis tools, etc. This functionality has been provided to satisfy requirement **R9**. A special type of action consists of model validators, which allow the *Model Inventory Controller* to check for the presence of errors in the models and then to notify the modeler accordingly.

The *Model Inventory Controller* communicates with the *Real-time Sessions Manager* when new real-time collaboration sessions need to be spawned or terminated, depending on the requests arriving from the modelers. Also, this component communicates to the *Persistency Controller* by issuing requests for fetching or saving models.

C. INVENTORY VIEWER

The *Inventory Viewer* is a platform-specific plugin loaded on the client side of the architecture aimed at interacting with the *Model Inventory*. It allows modelers to suitably integrate the API provided by the Rest API Gateway into the client. For example, modelers using Eclipse are able to visualize all models and DSML referenced by the *Model Inventory*,

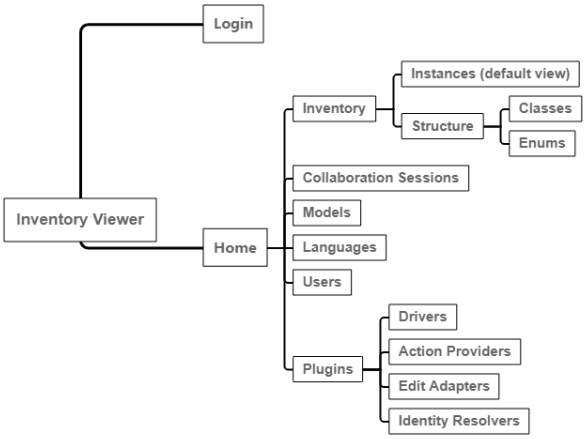


FIGURE 3. Overview of the information architecture of the BUMBLE-CE web-based model inventory viewer.

to manipulate its internal megamodel, and to trigger the third-party actions and drivers provided by the *Model Inventory Controller* and *Persistency Controller*, respectively. For the sake of simplicity, at the moment we are envisioning that *Inventory Viewer* plugins will communicate with the REST API Gateway by means of long polling via HTTPS.

D. BUMBLE-CE WEB-BASED MODEL INVENTORY VIEWER

The *BUMBLE-CE Web-based Model Inventory Viewer* allows modelers and MDSE engineers to quickly inspect and manipulate the megamodel stored in the *Model Inventory* at any time. This component also allows modelers to trigger all the actions currently loaded as plugins of the *Model Inventory Controllers* and to trigger the drivers used for the persistence layer. On the client side, this action can be performed using the *Inventory Viewer* plugin.

The user interface of *BUMBLE-CE Web-based Model Inventory Viewer* is designed with respect to the usability principles [37] and accessibility guidelines [38] to meet the functionalities proposed in the requirements with usability and accessibility. The *BUMBLE-CE Web-based Model Inventory Viewer* has two distinguished layouts in page style, namely the **Login page** layout and the **Home page** layout. The information architecture of our inventory viewer is shown in Figure 3. All the structures listed behind the Home page is different sub-pages that are displayed on the information board. On the Login page, the user of *BUMBLE-CE Web-based Model Inventory Viewer* enters the credentials (username and password). On successful entry, the user is redirected to the Home page. The Home page of the inventory viewer is where users may access information stored in the *Model Inventory*. Within the Home page layout, sections such as collaboration sessions and languages are located. The default view of the Home page provides a concise view of the inventory instances. This means that this is the sub-page to which the user is directed after logging in.

E. COLLABORATION PLUGIN

The main responsibility of this component is to establish and maintain a bidirectional communication channel with the collaboration engine during a real-time collaboration session. The communication channel is built on top of the well-known concept of language server protocol (LSP). The *Collaboration Plugin* uses a different flavour of LSP, depending on the type of editor used by the modeler. Based on the results of our studies of the state of the art and practice [6], [7], we aim to support the following three protocols:

1) LANGUAGE SERVER PROTOCOL (LSP)

It is the most stable version of the LSP concept and supports actions predicating on text-based edits, such as addition of a character or removal of a block of text. This protocol is used for propagating the changes performed in textual editors.

2) GRAPHICAL LANGUAGE SERVER PROTOCOL (GLSP)

It is an evolved version of the LSP with a focus on graphical modeling languages. Instead of dealing with textual edits, its main actions predicate on typical elements of a graphical editor, such as addition/removal of nodes and edges, moving of a node, rerouting of an edge and more.

3) MODEL SERVER PROTOCOL (MSP)

Such a protocol focuses on the structural semantics of an edited model and the semantic changes of a model with respect to the structure dictated by the metamodel it conforms to. Examples of actions represented in this protocol include: addition/removal of a model element (*i.e.,* an instance of a metaclass), update of the value of a structural feature of a model element (*i.e.,* the update of either one of its references or attributes). A platform-independent MSP does not exist for now. Such an MSP allows for a direct treatment of the semantic changes happening in the models, thus keeping BUMBLE-CE more future proof since it is capable of managing also other types of editors that are not currently covered by LSP or GSLP (*e.g.,* tree-based editors, table-based editors, form-based editors etc.).

All the above-mentioned protocols are language-independent, meaning that the information passing from the *Collaboration Plugin* to the BUMBLE-CE encodes also the meta-information related to the performed edit operations. However, the *Collaboration Plugin* is coupled to the specific editor being used (*e.g.,* Eclipse xText, Eclipse Sirius, MPS editor) since it needs to know how to listen to the edit operations performed in the local editor in order to relay them through one of the language server protocols. For example, in editors based on the Eclipse Graphical Modeling Framework (GMF), Java classes can use an instance of the DiagramEventBroker class, which acts as a listener for model changes and broadcasts EObject events to all registered listeners. This behaviour is specific to Eclipse GMF and other editors have similar (but not identical) event-based systems for notifying listeners to model changes. The local

editor is also updated when changes are made on the other collaborating parties, still via the three protocols mentioned above. In this case it is the BUMBLE-CE which propagates changes from the other editors to the local one. Different collaboration plugins might be used within the same modeling environment, thus allowing the modeler to choose their preferred editor to be used when collaborating on a certain model or a default one. *Collaboration plugin* fulfills the requirement **R2**.

F. SP GATEWAY

The SP Gateway acts as a reverse proxy that accepts JSON-RPC messages from the collaborating parties and suitably (i) abstracts from the details of the used language server protocols and (ii) routes the messages to the correct real-time collaboration session. This design choice allows us to be independent of the specific version of the used language server protocol and when possible, from the specific Editor Plugins, decouple the server-side components of the collaboration infrastructure from the third-party plugins and to balance the load of requests.

G. REAL-TIME SESSIONS MANAGER

This component is the core of the real-time features of the collaboration engine. Its main responsibility is to satisfy requirements **R1 and R6**, specifically: (i) to bring up real-time collaboration sessions, (ii) to propagate edit operations coming from each collaborating party to all the other collaborators, and (iii) provide a set of extension points for external plugins where third-party developers can extend the default real-time collaboration mechanisms with their own business logic (*e.g.,* by implementing their own model validator, resolvers for tracing model elements across modeling environments). Collaboration sessions happen in real-time, meaning that modelers can perform live concurrent editing on the same model. This can be achieved by exploiting a particular class of algorithms for multi-site real-time concurrency, called operational transformation [24]; or alternatively evaluating the usage of conflict-free replicated data types (CRDT) [39]. In both cases, it is possible to reasonably merge concurrent updates performed by different modelers without conflicts [39], [40]. In our work, we do not support conflict resolution for now. In the future, we aim to incorporate existing conflict management techniques for handling concurrent changes occurring during real-time collaboration sessions managed by BUMBLE-CE [41].

Internally, the *Real-time Sessions Manager* follows the publish-subscribe architectural pattern [42]. Every time a real-time collaboration session is started, a dedicated event bus is created capable of managing textual, graphical and semantic edits. Then, the following sequence of operations is performed: model validation, edit adaptation, and edit propagation. Since model validation, edit adaptation, and identity resolution depend on the semantics of the models and on the specific constraints of the collaborating parties, we decided to make BUMBLE-CE independent from their specific implementations. BUMBLE-CE provides two corresponding extension points to third-party developers for implementing their own business logic for model validation, edit adaptation and identity resolution as external plugins. This makes BUMBLE-CE independent of the specific implementations of the collaborations parties. This design choice is also convenient from a scientific perspective since it will allow us to experiment with different combinations and heuristics for collaborative modeling, and to still control the overall collaboration model (this will make our experiments replicable and independently verifiable). In the following, we describe each of the above-mentioned operations.

1) MODEL VALIDATION

It checks the well-formedness of the current edit operation. Depending on the specific needs of a software project, different validators can be executed on the currently-edited model and the received edit operations. For example, in a project we might have a constraint that every edit operation must always lead to a valid instance of the DSML.

The current edit operation proceeds to the next step of the sequence (*i.e., Edit adaptation*) or is discarded, depending on the outcome of the model validation step. This functionality is provided to satisfy requirement **R7**.

2) EDIT ADAPTATION

It adapts the currently-received edit operation (*e.g.,* a textual change) to the other modeling environments participating in the collaboration (requirements **R8**). This step is crucial since it is not possible to simply forward the edit operations performed on one modeling environment to all the other modeling environments. As we learned in our previous work [33], this is true even if the concrete syntax of the modeling environments is the same. *Edit adaptations* can act also as a filter, for example, we expect that several changes performed in graphical models might not need to be forwarded to textual modeling environments. Examples of edit changes which might be filtered out when passing from a graphical edit to a textual edit include: moving a node within the diagram, resizing of a node, rerouting of a connector etc.

3) IDENTITY RESOLVER

When adapting an edit operation, it is crucial to keep information about which element in the model edited in a modeling environment corresponds to which elements in other modeling environments. Internally, the edit adaptation step can use one or more *identity resolvers*. The main responsibility of an *identity resolver* is to implement a rule or a heuristics for mapping modeling elements in one environment to modeling elements in another environment. Such a resolution can be generic (*e.g.,* elements with the same ID are matched, name-based matching, structured-based matching etc.) or editor-specific.

Lastly, during edit propagation, the event bus sends the edit operations to all subscribed parties via the *SP Gateway*. This

means the *Collaboration Plugins* running in the modeling environment of the modeler receives the changes performed by other collaborators and applies them in the locally-edited model.

Internally, each real-time session stores a command stack, which allows modelers joining an ongoing real-time collaboration session to get the latest version of the model being edited according to the edit operations performed by the collaborating team.

H. PERSISTENCY CONTROLLER

The main responsibility of this component is to provide an abstract layer for persisting the various models referenced by the *Model Inventory*. Such an abstraction layer allows us to treat model persistence in a platform-independent manner. Indeed, the *Persistency Controller* provides a generic API to the *Model Inventory Controller*, independently of the persistence technologies used in the project (*e.g.,* databases, VCSs). Examples of endpoints provided by such generic API include:

- fetchModel(id): loads the model with the given id into the Model Inventory
- fetchAll(): loads all persisted models into the Model Inventory
- save(id): persists the model with the given id
- saveAll(): persists all models

The *Persistency Controller* receives calls according to the provided generic API and maps them towards the specific persistency technologies used in the project. The *Persistency Controller* is independent of any persistence technology, which is treated as external plugins of the BUMBLE-CE. Indeed, depending on the specific requirements of the project, modelers can decide to persist their models using one of the drivers that third-party developers will develop. In Figure 2 we have three examples of such drivers:

- MongoDB Driver: stores the models as entries in a MongoDB instance; this solution might be convenient for managing large models with thousands of model elements;
- Git Driver: saves the models by performing a combination of add, commit, push commands on a given GitHub repository and fetches them using a pull command; this solution might be convenient for managing the history of the models across branches.
- Filesystem Driver: saves the models as raw files in the file system of the server where the BUMBLE collaboration engine is running; this solution will be used for rapidly prototyping the other components of the BUMBLE collaboration engine, without incurring additional development effort.

  Depending on the specific scenario, the *Persistency Controller* might be called either on-demand by the modeler or automatically by the platform.

I. REST API GATEWAY

The *Rest API Gateway* allows the collaborating parties to interact with the *Model Inventory Controller* in a platform-independent manner. It acts as a reverse proxy which (i) accepts HTTPS messages from the collaborating parties and (ii) calls the corresponding methods of the *Model Inventory Controller* accordingly. This solution allows us to decouple the *Model Inventory Controller* from the third-party modeling environments running on the client side. Similarly to the *SP Gateway*, the presence of the *Rest API Gateway* allows us to balance the load of requests in case during the project we observe performance bottlenecks on the server side.

VI. PROTOTYPE IMPLEMENTATION

In this section, we describe the current status of the prototype implementation of BUMBLE-CE. The implementation for BUMBLE-CE allows modelers to initiate and maintain a real-time collaboration across different platforms. The implementation is available online and can be used by other researchers to replicate our work and extend the functionality of BUMBLE-CE, since all the components from the proposed architecture are not yet implemented. Below we present in details the components which are more stable from a technical point of view, namely: *Model Inventory* (Section VI-A), the *Model Inventory Controller* (Section VI-B), the *Web-based Model Inventory Viewer* (Section VI-C), and two collaboration plugins on the editors side, one for Eclipse EMF and one for Jetbrains MPS (Sections VI-D and VI-E, respectively).

A. MODEL INVENTORY

The *Model Inventory* is a repository used by BUMBLE-CE to store and keep track of models, DSML definitions, their relationships, and their relevant metadata. The *Model Inventory* contains a model conforming to a model inventory metamodel. The *Model Inventory* metamodel is defined as an Ecore metamodel and it is graphically shown in Figure 4. The metaclasses in our model inventory metamodel represent the main concepts related to BUMBLE-CE and the extension points for several functionalities provided by the plugins developed by MDSE-engineers. The main concepts of the model inventory metamodel include *User, Model, Collaboration Session, Language, Editor, Participant*, and the extension points include *Action Provider, Validator, Driver, Identity resolver* and *Edit adapter*. Below we describe the main concepts of this metamodel:

- **User**: The *User* class represents the modeler in the context of BUMBLE-CE. Name, password, and email address are included in the User's attributes. In addition, it contains references to the *Model* metaclass which represents models owned by the user.
- **Language**: This metaclass contains information about the various DSMLs used to define the models on which real-time collaboration can be activated. This metaclass

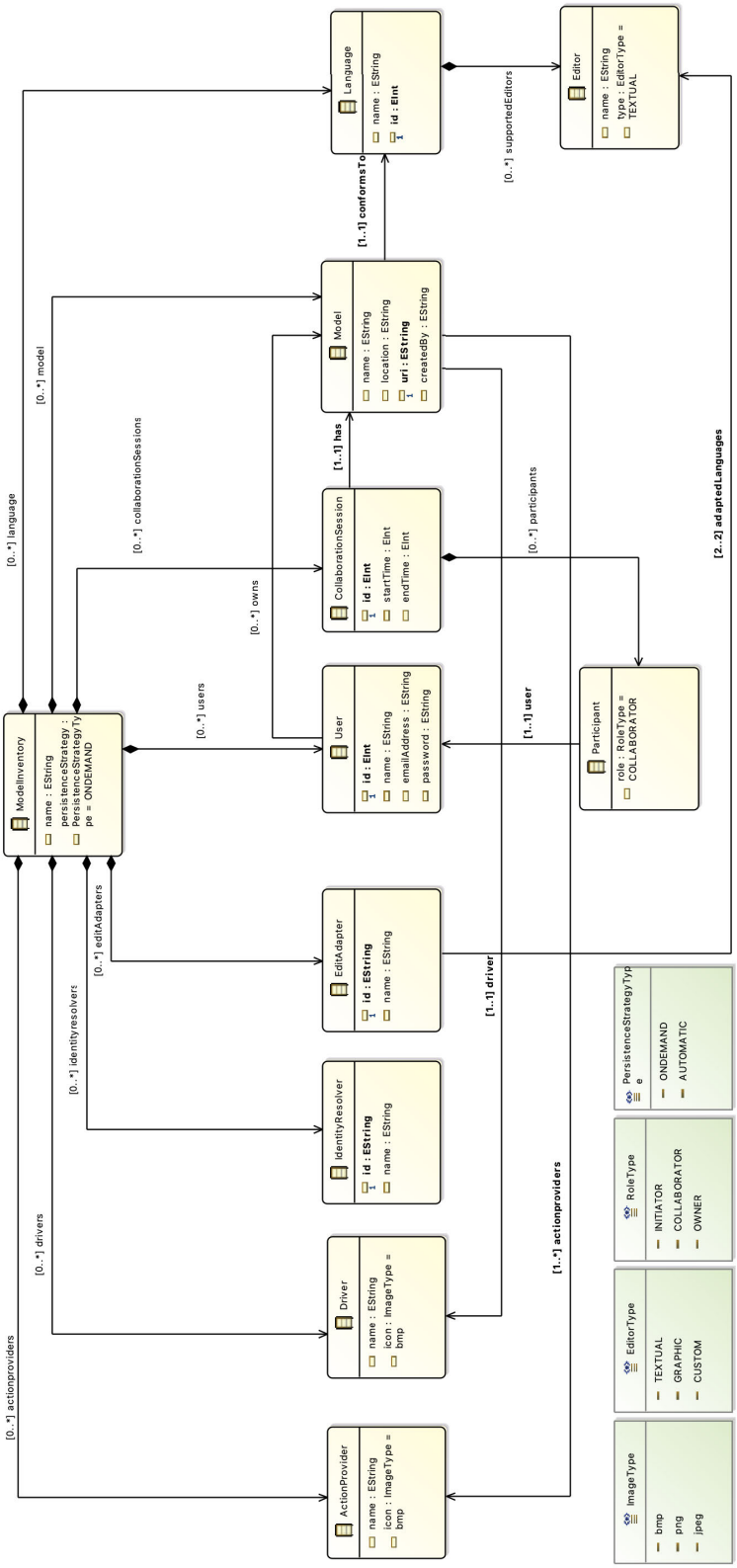


FIGURE 4. Model inventory metamodel.

contains the name of the languages and their (one or more) supported editors.

- **Editor**: The *Editor* metaclass stores information about the editors that support the languages stored by the metaclass *language*. For instance, Microsoft Visio is an editor that supports modeling in UML.
- **EditorType:** is an enumeration used to store the information about whether the editor is *textual*, *graphical*, or *custom*. For now, BUMBLE-CE does not support graphical editors.
- **Model**: represents a model on which collaboration can be activated. Each model possesses the following attributes: name, location, URI, and creator. A model can conform to one and only one language and it refers to its corresponding driver (for managing its persistence) and action providers (for triggering actions on the model).
- **CollaborationSession**: The modelers can initiate a session to collaborate with each other in real-time by right-clicking on the model and choosing Start Collaboration. We call this session as CollaborationSession in the metamodel. Each collaboration session is related to one model and is opened for modelers to collaboratively edit a model in real-time. It has a start and end time as its attributes.
- **Participant** represents the modelers who are currently participating in a collaboration session, with a certain role (read below).
- **RoleType:** is an enumeration that stores the role of a participant within an ongoing collaboration session. A modeler can participate in a collaboration session with one of the following three roles: *initiator* - the modeler who starts a collaboration, *owner* - the modeler who owns a model associated with the sessions, and *collaborator* - the modeler who participates in the session.
- **Driver:** The metaclass *Driver* stores the information about the drivers (for instance, a GitHub driver or a MongoDB driver) chosen by the modeler to persist their models.
- **ActionProvider:** This metaclass stores information about the third-party plugins that may provide functionalities desired by the modelers for their specific needs, such as to check consistency between models, generate code from models and so on.
- **EditAdapter:** Information about the plugins needed for adapting the edit operations received from one modeling platform to another is stored in the metaclass *EditAdapter*.
- **IdentityResolver:** The metaclass *IdentityResolver* stores information about the plugins responsible for mapping modeling elements in one modeling environment to the corresponding elements in another modeling environment.
- **PersistenceStrategyType:** The changes made to the model can be saved either *automatically* at regular intervals or *manually* when the modeler decides to save the changes.
- **ImageType:** This enumeration is used to indicate the image type of the thumbnails/icons for the plugins created by the MDSE-engineers.

It is important to note that the metamodel of the *Model Inventory* is one of the artifacts living inside BUMBLE-CE at runtime; such a metamodel defines the data structures for storing the metadata of the models and collaboration parties involved in the real-time collaboration. This metamodel shown in Figure 4 does not represent the metamodel of BUMBLE-CE.

B. MODEL INVENTORY CONTROLLER

A *Model Inventory Controller* operates on the *Model Inventory* and offers CRUD operations as well as authentication and authorization functionalities to the modelers. The *Model Inventory Controller* is built on the existing EMF.cloud model server [28]. The controller and the web-based model inventory viewer communicate through the *REST API Gateway*, which is based on the default REST API provided by EMF.Cloud.

The EMF.Cloud model server's registration system makes sure the instance models follow the scheme of the corresponding metamodels. For modeling environments that do not use UUIDs as identities for models, the server generates UUIDs and assigns them to each model component. This enables models to be initiated and changes to be managed on platforms with various model-saving mechanisms.

The server can process PATCH requests in a number of formats for change propagation, including EMF command style, JSON patch with EMF-like paths, and JSON patch using JSON Pointer paths. After receiving a change patch, the server applies the update to the saved model and uses web sockets to communicate the changes to its subscribers. The server can send updates in the JSON, XMI, and EMF command formats. Therefore, the collaboration plugins customized for different platforms can select the most effective method to encode and decode updates sent to and received from the server.

For a modeler that joins a given collaboration session, *Model Inventory Controller* ensures that the selected local model corresponds to one in the model server. This is done by performing a GET request, comparing the model received from the server with the local model and making any changes,if needed. A websocket connection is then initiated to the model server to receive changes made to the model by the collaborators during the collaboration session. The Websocket connection follows a publish-subscribe mechanism, where the modelers subscribe to the selected model, and the model server publishes a given change to all the modelers. One downside to this mechanism is its 'fire and forget' property, *i.e.,* there is no record of changes already published. For our work, this has no impact as once subscribed to the model server, a PATCH request is sent to report a change to the server conforming to the layout of the model server, which would

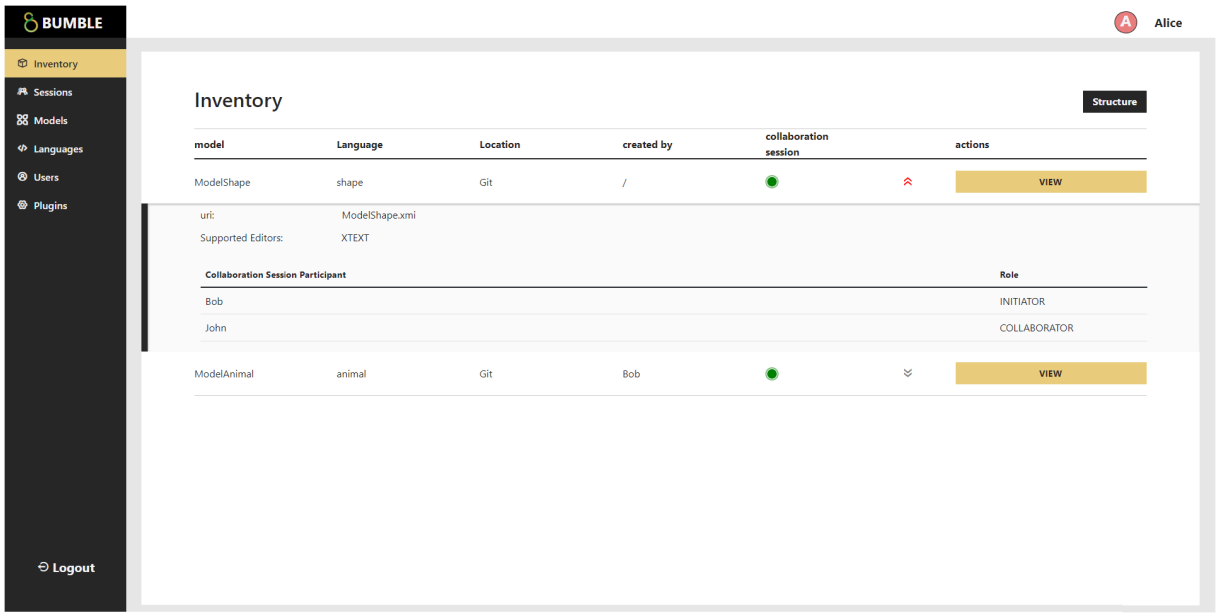


FIGURE 5. Main page of the *Web-based Model Inventory Viewer* showing *Model Inventory* instances.

thereafter immediately be broadcasted to all the subscribers from the server.

C. WEB-BASED MODEL INVENTORY VIEWER

The *Web-based Model Inventory Viewer* has been developed with the goal to facilitate BUMBLE-CE admin to examine and interact with the megamodel that is saved in the inventory. The *Web-based Model Inventory Viewer* provides modelers working in different development environments with a graphical interface for inspecting modeling artefacts and interacting with the *Model Inventory* units through REST API. From the standpoint of the *Web-based Model Inventory Viewer*, BUMBLE-CE provides Rest endpoints to get and update the contents of the megamodel representing the contents of the *Model Inventory*.

The *Web-based Model Inventory Viewer* is developed using the Vue.js[5] javaScript framework. Vue.js is a user interface framework that builds on top of HTML, CSS, and JavaScript standards. It offers declarative and component-based programming methods that improve the productivity of user interface development. Vue is selected as the framework for this project because of its flexibility and good performance [43] among popular frameworks currently available. It is reasonably simple to learn and use, and better suited for smaller projects [44].

State management refers to the management of the state of one or more user interface controls such as text fields, OK buttons, radio buttons in a graphical user interface. State management is also required in our work in order to maintain track of the current status of inventory items and information of the modeler. We use Vuex,[6] the official state management library for Vue.js, to fulfill this need. Vuex adheres to the same Flux principles [45], a set of guiding principles that define statement management patterns: Figure 5 shows the main page of the *Web-based Model Inventory Viewer*.

The *Web-based Model Inventory Viewer* allows modelers to perform CRUD operations on the instances of the *Model Inventory* metaclass, with the exception of collaboration sessions. This is because the *Model Inventory* controls the sessions, and users of inventory viewer are not permitted to edit them in this project. The REST API provided from the server and the fact that *Model Inventory* exists as a megamodel make the CRUD requests different from common REST-based communication paradigms. Normally, *create, delete* and *update* requests are separated using different HTTP methods such as POST, DELETE and PUT. The API offered by the server to DELETE and POST a model, however, only supports independent models. *Model Inventory* is a single megamodel, and all class instances are kept in the XMI-based representation of the *Model Inventory*. This transforms every CRUD operation on a *Model Inventory* instance into an UPDATE operation on the *Model Inventory* XMI file. Therefore, all of the *create, delete* and *update* actions applied to the instances are achieved through PUT request to update the XMI model representing the *Model Inventory*.

D. COLLABORATION PLUGIN FOR ECLIPSE EMF

On the EMF side, the *Collaboration Plugin*[7] uses an activator to control its life cycle. It activates a new window of a run-time editor and initiates the server client when the modeler launches the collaboration application and terminates the plugin when the modeler exits the editor window.

[5]https://vuejs.org
[6]https://vuex.vuejs.org
[7]https://github.com/blended-modeling/nl.vu.cs.bumble.emfcollaboration plugin

The run-time model editor allows the modeler to import new models from various resources or generate models from existing metamodels.

In this prototype, we use the publish-subscribe pattern for data communication. The Eclipse *Collaboration Plugin* listens to the changes happening in the editor domain and sends them to the server. The EMF cloud server acts as the publisher that receives messages from clients (*i.e.,* subscribers) and propagates the changes to all of its subscribers. The changes that were received by the client are then applied to the local model in the editor domain. In the example application, we assume the Ecore model already exists in the server so that modelers can post models generated from it into the server. Once the collaboration button is clicked, the plugin first checks if the local model has a copy in the server by sending a GET request to the server. If the server confirms that the model exists then the model is pulled from the server and the local version stored in the editor's resource set is replaced with the server version. If the model is not in the server, the plugin will POST the local model to the server. At this stage, the communication between the client and server is done via REST API. Once the initiation is done, the client subscribes to the model in the server through the WebSocket protocol.

To close the cycle of the publish-subscribe pattern, the plugin contributes to the process of *propagation* and *subscription* and has a built-in *listener* to monitor local changes.

1) LISTENER

In the context of EMF, the model itself and its contents can all be represented using EObjects. The EMF library provides a *ChangeRecorder* class that can listen to changes in EObjects and returns notifications that include the details of the changes. In this prototype, we only listen to semantic changes which means the listener will only be notified if a property change is fully done, *e.g.,* a modification in the name of a node. The collaboration plugin parses the notification and converts it into a JSON patch based on the operation type of the change (*i.e.,* replace, remove, and addition). The JSON patch includes the operation type, the path that locates the node and the new value.

2) PROPAGATION

The JSON patch is sent to the server if a change is detected. Once it is received, the server first applies the change to the model on its side and publishes it to the subscribers. The published message is again in the format of JSON patch which resembles the JSON patch it receives from clients. In practice, the JSON patch sent to and received from the server are not always exactly the same. Operations such as removal and addition may cause position changes in other nodes. Thus, a single JSON patch sent from the client may eventually lead to a list of JSON patches published from the server to its subscribers and each JSON patch in the list states a single step of change. The propagation function is provided by the EMF cloud server and it ignores the platform of the JSON patch sender and model subscribers. As long as the message the client sends to and receives from the server is in the format of JSON patch, the platform of the client is not restricted.

3) SUBSCRIPTION

The Eclipse collaboration plugin subscribes to the model in the server via WebSocket. It receives JSON patches from the server and applies the changes to the local model in the modeler's editor domain. This is achieved by parsing the JSON patches one by one and for every single patch, the parser locates the position of the node to be changed through the path value in the patch and creates a new EObject with the new value, and attaches it to the position located. From the modeler's perspective, the model in the editor domain changes with new properties on certain nodes in real-time if other clients made a change on their side.

E. COLLABORATION PLUGIN FOR MPS[8]

To enable real-time collaboration on the MPS side, we implemented two functionalities: enable/start a collaboration session, and disable collaboration. Each functionality comprises a sequence of action coded to initiate different sets of logic. These actions are combined into a group called *Collaboration* which is configured to display these actions in the context menu for nodes. A model in MPS is stored in a data structure called node, specifically in a root node. Thus in order to begin collaboration, a modeler right-clicks on the root node and clicks on *Enable collaboration*. This action from the modeler will launch the *Collaboration plugin* in MPS, if it is not already running. Upon the launch of this plugin, three components are fired up one after another: the *synchronizer*, the *mapper*, and the *listener*.

1) SYNCHRONIZER

This component is responsible for ensuring that all aspects of the selected node in MPS are equivalent to that of the model in EMF model server. This implies, that the language structure of the given node complies with the Ecore logic of the EMF model, as well as the content at the start of the collaboration. This is achieved with the following sub-components:

1) Validator: Ensures that the selected node is present in the EMF model server by performing a GET request with the name of the selected node. If available, the *validator* checks whether the language structure of the selected node conforms to the metamodel of the corresponding model in the model server. This is achieved with the help of a *mapper* component. If the validation process fails for any one of the two steps mentioned above, the collaboration is automatically disabled.
2) Mapper: Since the logic of the language structure of MPS and Ecore are packed differently, this component cross-checks the *EStructuralFeatures* and *ESuperTypes*

[8]https://github.com/blended-modeling/MPS-Collaboration-Plugin

of Ecore with the language structure in MPS. Each root node of the language structure of MPS represents a *concept*, which might or might not extend and implement other concepts. A *concept* is referred to as an *EClass* in EMF terminology. In order to store and read Ecore data, the Ecore file is fetched from the server via GET request in JSON format, and stored in data classes with Jackson's Object Mapper.[9] In order to read data from the root nodes of MPS, MPS's Open API, SModel,[10] was used. After ensuring that all EClasses from the EMF metamodel are present in the language structure of MPS by their name, the EStructural features and ESupertypes of each EClass are compared.

3) ContentSynchroniser: After mapping is performed between MPS node and EMF models, the ContentSynchroniser component compares and synchronizes the *content* of the selected node in MPS to that of the EMF model in the model server. If there is any inconsistency found, then the content of the selected node is overwritten to that of the model present in the model server. At the end of this process, a structural map is produced which is a mapping of each node to its location in the model. This structural map is used to find the relevant node when a patch is received from a server in order to apply the relevant change.

With the help of the mentioned sub-components, we were able to generify the structural communication between MPS and Eclipse, such that an introduction of a different meta-model will not cause any additional complications that might require technical intervention to make the framework work.

2) MAPPER

We established the communication between MPS and EMF via an external library, the emfcloud-modelserver.[11] Originally the emfcloud-modelserver has been designed to accommodate EMF logic so we designed our own *mapper* component discussed earlier to interpret and exploit this logic for MPS. Models stored on the model server can be attained via GET requests in various formats, we used JSON format in our work. In order to propagate the changes made during the collaboration session on a given model, the modeler is subscribed to the model via websocket. We receive these edit operations on MPS side as JSON patches. Once subscribed, patch operations can be performed on the models using MPS's SModel language to reflect any change made to the model in the model server locally in MPS.

3) LISTENER

MPS provides a library called Open API[12] which provides controlled access to a given model and also provides interfaces in order to provide a custom implementation in various aspects, in our case for the listener. Among the listeners provided by OpenAPI, we are using SNodeChangeListener in our work. When an edit operation is performed on a given node in MPS, SNodeChangeListener is notified via MPS's message bus and the operation is propagated to the EMF model server. For now, the listener is configured to report changes character by character on MPS side in our work. In the future, we will refine the implementation to receive and propagate the change as a whole.

When the modeler decides to end the collaboration session, (s)he can right click the node involved in their collaboration session and click *Disable collaboration*. With this action, the *Listener* and *EmfModelServer websocket client* are disabled one after another.

Overall, on the implementation side, we encountered a few challenges during the implementation of the collaboration plugins for EMF and MPS, such as setting up the operating environment, researching the mechanism of the EMF model and its cloud server, converting model objects into JSON patches and the other way around, and making the implementation as generic as possible. We experienced MPS environment as a little unstable particularly while using Java source code regarding the ability of the environment to detect the presence of required libraries. This often resulted in crashing for MPS and the only possible solution we could find was reimport of the project. The online help resources of MPS include the MPS's official user guide, the community forums and MPS official slack channel. Due to limited userbase of MPS, finding help regarding technical issues can be a little cumbersome.

VII. EXAMPLE APPLICATION

To demonstrate the functioning of BUMBLE-CE, we apply it in the context of two of the most used modeling platforms, specifically: Eclipse EMF and JetBrains MPS. The example is used as a common basic use case within the BUMBLE European project, which involves the usage of a simple DSML for representing state machines. Below, we explain (i) the *State Machine* metamodel, (ii) the *TrafficSignal.statemachine* model conforming to the State Machine metamodel, and (iii) an example of collaboration session between EMF and MPS.

A. STATE MACHINE DSML

The *State Machine* DSML is initially designed by the Modeling Value Group (MVG) in MPS.[13] We firstly translated the language definition of *State Machine* from MPS to an Ecore-based metamodel. The Ecore metamodel of the *State Machine*

[9]https://fasterxml.github.io/jackson-databind/javadoc/2.7/com/fasterxml/jackson/databind/ObjectMapper.html

[10]https://github.com/JetBrains/MPS/tree/master/core/openapi/source/org/jetbrains/mps/openapi

[11]https://github.com/eclipse-emfcloud/emfcloud-modelserver

[12]https://github.com/JetBrains/MPS/tree/master/core/openapi/source/org/jetbrains/mps

[13]https://github.com/ModelingValueGroup/statemachines

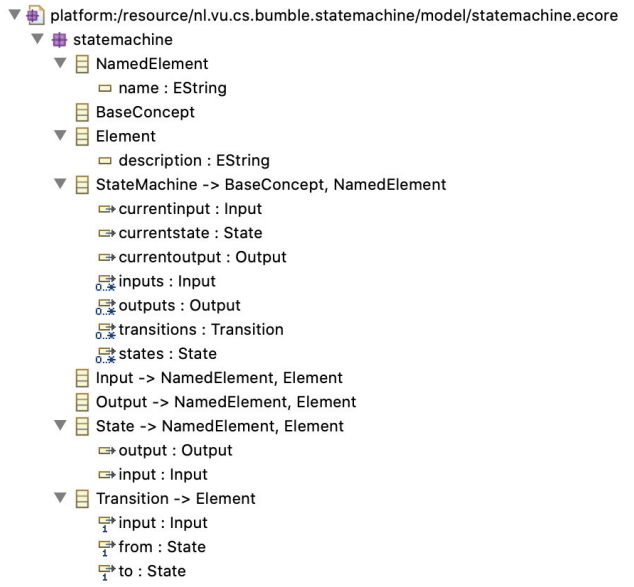


FIGURE 6. Ecore metamodel of *state machine*.

DSML is shown in Figure 6. According to such metamodel, a *State Machine* extends three metaclasses: *BaseConcept*, *NamedElement* and *Element*; also, a state machine can contain sets of instances of the *Input*, *Output*, *State*, and *Transition* metaclasses. The *State Machine* metaclass keeps track of the current input, output and state through the references *currentinput*, *currentoutput* and *currentstate* to the metaclasses *Input, Output* and *State*. The current *State* receives a trigger defined by the metaclass *Input*, transitions to a new *State* and produces a new *Output*. The *Transition* metaclass handles the control flow from one state to another state on receiving an input through references to *Input* and *State* metaclasses. The metaclasses *Input, Output, and State* have attributes *name* and *description*. The metaclass *State* has references to the metaclasses *Input* and *Output*; and the metaclass *Transition* has references to the metaclasses *Input* and *State*. In this way, our metamodel illustrates essential concepts to define a state machine. According to BUMBLE-CE, the modeler has the responsibility of storing the metamodel of the used DSML (*StateMachine* in our case) in the server. We expect that the metamodel of the used DSML is not subjected to changes frequently and storing the metamodel on the server is only a one-time action performed by the BUMBLE-CE admin.

B. THE TRAFFICSIGNAL.STATEMACHINE MODEL

Figure 7 shows our *TrafficSignal.statemachine* model within the tree-based default editor of EMF. The model conforms to the *StateMachine* metamodel and it is used for exemplifying the real-time collaboration among the EMF and MPS editors. The *TrafficSignal.statemachine* model represents the working of a traffic signal with three inputs *Go, Wait and Stop* and three states *Red, Yellow and Green*. When in state *Red* or *Green* and on receiving inputs *Go* or *Stop* respectively, the *TrafficSignal.statemachine* will transition to the state *Yellow*.

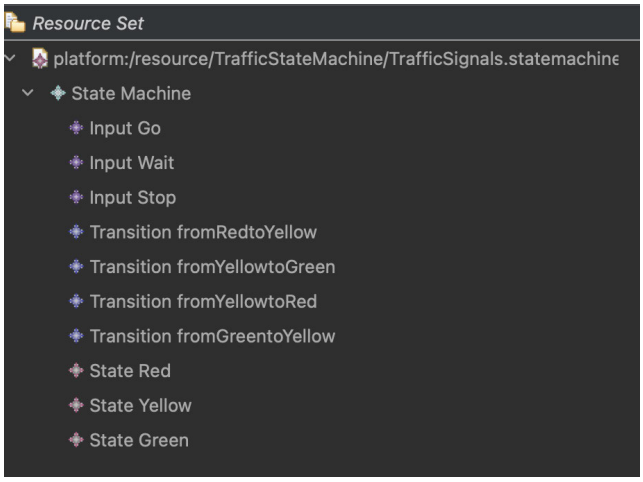


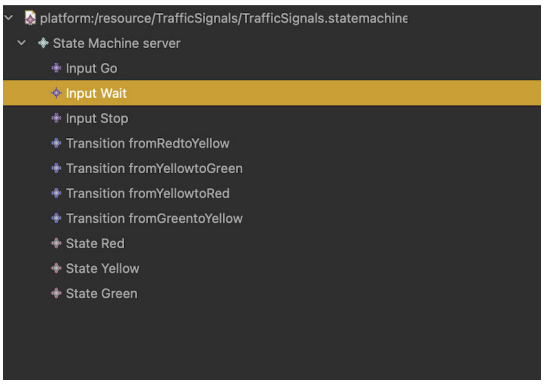
FIGURE 7. The *TrafficSignals.statemachine* model.

When in state *Yellow*, if input *Stop* is received, the *TrafficSignal.statemachine* transitions to the state *Red* and if input *Go* is received, the *TrafficSignal.statemachine* transitions to the state *Green*.

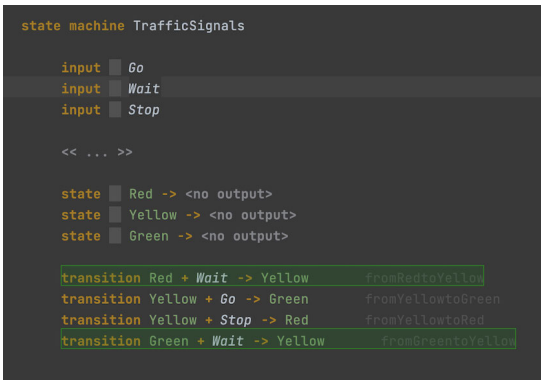
C. EXAMPLE OF REAL-TIME COLLABORATION SESSION

To start the real-time collaboration session, a modeler right clicks on the *TrafficSignal.statemachine* in their respective editor and clicks on the *Start Collaboration* menu item provided by BUMBLE-CE. The *Session Manager* uploads the *TrafficSignal.statemachine* on the server so that other modelers can also join the collaboration session. The model that has been just added on the server acts as a single source of truth for the collaboration session. Specifically, once a modeler performs an operation on the model (for instance, adds or deletes a state, edits the name of a transition, or changes the position of an input in the tree), the change will be propagated to all the other participating editors through the web sockets mechanism provided by EMF.Cloud. Figure 8 shows the collaboration scenario for the *replace* operation between EMF and MPS. The *TrafficSignal.statemachine* is shown for both EMF and MPS before starting the collaboration in Figures 8a and 8b, respectively. The modeler on the EMF side replaces the name of the *Wait* Input with *Hold* and the change is immediately reflected on the MPS side, see Figures 8c and 8d. Figure 8e shows the log history containing all the events happening on the MPS side, where we can observe that the *Replace* operation is successfully received and the intended changes are updated in the model.

The information about the collaboration sessions, participating modelers and the model under the collaboration are displayed on the *Web-based Model Inventory Viewer* as well. We have built custom *Collaboration Plugins* on both MPS and EMF side, explained below, to enable the editors to connect to BUMBLE-CE. The real-time collaboration is supported in both directions, that is, EMF to MPS and vice-versa.

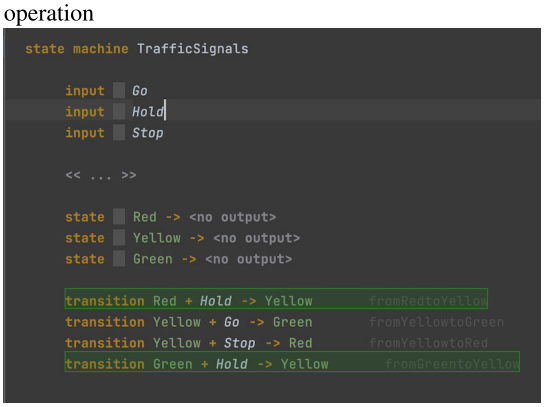


(a) *TrafficSignals.statemachine* in EMF before any edit operation

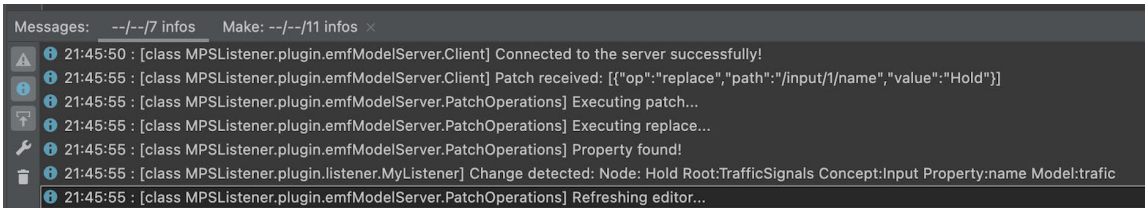


(b) *TrafficSignals.statemachine* in MPS before any edit operation

(c) *TrafficSignals.statemachine* in EMF, the name of Input *Wait* is edited to *Hold*. The operation is propagated to the MPS side.



(d) *TrafficSignals.statemachine* in MPS, the edit operation to change the name of Input *Wait* to *Hold* is received on MPS side and reflected in the model.



(e) Logs in MPS showing that the edit operation is successfully received and propagated in the *TrafficSignals.statemachine*

FIGURE 8. Example scenario of cross-platform (between Eclipse EMF and JetBrains MPS) real-time collaboration using BUMBLE-CE.

VIII. DISCUSSION

Reflection – The complexity of engineered systems is ever increasing, resulting in larger and more diverse models. This increased complexity can be handled by following collaborative MDSE practices. Distributed software development teams increasingly rely on collaboration tools and frameworks for accomplishing a variety of tasks [46], [47], [48]. The generic architecture and its prototype implementation described in this paper takes a step forward towards enhancing this collaboration by facilitating cross-platform real-time collaboration– *i.e.,* providing the flexibility during modeling to accommodate live edits created synchronously by multiple modelers, across different modeling platforms.

We started with eliciting requirements for building BUMBLE-CE. These requirements present the functionalities necessary for collaboration in real-time and across different platforms. However, the requirements have been gathered in the context of the BUMBLE project, so we suggest the reader to consider the list as being not exhaustive or at the very least as needing a customization with respect to project-specific and organizational constraints. Our proposed architecture fulfills these requirements and ensures that the modelers do not need to make modifications in their own tools or frameworks to use BUMBLE-CE. The access of different stakeholders involved in the modeling process to BUMBLE-CE is controlled through authentication mechanisms.

The main strength of BUMBLE-CE is that it supports real-time collaboration while being independent of the modeling platforms and concrete syntaxes chosen by the modelers. BUMBLE-CE provides an extensible framework which

enables MDSE engineers to add external plugins through available extension points to incorporate specific logic or mechanism needed for a particular software project. For instance, the modelers can choose a particular model validator or a persistency controller driver for their purposes. However, BUMBLE-CE is still a work in progress project. Below we report lines of activities for improving the project, both on implementation and research side.

Limitations and Needed Improvements

- *Conflict resolution:* When modelers work together on a model and perform edit operations, conflicting requests become inevitable due to the concurrent changes made by the different modelers. Conflict management is a main concern for tools and frameworks facilitating collaborative modeling. However, for BUMBLE-CE we process the edit requests sequentially *i.e.,* in order of arrival. In this way, the success or failure of a request is based on the current situation of the model. This approach for conflict management allows automatic resolution of all requests and enables BUMBLE-CE to handle all message requests with minimal processing which results in improved responsiveness. In future, we also aim to incorporate existing conflict management techniques for handling concurrent changes occurring during real-time collaboration sessions managed by BUMBLE-CE [41].
- *Multi-view modeling:* Collaborative modeling is a joint effort of several modelers to create a representation of those parts of the software system that they consider relevant for the modeling goal. Different modelers working on a shared model to collaborate on a larger project may focus on different parts of the same model. Similarly, modelers may desire to view a model for varying purposes. A specific view may contain only a portion of a model or utilize a distinct visualization. Multi-view modeling is, therefore, an important feature for collaborative modeling tools and has been studied extensively [31], [49], [50]. Currently, BUMBLE-CE does not support multi-view modeling functionality.
- *Workspace awareness:* Workspace awareness refers to the ability of a tool to allow modelers to see what other collaborators are doing in real-time. Workspace awareness is an important aspect of collaborative modeling because it allows users to work together more effectively by providing transparency and visibility into the actions and contributions of other users [51]. Currently, BUMBLE-CE does not provide awareness of each others actions to the modelers. In the future, we are planning to add workspace awareness to our collaboration engine such as, include seeing the cursor of other users, viewing the changes, comments and annotations made by other users in real-time.
- *Improving usability of BUMBLE-CE:* There are some improvements needed on the usability side of the collaboration plugins. A dedicated *Model Inventory Viewer* is yet to be developed for both EMF and MPS. This will allow the modelers to choose a model for collaboration among a collection of models available for collaboration in their inventory. Being based on the same data stored in the central model inventory, platform-specific inventory viewers will be synchronized with the generic BUMBLE-CE *Web-based Model Inventory viewer*.
- *Support for more operations:* For now, BUMBLE-CE only supports basic operations on models, *i.e.,* add, replace and delete. Adding more operations such as moving or copying modeling elements will enrich the functionality of BUMBLE-CE and will enhance the usability of the tool.
- *Support for more editors:* BUMBLE-CE currently supports tree-based editors. The prototype tool can be extended to support graphical, textual and more editors. We have picked up on providing a support for the graphical editors in near future so that a real-time collaboration across different concrete syntaxes can be made available for modelers.
- *Evaluation of proposed approach:* In this paper, we have provided an example application of BUMBLE-CE by showing collaboration between Eclipse EMF and JetBrains MPS for a state machine model, *TrafficSignals.statemachine*. However, we have not yet evaluated BUMBLE-CE through a case study or a large-scale evaluation. We realize that scalability in collaborative modeling has many facets, including the ability to collaborate on larger models and supporting a large number of modelers for real-time collaboration sessions. In this respect, the performance of our collaboration engine is also yet to be analyzed in terms of scalability. In the future, we aim to perform user studies on both academic and industrial level which will help us to assess the scalability and performance of BUMBLE-CE.
- *Undo functionality:* BUMBLE-CE does not support undo functionality. Storing the operations performed by a modeler in a stack will facilitate the modeler to perform undo operations across platforms.

IX. CONCLUSION

Cross-platform real-time collaboration is becoming increasingly relevant due to the availability of different modeling platforms. In this work, we elaborated on the requirements, architecture, in-progress implementation and an example application of our generic solution for achieving real-time collaboration across different platforms – *i.e.,* BUMBLE-CE. To the best of our knowledge, there is no prior work done for facilitating *cross-platform* real-time collaboration. BUMBLE-CE has been designed to be independent of both the modeling platforms and the DSML used by the modelers. BUMBLE-CE facilitates the real-time collaboration through a set of BUMBLE-CE plugins which are loaded into the modeling tools of the modelers with an aim to minimally impact the modeling tool. The modelers can also view an always-updated overview of the existing models in

the project, current collaboration sessions, active participants in the collaboration sessions and more through a web-based model inventory viewer. The implementation for BUMBLE-CE is still in progress and needs more functionalities to be added so that our prototype is in line with the proposed architecture. We also elaborated on these limitations and areas of improvement in the paper.

Future work – In the near future, we want to incorporate more editors to BUMBLE-CE so that we can uncover the effort needed for adding different types of editors as collaboration parties to our framework. We further plan to evaluate the performance and scalability of our collaboration engine by conducting user studies with both academic and industrial participants. We find that efficient conflict resolution mechanisms and high workspace awareness greatly impact the user experience when modelers are collaborating in real-time with each other. As our implementation is available online, interested researchers can pursue these lines of research and extend the current functionality of BUMBLE-CE with new plugins providing their own new contributions to the platform.

REFERENCES

[1] D. C. Schmidt, "Model-driven engineering," *IEEE Comput. Soc.*, vol. 39, no. 2, p. 25, Feb. 2006.

[2] C. A. González and J. Cabot, "Formal verification of static software models in MDE: A systematic review," *Inf. Softw. Technol.*, vol. 56, no. 8, pp. 821–838, Aug. 2014.

[3] I. Mistrík, J. Grundy, A. V. D. Hoek, and J. Whitehead, "Collaborative software engineering: Challenges and prospects," in *Collaborative Software Engineering*. Berlin, Germany: Springer, 2010, pp. 389–403.

[4] D. Di Ruscio, M. Franzago, I. Malavolta, and H. Muccini, "Envisioning the future of collaborative model-driven software engineering," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2017, pp. 219–221.

[5] M. Franzago, D. D. Ruscio, I. Malavolta, and H. Muccini, "Collaborative model-driven software engineering: A classification framework and a research map," *IEEE Trans. Softw. Eng.*, vol. 44, no. 12, pp. 1146–1175, Dec. 2018.

[6] I. David, K. Aslam, S. Faridmoayer, I. Malavolta, E. Syriani, and P. Lago, "Collaborative model-driven software engineering: A systematic update," in *Proc. ACM/IEEE 24th Int. Conf. Model Driven Eng. Lang. Syst.*, Oct. 2021, pp. 273–284.

[7] I. David, K. Aslam, I. Malavolta, and P. Lago, "Collaborative model-driven software engineering—A systematic survey of practices and needs in industry," *J. Syst. Softw.*, vol. 199, May 2023, Art. no. 111626.

[8] L. Hattori, "Enhancing collaboration of multi-developer projects with synchronous changes," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.*, May 2010, pp. 377–380.

[9] F. Budinsky, R. Ellersick, D. Steinberg, T. J. Grose, and E. Merks, *Eclipse Modeling Framework: A Developer's Guide*. Reading, MA, USA: Addison-Wesley, 2004.

[10] F. Campagne, *The MPS Language Workbench*. vol. 1. Saint Joseph, Réunion: Fabien Campagne, 2014.

[11] J.-P. Tolvanen and M. Rossi, "MetaEdit+: Defining and using domain-specific modeling languages and code generators," in *Proc. Companion 18th Annu. ACM SIGPLAN Conf. Object-Oriented Program., Syst., Lang., Appl.*, Oct. 2003, pp. 92–93.

[12] 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," in *Proc. 40th Int. Conf. Softw. Eng., Softw. Eng. Pract.*, 2018, pp. 225–234.

[13] P. Nicolaescu, M. Derntl, and R. Klamma, "Browser-based collaborative modeling in near real-time," in *Proc. 9th IEEE Int. Conf. Collaborative Comput., Netw., Appl. Worksharing*, 2013, pp. 335–344.

[14] (Feb. 2021). *Modelix and the Future of Language Engineering*. [Online]. Available: https://blogs.itemis.com/en/modelix-and-the-future-of-language-engineering

[15] M. Voelter and K. Solomatov, "Language modularization and composition with projectional language workbenches illustrated with MPS," *Softw. Lang. Eng.*, vol. 16, no. 3, pp. 1–10, 2010.

[16] L. C. Kats and E. Visser, "The Spoofax language workbench," in *Proc. SPLASH/OOPSLA Companion*, 2010, pp. 237–238.

[17] (May 2021). *Visual Studio Live Share: Visual Studio*. [Online]. Available: https://visualstudio.microsoft.com/services/live-share/

[18] A. Cicchetti, F. Ciccozzi, and A. Pierantonio, "Multi-view approaches for software and system modelling: A systematic literature review," *Softw. Syst. Model.*, vol. 18, no. 6, pp. 3207–3233, Dec. 2019.

[19] *BUMBLE Project Official Webpage*. Accessed: Oct. 14, 2022. [Online]. Available: https://itea3.org/project/bumble.html

[20] I. David, M. Latifaj, J. Pietron, W. Zhang, F. Ciccozzi, I. Malavolta, A. Raschke, J.-P. Steghöfer, and R. Hebig, "Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study," *Softw. Syst. Model.*, vol. 22, no. 1, pp. 415–447, Feb. 2023.

[21] K. Riemer and F. Frößler, "Introducing real-time collaboration systems: Development of a conceptual scheme and research directions," *Commun. Assoc. Inf. Syst.*, vol. 20, pp. 204–225, Dec. 2007.

[22] Y. Sun, D. Lambert, M. Uchida, and N. Remy, "Collaboration in the cloud at Google," in *Proc. ACM Conf. Web Sci.*, New York, NY, USA, Jun. 2014, p. 239, doi: 10.1145/2615569.2615637.

[23] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages*. New York, NY, USA: Pearson Education, 2009.

[24] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering*. San Rafael, CA, USA: Morgan Claypool, 2012.

[25] (1997). *OMG/MOF Meta Object Facility (MOF) Specification. OMG Document AD/97-08-14*. [Online]. Available: http://www.omg.org/

[26] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: Eclipse Modeling Framework*. London, U.K.: Pearson Education, 2008.

[27] *EMF.Cloud—Evolve Your Modeling Tools to the Web!* Accessed: Jun. 27, 2022. [Online]. Available: https://www.eclipse.org/emfcloud/

[28] J. Helming, M. Koegel, and P. Langer. *The EMF.Cloud Model Server*. Accessed: Jun. 27, 2022. [Online]. Available: https://eclipsesource.com/blogs/2021/02/25/the-emf-cloud-model-server/

[29] C. Debreceni, G. Bergmann, M. Búr, I. Ráth, and D. Varró, "The MONDO collaboration framework: Secure collaborative modeling over existing version control systems," in *Proc. 11th Joint Meeting Found. Softw. Eng.*, Aug. 2017, pp. 984–988.

[30] J. Corley, E. Syriani, and H. Ergin, "Evaluating the cloud architecture of AToMPM," in *Proc. 4th Int. Conf. Model-Driven Eng. Softw. Develop.*, 2016, pp. 339–346.

[31] M. Maróti, T. Kecskés, R. Kereskényi, B. Broll, P. Völgyesi, L. Jurácz, T. Levendovszky, and A. Lédeczi, "Next generation (meta) modeling: Web-and cloud-based collaborative tool infrastructure," *MPM@ MoDELS*, vol. 1237, pp. 41–60, 2014.

[32] J. L. Cánovas Izquierdo and J. Cabot, "Collaboro: A collaborative (meta) modeling tool," *PeerJ Comput. Sci.*, vol. 2, p. e84, Oct. 2016.

[33] S. N. Voogd, K. Aslam, L. Van Gool, B. Theelen, and I. Malavolta, "Real-time collaborative modeling across language workbenches—A case on JetBrains MPS and Eclipse Spoofax," in *Proc. ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst. Companion (MODELS-C)*, Oct. 2021, pp. 16–26.

[34] D. Wüest, N. Seyff, and M. Glinz, "FlexiSketch: A lightweight sketching and metamodeling approach for end-users," *Softw. Syst. Model.*, vol. 18, no. 2, pp. 1513–1541, Apr. 2019.

[35] J. Bézivin, F. Jouault, and P. Valduriez, "On the need for megamodels," in *Proc. Best Practices Model-Driven Softw. Develop. Workshop, 19th Annu. ACM Conf. Object-Oriented Program., Syst., Lang., Appl.*, 2004, pp. 1–9.

[36] G. Blair, N. Bencomo, and R. B. France, "Models@ run.time," *Computer*, vol. 42, no. 10, pp. 22–27, 2009.

[37] R. Molich and J. Nielsen, "Improving a human–computer dialogue," *Commun. ACM*, vol. 33, no. 3, pp. 338–348, Mar. 1990.

[38] *Accessibility*. Accessed: Jun. 27, 2022. [Online]. Available: https://www.w3.org/standards/webdesign/accessibility

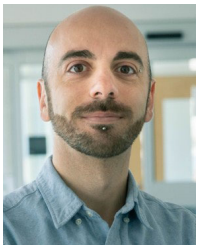
[39] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *Proc. Symp. Self-Stabilizing Syst.* Cham, Switzerland: Springer, 2011, pp. 386–400.

[40] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen, "Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems," *ACM Trans. Comput.-Hum. Interact.*, vol. 5, no. 1, pp. 63–108, Mar. 1998.
[41] M. Sharbaf, B. Zamani, and G. Sunyé, "Conflict management techniques for model merging: A systematic mapping review," *Softw. Syst. Model.*, vol. 2022, pp. 1–49, Jan. 2022.
[42] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Reading, MA, USA: Addison-Wesley, 2003.
[43] E. Saks, "JavaScript frameworks: Angular vs react vs Vue," Bachelor's Thesis, Haaha-Helia, Univ. Appl. Sci., Finland, 2019.
[44] E. Wohlgethan, "Supporting web development decisions by comparing three major JavaScript frameworks: Angular, React and Vue.js," Ph.D. dissertation, Dept. Comput. Sci., Hochschule für Angewandte Wissenschaften, Hamburg, Germany, 2018.
[45] P. Halliday, *Vue.js 2 Design Patterns and Best Practices: Build Enterprise-ready, Modular Vue.js Applications With Vuex and Nuxt*. Birmingham, U.K.: Packt Publishing, 2018.
[46] N. A. Moketar, M. Kamalrudin, S. Sidek, M. Robinson, and J. Grundy, "An automated collaborative requirements engineering tool for better validation of requirements," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng.*, Aug. 2016, pp. 864–869.
[47] S. Oppl, "Articulation of work process models for organizational alignment and informed information system design," *Inf. Manage.*, vol. 53, no. 5, pp. 591–608, Jul. 2016.
[48] J. L. Pérez-Medina and J. Vanderdonckt, "Sketching by cross-surface collaboration," in *Proc. Int. Conf. Inf. Technol. Syst.* Cham, Switzerland: Springer, 2019, pp. 386–397.
[49] B. Combemale, O. Barais, and A. Wortmann, "Language engineering with the GEMOC studio," in *Proc. IEEE Int. Conf. Softw. Archit. Workshops (ICSAW)*, Apr. 2017, pp. 189–191.
[50] E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. Van Mierlo, and H. Ergin, "AToMPM: A web-based modeling environment," in *Proc. 16th Int. Conf. Model Driven Eng. Lang. Syst.*, Oct. 2013, pp. 21–25.
[51] C. Gutwin and S. Greenberg, "The mechanics of collaboration: Developing low cost usability evaluation methods for shared workspaces," in *Proc. IEEE 9th Int. Workshops Enabling Technol., Infrastruct. Collaborative Enterprises*, Mar. 2000, pp. 98–103.

KOUSAR ASLAM (Member, IEEE) received the Ph.D. degree in computer science from the Eindhoven University of Technology. She is currently a Postdoctoral Researcher with Vrije Universiteit Amsterdam, The Netherlands. Her research interests include blended modeling, software evolution, and model-based re-engineering. She is a member of VERSEN, EUGAIN, and Amsterdam Data Science.

YU CHEN received the bachelor's degree in computer science from Vrije Universiteit Amsterdam, The Netherlands, where she is currently pursuing the master's degree in computer science. Her research interest includes front-end engineering.

MUHAMMAD BUTT is currently pursuing the bachelor's degree in computer science with Vrije Universiteit Amsterdam. His experience as of this moment is more orientated toward back-end engineering.



IVANO MALAVOLTA (Member, IEEE) received the Ph.D. degree in computer science from the University of L'Aquila, in 2012. He is currently an Associate Professor and the Director of the Network Institute, Vrije Universiteit Amsterdam, The Netherlands. He has authored more than 150 peer-reviewed scientific articles in international journals and conferences proceedings. His research interests include data-driven software engineering, with a special emphasis on software architecture, mobile software development, robotics software, and model-driven engineering. He is a program committee member and a reviewer of international conferences and journals in the software engineering and robotics fields. He is a member of ACM, VERSEN, and Amsterdam Data Science.