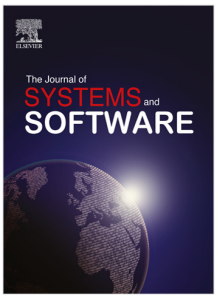
Journal Pre-proof



CoMPers: A configurable conflict management framework for personalized collaborative modeling

Mohammadreza Sharbaf, Bahman Zamani, Gerson Sunyé

PII: S0164-1212(24)00271-1
DOI: https://doi.org/10.1016/j.jss.2024.112227
Reference: JSS 112227

To appear in: *The Journal of Systems & Software*

Received date : 24 May 2024
Revised date : 22 August 2024
Accepted date : 23 September 2024

Please cite this article as: M. Sharbaf, B. Zamani and G. Sunyé, CoMPers: A configurable conflict management framework for personalized collaborative modeling. *The Journal of Systems & Software* (2024), doi: https://doi.org/10.1016/j.jss.2024.112227.

CoMPers: A Configurable Conflict Management Framework for Personalized Collaborative Modeling

Mohammadreza Sharbaf[a,*], Bahman Zamani[a] and Gerson Sunyé[b]

[a]MDSE Research Group, University of Isfahan, Isfahan, Iran
[b]LS2N, University of Nantes, Nantes, France

ARTICLE INFO

Keywords:
Model-Driven Engineering
Collaborative Modeling
Change Propagation
Conflict Management
Personalized Collaboration

ABSTRACT

Context: Modeling is an activity in the software development life cycle where experts and stakeholders collaborate as a team. In collaborative modeling, adhering to the optimistic versioning paradigm allows users to make concurrent changes to the same model, but conflicts may arise. To achieve an integrated and consistent merged model, conflicts must be resolved. **Objective:** The primary objective of this study was to provide a customizable and extensible framework for conflict management in personalized change propagation during collaborative modeling. **Method:** We propose CoMPers, a customizable and extensible conflict management framework designed to address various conflicts encountered in collaborative modeling. We present the duel algorithm for automatically detecting and resolving conflicts according to user preferences. The framework utilizes personalized change propagation to customize collaboration and supports the conflict management process by executing the duel algorithm based on user preferences. As a proof-of-concept, we have implemented the CoMPers framework and extended the EMF.cloud modeling framework to demonstrate its applicability. **Results:** We have constructed a proof-of-concept implementation and conducted a real-world case study, a benchmark experiment, and a user experience evaluation. Our findings demonstrate that: (1) CoMPers enables collaborators to configure propagation strategies according to their habits; (2) CoMPers successfully identifies all anticipated conflicts and achieves a 100% accuracy in conflict handling; (3) The majority of participants agreed that CoMPers is user-friendly for collaborative modeling. **Conclusion:** This paper presents the CoMPers framework, which is based on personalized change propagation, and helps collaborators customize conflict management activities. The results confirm the feasibility and advantages of consistent and concurrent modeling within the collaborative CoMPers platform, with an acceptable functionality for approximately ten collaborators.

1. Introduction

Models are becoming primary artifacts that are used to develop modern systems in the engineering domain [1]. When the software is complex, the size and complexity of models increase dramatically. This enforces that many engineers and stakeholders participate in large teams and collaborate to evolve models [2]. In this context, some of the participants may work concurrently and independently on the same model from different geographical sites. Each participant focuses on specific aspects of the system and locally modifies only a particular part of the model. Having engineers from diverse domains, the collaborative framework should be able to customize the collaboration scenario and adapt the modeling environment based on the needs and habits of the engineers [3].

Model-Driven Engineering (MDE) is a software engineering discipline promoting models as first-class artifacts of the software lifecycle [4]. In MDE project, models are expressed in the right modelling formalism that is suitable for the task at hand [5]. Therefore, MDE is a potential solution can involve developers with a diverse set of expertise to help design system collaboratively [3]. To deal with developers from different domains and expertise, it is important to provide a proper collaborative modeling framework that allows collaborators to personalize their collaboration and environment according to their needs [6]. There are two types of collaborations: offline and online. In the former, engineers check out a model from a repository and send back their local changes. In the latter, engineers may simultaneously edit a model, and immediately propagate to others.

However, the foundational issue is that one cannot simply just outsource the decision between online and offline to the system managers, because preserving the user intention in collaboration would become practically infeasible. Therefore, mixing online and offline collaboration can be beneficial for collaborators with diverse needs and preferences. In previous work [7], we introduced the PCP approach, which proposed the initial idea of combining personalized change propagation with both online and offline collaboration. The PCP approach incorporates two patterns to effectively support sending and receiving different changes based on collaborators' needs in collaborative modeling.

To address the challenges of managing conflicts in this hybrid approach, robust mechanisms are essential. These mechanisms include conflict detection and resolution, change tracking, and personalized change propagation. Techniques and tools for conflict management are not optional, but a necessity to cope with the growing number of conflicts and to achieve an integrated yet consistent merged model [2]. Conflict management in collaborative modeling

*Corresponding author
m.sharbaf@eng.ui.ac.ir (M. Sharbaf); zamani@eng.ui.ac.ir (B. Zamani); gerson.sunye@univ-nantes.fr (G. Sunyé)
ORCID(s): 0000-0001-5113-7689 (M. Sharbaf); 0000-0001-6424-1442 (B. Zamani); 0000-0001-6407-8075 (G. Sunyé)

deals with techniques, activities, and tools for enhancing consistency in the resulting model during collaboration [8].

This paper provides a comprehensive overview of the CoMPers conflict management framework, introducing its layered architecture and techniques for supporting and customizing conflict management within personalized change propagation. Within the CoMPers framework, we propose a unified collaboration scheme and extend the PCP approach [7] through the introduction of new extensions for the customization of basic collaboration actions, as well as the definition of conflict patterns and preferences to effectively handle conflicts in collaborative modeling. The conflict management layer within CoMPers enables the specification of conflict patterns and the configuration of conflict management based on user preferences. It also executes conflict detection, resolution, and awareness through the tuning of the duel algorithm according to user configurations aligned with the PCP approach. This paper focuses on presenting a novel collaboration scheme and duel algorithm to support personalized change propagation and configurable conflict management within the CoMPers framework. We show that CoMPers can be tuned under some set of preferences to detect and resolve conflict based on the mentioned techniques. To this end, we provide a comprehensive explanation of the CoMPers architecture and demonstrate the extensibility of its modules through a series of implementations.

The CoMPers framework should fundamentally support change propagation, model integrity, and consistency in parallel collaborative modeling. To assess these capabilities, we begin by implementing a proof of concept of the CoMPers framework specifically designed for Ecore models. This implementation involves the development of the CoMPers collaboration server and an extension of the EMF.cloud modeling environment. Then, we demonstrate the applicability of our collaboration scheme using the Wind Turbine motivating example [9, 10]. We also assess the PCP operations involved in creating and initializing models, as well as allowing users to configure propagation strategies. Moreover, we utilize the benchmark presented in the AMOR project [11] to evaluate the conflict detection and resolution components. Furthermore, we conducted a user experience evaluation workshop with modeling experts to investigate the usefulness and ease of use of the CoMPers framework in a real-world collaboration case. The results indicate that conflicts are effectively managed, and CoMPers serves as an appropriate platform for collaboration and change propagation in models.

The rest of this paper is organized as follows. Section 2 introduces a motivating example to illustrate the envisioned idea. Section 3 provides the background foundations on collaborative modeling and conflict management. Section 4 elaborates on the design and execution of the CoMPers framework. We first describe the general collaboration scheme used in the CoMPers framework. Then, we present the overview of applying personalized change propagation approach in CoMPers. Finally, we outline the detailed architecture of configurable conflict management layer and defines the duel algorithm to handle conflicts within the personalized change propagation platform. Section 5 discuss on implementation of CoMPers. Section 6 describes the evaluation of our approach, while Section 7 overviews related works. Finally, Section 8 concludes the paper and highlights direction for future work.

2. Motivating Example

This section demonstrates the need for personalized change propagation and conflict management through a motivating collaborative modeling example. In particular, Fig. 1 displays a simple model of the cooling control subsystem in the Wind Turbine case study [9, 10]. The subsystem consists of *input* temperature sensors, *output* signals, *controller* units for managing fans and brakes, and *system parameters* that specify temperature limits for initiating the cooling system. The design of control units for cooling the generator is associated with the specifications provided by three stakeholders from different domains: *WT Manager*, *IO Manager*, and *System Manager*. The *WT Manager* defines the system and controllers that determine the cooling controller subsystem for a wind turbine. The role of the *IO Manager* is mainly to specify the input and output objects that connect the cooling controller subsystem to the wind turbine generator. Lastly, the *System Manager* is responsible for defining system parameters and modifying the properties of existing controllers. In such cases, stakeholders with various habits and requirements work on models from different sites and situations. For example, the *IO Manager* may collaborate while sitting on the train.

To achieve this goal, a collaboratvie modeling framework should allow the stakeholders to personalize the propagation of change operations on each model resources according to their needs and habits. Stakeholders should be allowed to choose when to receive approved changes. For instance, on the one hand, the *WT Manager* and *System Manager* like to immediately receive changes made by others and select the online method. On the other hand, the *IO Manager* prefers to become aware of the new changes when others have finished their work or at a specific time and selects the offline method. Moreover, stakeholders should be able to request the latest common version of a model at any time. This feature is useful for new collaborators as well as stakeholders who have been disconnected for a while. For instance, the *IO Manager* can request new changes when connected to the internet network. Furthermore, stakeholders can send their changes to others or delay sending them until they have reached a certainty. In our example, the *System Manager* decides to send their changes with online and synchronous transactions, but the *WT Manager* and *IO Manager* select the offline and asynchronous submission.

Figure 2 depicts a snapshot of the collaboration between stakeholders in our motivating example at timestamp 3. Timestamp is a simple logical clock used to apply or

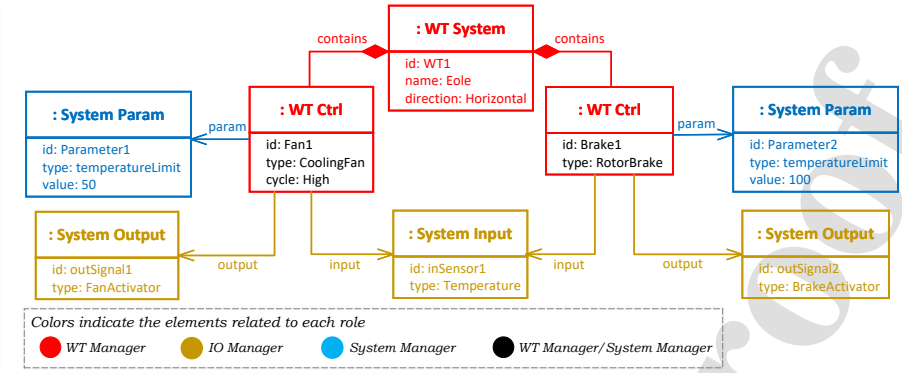


Figure 1: Original Model of Cooling Control Subsystem in the Wind Turbine Case Study

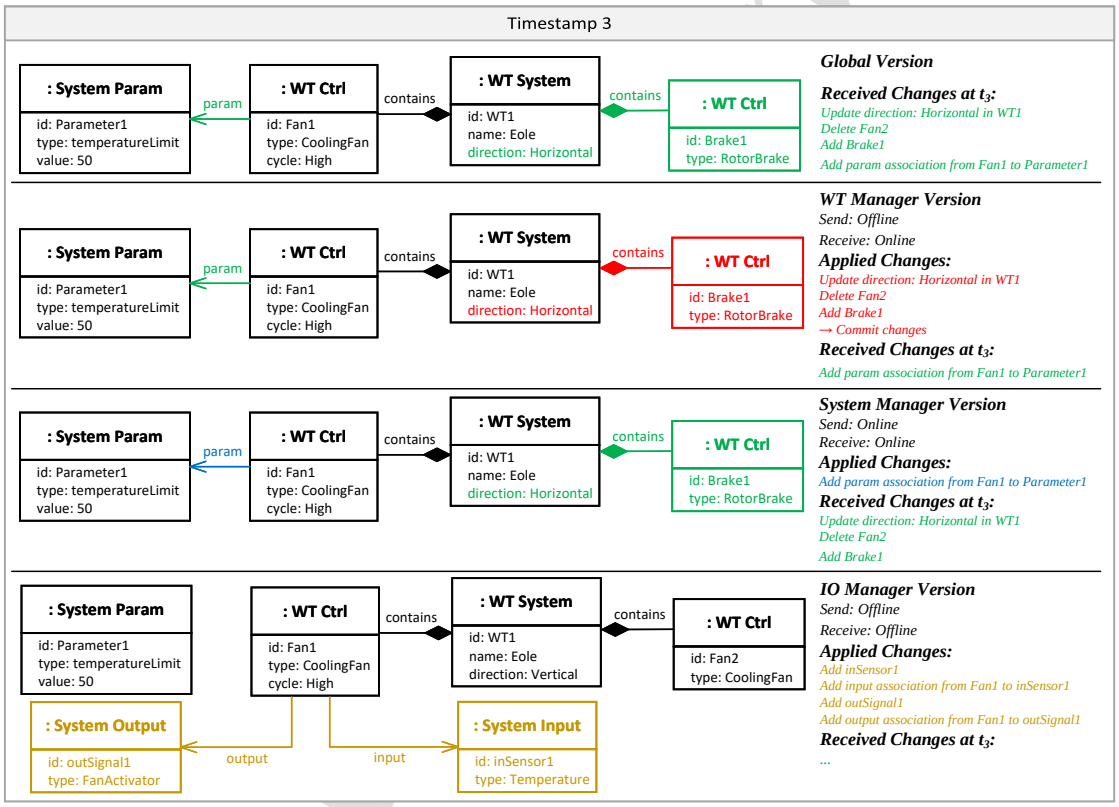


Figure 2: Personalized Change Propagation in Timestamp 3 of the Wind Turbine Case Study

propagate collaborators' change operations. At timestamp 3, collaborators have received changes applied in timestamps 1 and 2, based on their chosen receive methods. Each change operation begins with an atomic operator and follows with the specification of the element under the change. For example, "Add input association from Fan1 to inSensor1" defines a new association relationship named "input" is added from class "Fan1" to class inSensor1. Here, the *WT Manager*, who follows an offline sending method, decides to publish thier modifications. The *WT Manager* then commits their change operations, which include updating the direction of WT1 to horizontal, deleting Fan2, and adding Brake1. Concurrently, the *System Manager*, who works based on the online sending and receiving methods, adds param association to Fan1. All of these changes are immediately received by the server, applied on the global

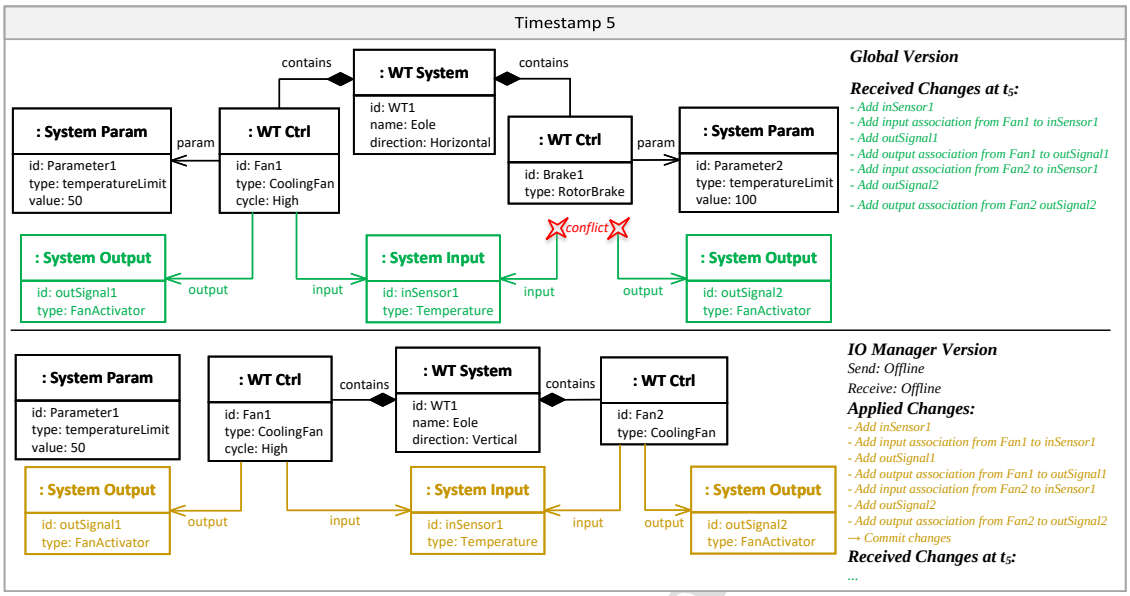


Figure 3: Example of Conflict in Timestamp 5 of the Wind Turbine Case Study

model, and finally published for stakeholders who decide to receive changes in online manner. Therefore, the *System Manager* receives three change operations applied by the *WT Manager*. Similary, the *WT Manager* receives one change operation published by the *System Manager*. However, the *IO Manager* doesn't receive any change operations since the *IO Manager* collaborates offline.

Having several stakeholders working on the same model independently, it is possible that two collaborators concurrently modify the model in different ways, which can lead to conflicts. For example, if the *IO Manager* applies two change operations (adding input and output associations for Fan2) in parallel and offline, these changes may contradict the change performed by the *WT Manager* (deleting Fan2). In this case, the model will have conflicts due to the application of both changes. Figure 3 shows an example of such conflict occurring when the server tries to integrate the received changes at timestamp 5. Therefore, the system should be able to detect and resolve such conflict situation in order to keep the model consistent for all stakeholders. To manage the mentioned conflict, the collaboration framework can simply reject the contradicting change operations applied by the *IO Manager* to maintain consistency. Consequently, in addition to supporting personalized publishing of change operations between collaborators, a suitable collaboration environment has to be able to identify contradicting modifications to prevent the spread of discordance among collaborators.

3. Background

This section introduces the foundation of collaborative modeling and describes the concept of conflict management in model merging, providing a comprehensive guide for the rest of this paper.

3.1. Collaborative Modeling

Model-Driven Software Engineering (MDSE) [4] promotes the migration from code-centric to model-based development, where models are used as first-class entities throughout the entire software engineering life cycle [12]. Similar to the code-centric approach, the model-based development of complex software-intensive systems requires stakeholders with different backgrounds and skills to collaborate on various aspects. This collaboration results in the software models that define the specifications, architecture, and design of the system. In this context, the joint creation of models is considered collaborative modeling, which depends on a set of collaborative means such as model versioning systems and model merging mechanisms that allow stakeholders to coordinate as a team [2, 13].

In collaborative modeling, change propagation occurs in an offline or online manner [5, 14]. Offline collaboration is based on asynchronous interactions. In this scenario, users check-out models from a version control system (VCS) and commit local changes to the repository. In the online scenario, users work synchronously and may simultaneously edit a model and share their changes together. While changes are propagated in both offline and online manners, the support of merge process is crucial in order to obtain a consistent, integrated version for concurrent evolved software models [15].

However, conflicts may arise in the model merging process when concurrent incompatible modifications directly or indirectly affect the same model element. Conflicts may also

happen due to a set of different modifications with the same intention. For instance, a dangling reference is a conflict in UML class diagrams that occurs when one modeler adds an association to a class, whereas another modeler removes that class [16]. In such situations, either the modifications cannot be integrated to produce a unique model, or the integration would result in an inconsistent merged version of the model. Therefore, collaborative modeling environments must properly support conflict management in the merging process.

3.2. Conflict Management

Conflict management is an integral part of the model merging process in collaborative software development. It encompasses various techniques, activities, and tools aimed at improving consistency in the final merged result. One key aspect is conflict detection, where potential conflicts are identified. Additionally, conflict awareness helps alert users to these potential conflicts, ensuring they are mindful of them. Finally, conflict resolution involves addressing and resolving any conflicts that have been identified [16].

Conflicts in model merging can arise from *overlapping changes* and *violations* [17]. Conflicts due to overlapping changes result from the existence of contradicting changes (e.g., renaming a class with two different names) or modifications that express the same meaning in varying ways (e.g., state composition in UML state machines). Overlapping changes can easily give rise to false situations and should not be performed concurrently in the merged model. In comparison, conflicts due to violations result from applying changes that impact the conformance of the model to its well-formedness rules (e.g., dangling reference [18]) or violate the validation rules of modeling languages (e.g., inheritance cycle [18]). Apart from the given conflict reasons, different types of model merging conflicts can also be categorized into syntactic and semantic conflicts, depending on whether they relate to syntax or semantics of the models [19]. We omit more details for the sake of conciseness.

In this regard, Sharbaf et al. [8] have proposed a taxonomy for conflict management techniques, which outlines the optional and mandatory aspects of conflict management approaches for model merging. According to this taxonomy, a valid conflict management approach must incorporate all mandatory aspects, including “Conflict Specification”, “Conflict Detection”, and “Conflict Resolution”. Additionally, “Conflict Awareness” and “Conflict Prevention” are optional aspects that can assist in notifying users and reducing the occurrence of conflicts during the merging process. In the following, we will provide a brief overview of the techniques specified for three mandatory aspects.

Conflict specification is used to describe the conflict situation and the conditions under which conflicts occur. A conflict can be specified using *pattern-based* techniques, *formalism*, and *natural language*. Moreover, some approaches involve an *implicit specification* where conflicts are self-defined using implicit relationships in conflict detection techniques.

Conflict detection is used to discover consistency violations or detect conflict situations that arise in the model merging process. Sharbaf et al. [8] have classified the possible techniques for conflict detection into four categories: 1) *Pattern matching* techniques that search for the presence of a set of model elements as a conflict pattern in the merge process. 2) *Constraint violation* techniques that detect conflicts by checking the validity and conformance of merged models, based on well-formedness rules for models and metamodels. 3) *Change overlapping* techniques that discover conflicts by checking for contradicting changes that concurrently modify a model element in different ways. 4) *Formal methods* techniques that discover conflicts based on formal logic and analysis.

Conflict resolution focuses on techniques that are used to resolve the conflict during the merging process or fix inconsistency in the merged model. Most of existing conflict management approaches follow a *Manual* resolution, in which the resolution phase is delegated to the users once a conflict is detected. Other approaches proposed *Semi-automatic* resolution techniques that attempt to repair conflicts automatically and interact with the user for discordant changes or further information. A few approaches have introduced *Automatic* resolution techniques that fully automate the resolution phase by applying specific rules or performing operational transformations.

4. The CoMPers Framework

In this section, we present the CoMPers framework for managing conflicts within personalized collaborative modeling. Figure 4 provides an overview of the layered architecture of the CoMPers framework, comprising three distinct layers:

- **Layer 1. Collaboration Scheme:** This layer defines a scheme for collaborative modeling, designed to accommodate multiple collaborators who concurrently review and modify a model.
- **Layer 2. Personalized Change Propagation:** This layer introduces personalized change propagation, specifying how it implements the collaboration scheme defined in Layer 1.
- **Layer 3. Conflict Management:** This layer outlines the CoMPers conflict management architecture, encompassing techniques that support essential components of conflict management.

In the following, we delve into the details of each layer.

4.1. CoMPers Collaboration Scheme

First and foremost, the goal of a collaborative modeling platform is to enable collaborators to access and modify

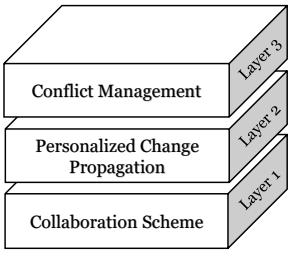


Figure 4: Layered Architecture of the CoMPers Framework

models effectively and without any inconsistencies. To satisfy this goal, we have to address the collaboration needs introduced in the motivating example (see Section 2). Therefore, our collaboration scheme takes into account personalized change propagation and conflict management as the main requirements. In particular, a scheme for collaborative modeling should be able to accommodate multiple collaborators who concurrently review and modify a model [20, 2]. Furthermore, if the collaborative platform does not take any precautions regarding conflicts, this will inevitably lead to confusion and inconsistency in shared resources. Therefore, we select the operation-based approach to propagate local changes using a collaboration server. A change operation is the description of an atomic manipulation of a document [21]. We considere Add, Delete, and Update operations applied on elements of local models. In this case, the server is responsible for checking the conflict to accept or reject the change operations that are committed by each client.

To develop a collaborative modeling platform, we consider a server that acts as a central hub to store and synchronize models. In particular, the server should control change operations submitted by clients to ensure their consistency and then propagate consistent modifications to other clients. Moreover, each client should be able to download a specific model to its local system. Clients can execute several change operations on their local copies to modify a model. The applied operations can be submitted to the server to be checked for possible conflicts. Any changes that lead to conflict situations and cannot be resolved must be returned to the relevant client to be canceled. However, the modifications approved by the server must be used to update the model, including the version stored on the server and the clients' local versions. To this end, the server can submit the modifications to the online clients or hold them for a late download by the offline clients.

The aforementioned process includes the basic actions that, nowadays, a platform should provide to support a suitable collaboration among multiple users [22]. These actions may be called differently in various implementations. For example, clone, pull, and push are used in Git, while checkout, update, and commit are used in SVN. Like any collaboration platform, we define the basic actions of the collaboration scheme as follows:

- ***Checkout***, which is the ability to download the latest version of a model from the server to the local workspace of a specific client who initiated the operation.
- ***Update***, which is the ability to retrieve the approved modifications from the server-side to the local workspace of a specific client who initiated the operation.
- ***Commit***, which is the ability to submit the change operations of a specific client to the server-side.

In the following, we explain the behavior of the basic actions for the server and the clients in our collaboration scheme using state machines. A state machine consists of states and transitions between states. The current state specifies the system at a certain time. A system starts from the initial state and moves to the other by receiving input or sending output events. We used "?" and "!" labels on the edges to denote the receiving and sending of a certain event, respectively. In the concept of collaboration, the label of each transition specifies the event and related data inside parenthesis. A transition will be executed immediately when its input event arrives, and during the execution, it produces its output event. Two state machines, one on the server side and the other on the client side, can synchronize on events sent by one and received by the other.

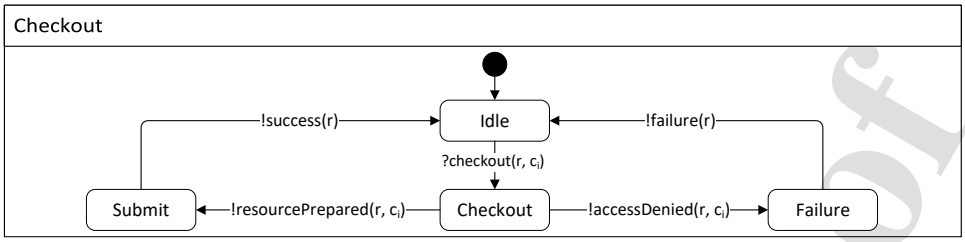
4.1.1. Server Behavior

The behavior of collaboration scheme for the server includes three state machines to handle *checkout*, *update*, and *commit* requests concurrently. The main states of each request are briefly investigated in the following.

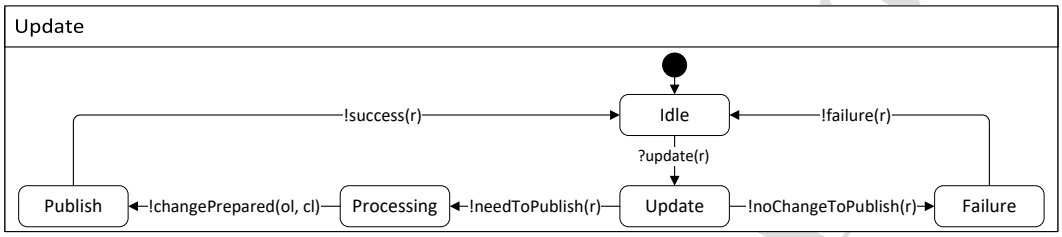
Checkout. Figure 5a shows the process of receiving checkout request consists of *Idle*, *Checkout*, *Submit*, and *Failure* states. Idle state accepts checkout requests from any clients to download an specific resource (i.e., model). When the client has no access to the resource, the server produces an accessDenied event followed by a failure event. Otherwise, the latest version of the requested resource is prepared and server moves to the Submit state that followed by a success event.

Update. Figure 5b displays the process of receiving update request, including *Idle*, *Update*, *Processing*, *Publish*, and *Failure* states. In case of receiving update request for a resource (i.e., model), the server rejects it when the server has no approved change for the resource, followed by a failure event. Otherwise, the server should process all collaborators, and publish the approved changes to the clients which their model are out-of-date.

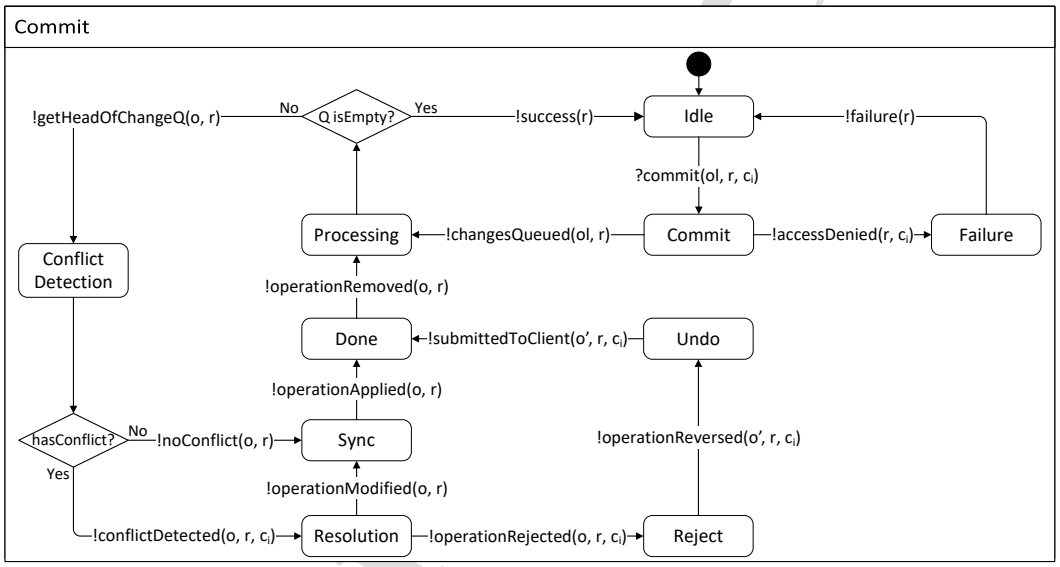
Commit. Figure 5c shows the process of receiving commit request, where Idle state receives a commit request and a list of change operations on a specific resource (i.e., model) from a client. The server produces an accessDenied event when the collaborator has no access to the resource. Otherwise, the server checks the incoming changes one by one to detect conflicts. When a



(a) Checkout Action Scheme



(b) Update Action Scheme



(c) Commit Action Scheme

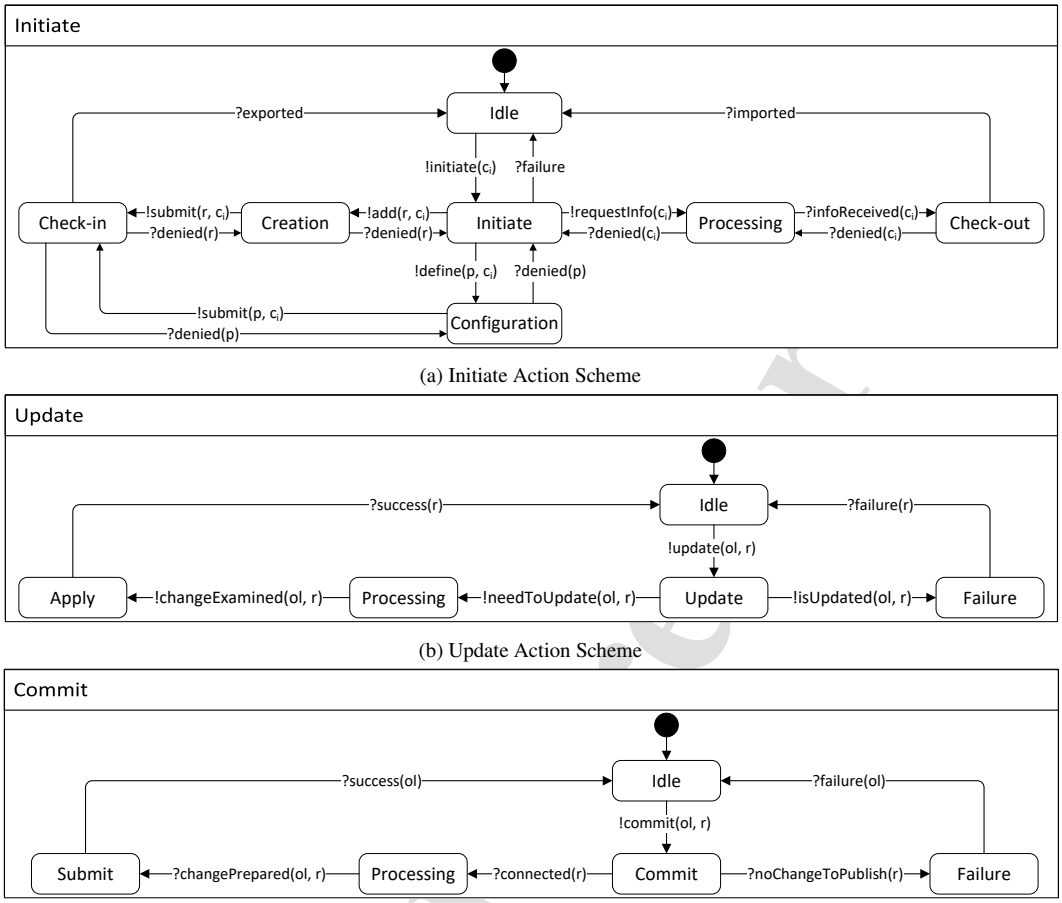
Figure 5: State-machine of Actions in the Collaboration Server – r is the resource, cl is the list of subscribed clients, ci is the requester client, ol is the list of change operations, o is the operation on head of queue, and o' is the reverse of o. Marks ? and ! mean receive and send events, respectively.

change is conflicting, the server goes to the Resolution state to fix the conflict. If the fix was successful or the change operation did not lead to a conflict, the server goes into Sync state and saves the change as applied operation. Otherwise, the server produces an operationRejected event and moves forward to undo the operation on the relevant client. Then server moves forward to the Done state and continues until all the change operations are processed and send the success event to the owner of the commit.

4.1.2. Client Behavior

Each client to cooperates with the server must send and receive events that trigger *checkout*, *update*, and *commit* requests. The basic action that clients have to support are *initiate*, *update*, and *commit*. In the following, we describe the behavior of collaboration scheme for the client consists of three state machines to handle *initiate*, *update*, and *commit* requests.

Initiate. Figure 6a shows the process of sending an initiate request, where a client can move to Idle state based



(a) Initiate Action Scheme

(b) Update Action Scheme

(c) Commit Action Scheme

Figure 6: State-machine of Actions in the Collaboration Client – r is the resource, ci is the requester client, p is the propagation strategy, and ol is the list of change operations. Marks ? and ! mean receive and send events, respectively.

on the server's response. The initiate requests consists of three events to handle initialization for a client. Clients can receive the list of resources (i.e., models) to checkout them by sending a *requestInfo* event. They can customize how to propagate (i.e., send or receive) change operations for a specific resource by sending a *define* event. Furthermore, a client may create a new resource and submit it to the server by sending an *add* event.

Update. Figure 6b presents the process of receiving an update on a specific resource (i.e., model) from the server. Each update event produced by the server can be received, but only the required change operations are applied to update the resource.

Commit. Figure 6c shows the commit process for a client. All clients can commit their changes to the server by sending commit event on the dedicated resources (i.e., models).

4.2. CoMPers Personalized Change Propagation

In previous work [7], we introduced the PCP approach to provide infrastructure for personalized change propagation in collaborative modeling environments. In this section, we focus on CoMPers personalized change propagation layer. We first present a conceptual overview of the main concepts of the PCP approach. Then, we discuss how the PCP approach has been conceived to provide a well-formed collaborative platform based on the proposed collaboration scheme.

4.2.1. PCP Conceptual Overview

Change propagation, either synchronous (instantaneous) or asynchronous (delayed), is an essential part of collaborative modeling and has a high impact on stakeholders' cooperation. A collaborative modeling platform should provide the following prerequisites to support a well-formed change propagation features [7]:

- *Client registration* and *unique identity* are required to authenticate collaborators

- Clients should receive a *list of models* when connect to the collaboration server
- Each *model* should be identified by a *unique identity*
- The collaboration server must *create global copy* for each model
- Each *change operation* has a *publisher* and modifies a *model* that should be specified for the *check-in*
- The collaboration server must *create check-out channels* and *notify clients* for new updates
- Each client can *subscribe* to only one *check-out* method for each model

Personalized Change Propagation (PCP) refers to the solution in which each user is able to customize the fundamental features of change propagation according to his/her needs. It consists of a *collaborative server* and a number of clients, which underpin the development of different modeling artifacts. In order to optimize the change propagation, the collaborative server works as a broker to store and transmit models and their changes by means of Application Program Interfaces (APIs). Each client who joins the collaboration has a user identity and particular access to modify model views. Clients may adapt *Publish-Subscribe* and *Request-Response* pattern for change propagation. Following the *Publish-Subscribe* pattern, the collaboration server does not broadcast every change operations to other clients. The server enables the definition of particular channels to personalize sending and receiving modifications for each client, given the change propagation features. To this end, the *publisher* client should select a check-in method for sending its modifications to the collaboration server. The defined check-in methods are as follows:

- ***Online*** check-in that sends each change operation to the server immediately,
- ***Offline*** check-in that sends a group of change operations to the server at a later time.

In the collaboration server, the change propagation is considered as updating and synchronizing models. When the server receives new change operations for a model, it first updates the global instance of the model. Then it forwards all change operations to the check-out channels that are specified for the model. Clients should subscribe to one check-out channel for receiving changes that are publishing in a model. The PCP approach proposed three following check-out channels for each model.

- ***On-demand*** in which subscribers immediately receive any change operation of the resource when they are connected to the collaboration server. This method is similar to the *any modification* approach of the push method.
- ***On-schedule*** in which subscribers receive existing change operations of the resouce in a *Specific Time* (e.g., at 2:00 A.M of every day), in a *Period* (e.g., every 10 minutes), or based on a *Numerical Threshold* (e.g. for every 10 changes). We can map the *On-schedule* check-out method to the *periodic* and *change threshold* approaches of the push method.
- ***On-close*** in which subscribers receive all change operations that are submitted to the server by users that are working on the resource when all users close the resource or commit the changes. This method is a severe case that propagates changes on specific conditions, which is concerned with *Event-Driven* approach of the push method.

Moreover, PCP allows clients to receive a resource from the collaboration server following *Request-Response* pattern. Using this method that called *On-request*, clients can receive the latest version of a resource and its *position* from the server when requesting it. The position is a number, which specifies the last change operation that the collaboration server applied to the resource.

4.2.2. Realization of Collaboration Scheme

Personalized change propagation can be used by the CoMPers collaboration scheme (Section 4.1) to allow collaborators customize their platform based on their needs and habits. To realize this idea, it is crucial to support both *Publish-Subscribe* and *Request-Response* communications in order to perform PCP in a collaborative platform. In this section, we detail how the PCP approach has been extended to support the prerequisites introduced in Section 4.2.1 for a suitable collaborative modeling.

To address the aforementioned requirements, we present a mapping between the basic actions of collaboration scheme and PCP operations. Figure 7 displays the architecture of the proposed collaborative modeling platform with respect to the PCP approach and CoMPers collaboration scheme. In this architecture, the collaboration server includes two components: *incoming manager* and *outgoing manager*. The incoming manager is in the charge of receiving clients' requests and data (e.g., change operations or client's preferences), which reflects the *Commit* action of server in collaboration scheme. Moreover, the outgoing manager is responsible for sending the accepted changes to the clients based on the *Response* and *Publish* operations. In this context, the response operation provides the *Checkout* action for the server, and different methods of publish operation permits the server to support of *Update* action in collaboration scheme.

The *change manager* in each client is also in the charge of communication with the server. In this case, the *Receive* and *Online/Offline Publish* respectively provide *Update* and *Commit* actions of clients in collaboration scheme. Moreover, possible requests of *Initiate* action are reflected by several operations. The *requestInfo* is supported by *Registration* and *Request* operations. The *Add* operation allows the *Creation* of new model, and the *Subscribe* operation permits client to define the configuration for change propagation.

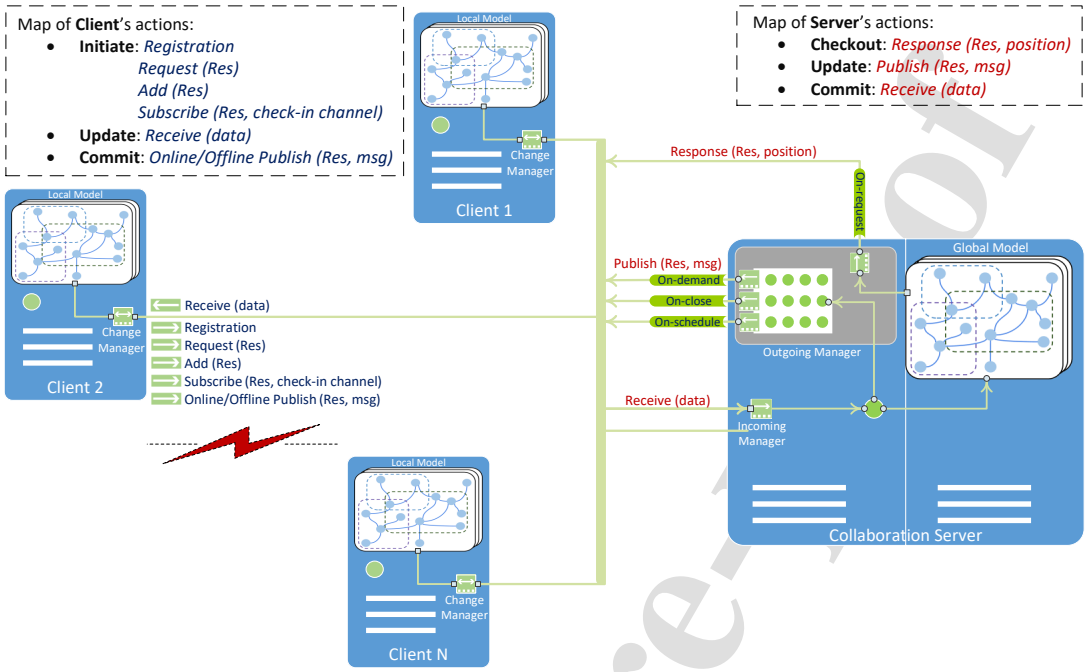


Figure 7: Mapping PCP Features to Realize Basic Actions of Collaboration Scheme

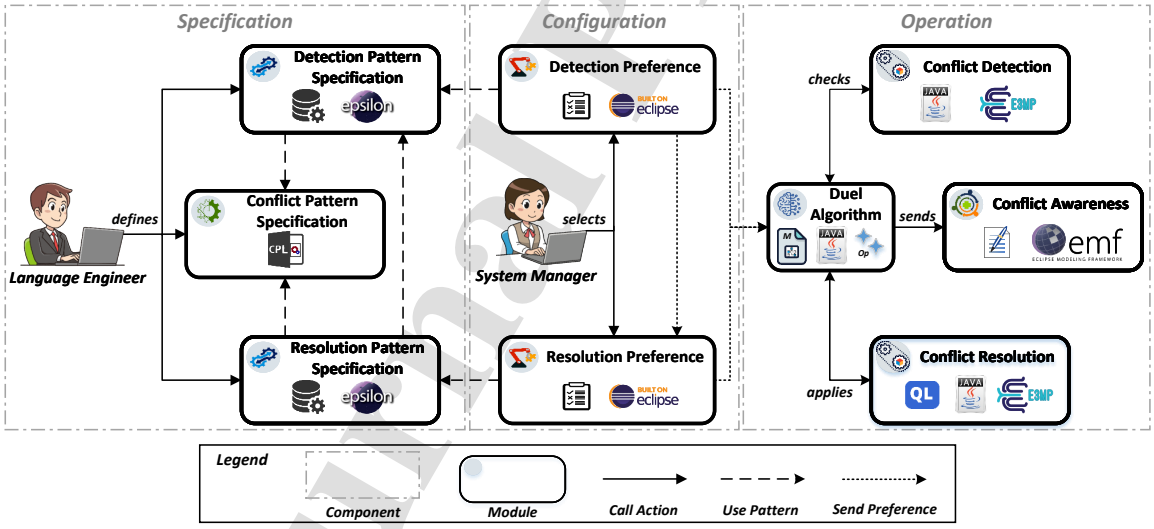


Figure 8: The Architecture of CoMPers Conflict Management Layer

4.3. CoMPers Conflict Management

In this section, we introduce the CoMPers conflict management layer in order to detail its architecture and specify techniques used to support and customize conflicts management in personalized change propagation. We have designed the CoMPers conflict management specifically based on the CoMPers collaborative modeling platform and have extended its components as a sub-system of the *collaboration server*, which is presented in Section 4.2. The architecture of CoMPers conflict management is shown in Fig. 8. The CoMPers framework has been designed around three main components: *Specification*, *Configuration*, and *Operation*.

The *Specification* component provides three specification modules for language engineers to define a list of patterns and rules in order to manage a conflictual situation

in the merging process. In this component, a language engineer can describe a conflict situation using the CPL [18] template in *Conflict Pattern Specification* module. Following the specified conflict patterns, language engineers can define several conflict detection rules using the EPL[1] [23] or EVL[2] [24] rules. They can also design some resolution rules using the EOL[3] [25] or EVL [24] rules, regarding to the defined specification patterns and detection rules.

The *Configuration* component is responsible for arranging the conflict management procedure. In this case, the *system manager* can specify the preferences for conflict detection and resolution to customize how the *duel algorithm* performs conflict management process. The *detection preferences* indicate the conflict detection technique and the list of conflict detection rules that must be checked to approve an incoming change operation. Depending on the chosen detection technique by the system manager, CoMPers suggests resolution techniques that can be used to resolve conflict situations. The *resolution preferences* indicate the type of resolution technique or resolution rules the system manager wants to apply in conflicts.

The *Operation* component is responsible for validating incoming change operations in the collaboration server. In particular, this component acts as an interface between the *incoming manager* and the *outgoing manager* in the collaboration server. Each change operation that is received in the incoming manager should be submitted to the duel algorithm to be checked based on the detection preferences. In this case, all the change operations can be moved to the outgoing manager for propagation, but only the change that leads to a conflict and does not comply with the resolution preferences will be rejected. The *Operation* component consists of four modules: *Duel Algorithm*, *Conflict Detection*, *Conflict Resolution*, and *Conflict Awareness*.

As stated in Section 3.2, *conflict specification*, *conflict detection*, and *conflict resolution* are three mandatory aspects of a valid conflict management approach. To achieve this goal, the conflict management layer of the CoMPers framework, in addition to preparing mandatory aspects, has provided proper support for the optional *conflict awareness* aspect. In the following, we will go through the techniques that we used in the specified modules to realize conflict management in the CoMPers framework.

4.3.1. Conflict Pattern Specification Module

To address the conflict specification, we integrate the CPL presented in our previous work [18] within the CoMPers framework. CPL is a formal language used to describe conflict situations using conflict patterns. Using CPL, language engineers can specify and share the conditions of conflict occurrences, which can aid in better understanding and defining conflict detection and resolution patterns.

[1]Epsilon Pattern language (EPL)
[2]Epsilon Validation Language (EVL)
[3]Epsilon Operation Language (EOL)

4.3.2. Detection Pattern Specification Module

For operation-based frameworks, such as CoMPers, a conflict detection pattern must accurately describe the state of model elements and properties of the change operation that result in conflicts. As an example, a pattern could be defined to detect the *Inheritance Cycle* [18] conflict depicted in Fig 9. In this conflict, *adding* a new inheritance relationship to a newly updated UML class diagram leads to the formation of an inheritance cycle. To illustrate, Listing 1 displays the conflict detection pattern for the inheritance cycle written in EVL. E3MP utilizes the Epsilon engine to search for the condition stated in the *check* block of Listing 1.

```
context Model!Class {
constraint cyclicInheritance {
   guard : self.generalization.isDefined()
   check {
      if(self.generalization.general→closure(it|
         it.generalization.all)→exists(c|c.owner.name == self.name)){
            return true ;
      }
    }
}
```

Listing 1: EVL Rule to Detect Cyclic Inheritance Conflict

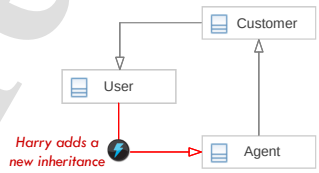


Figure 9: Inheritance Cycle: An Example of Conflict Detected by Pattern Matching

4.3.3. Resolution Pattern Specification Module

Each resolution pattern describes the modification that can be applied to update the model or edit the change operation to fix a specific conflict. Language engineers can define one or more resolution patterns for every conflict detection pattern based on the EVL or EOL languages. Using the EVL and EOL rules, CoMPers can directly update a model. To illustrate, Listing 2 shows an example of EOL conflict resolution rules for inheritance, where the inheritance relationship is reversed to eliminate the cycle.

```
var emfTool= new Native("org.eclipse.epsilon.emc.emf.tools.EmfTool");
var ecoreUtil= emfTool.ecoreUtil;

for (Operation in Source!Operation.all) {
   if(op.kind == "Generalization"){
      var cloneOp = ecoreUtil.copy(op);
      op.source = cloneOp.target;
      op.target = cloneOp.source;
   }
}
```

Listing 2: Example of EOL Script for Resolving Cyclic Inheritance Conflict

4.3.4. Detection Preference Module

To configure the conflict detection process, the system manager must select the detection preferences, including

the techniques for operating the conflict detection as well as the detection patterns that required to be checked in the conflict detectin process. In the following section, we will briefly describe two main ***detection preferences*** that conflict manager must decide about it in the CoMPers framework.

Change Overlapping. In collaborative modeling, conflicts may arise form concurrent manipulation of models due to the dependencies between atomic change operations (i.e., add, delete, and update). The origin of these conflicts can be categorized based on the equivalent or contradicting changes, such as *add/add*, *update/update*, and *delete/update*, which are applied to the same model element [16]. For instance, *delete/update* conflicts in the UML class diagram occur when one modeler deletes a class that is concurrently updated by another. Moreover, *Add/delete* and *delete/update* are instances of atomic change combinations that can lead to conflicts if applied together on different elements [17]. As shown in Fig 10, *add/delete* change overlapping in the UML class diagram results in a *dangling reference* [18]. This happens when one modeler adds a new association that uses a class concurrently deleted by another. In this context, we developed a conflict detection module that checks for possible conflicting combinations to identify conflicts among the incoming change operations applied in parallel. For example, in the case of the *dangling reference* illustrated in Fig 10, the change overlapping approach detects the conflict between the association added by Harry and the delete operation performed by Sally, when examining the queue of received change operations in the collaboration server.

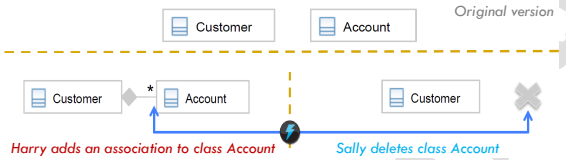


Figure 10: Dangling Reference: An Example of Conflict Detected by Change Overlapping

Pattern Matching. Due to the existence of different types of conflicts and their dependency on the modeling language, we require an extensible solution capable of addressing these conflicts. We propose to detect these conflicts through the definition of conflict patterns that describe the occurrence of such conflicts. In this case, a graph matching algorithm used to find subgraphs that match detection patterns. Considering that the number of conflict detection patterns are very diverse, at this phase we ask the system manger to select the required detection patterns to be checked.

4.3.5. Resolution Preference Module

To tune the conflict resolution process, the system manager must select the resolution preferences. If the system manager selects the *Pattern Matching* technique, *Pattern-Based Resolution* and *Abort Change* are available resolution approaches. Furthermore, if the *Change Overlapping* is chosen as a conflict detection technique, we can simply *Abort* the change in question or perform one of the available resolution techniques: *Latest Operation Wins*, *Highest Priority Wins*, or *RL-Based Resolution*. In the following, we briefly explain each technique as ***resolution preferences*** that conflict manager must decide about the used of it in the CoMPers framework.

Pattern-Based Resolution. Like the conflict detection pattern, we propose to define conflict reconciliation using the conflict resolution pattern. We have extended the E3MP toolkit to perform the conflict resolution patterns defined using the EVL or EOL.

Abort Change. A simple and lazy strategy to resolve a conflict is to abort the conflicting change operation. In this approach, the system manager chooses to discard each conflicting change operation in order to maintain a consistent model.

Latest Operation Wins. The conflict can be resolved by applying only one change and discarding the other. In this strategy, we assume that the most recent change to the model is the most significant, so we apply the change operation that the client most recently performed.

Highest Priority Wins. This strategy allows for the application of one operation among two conflicting change operations. Given a collection of client priorities, we assume the operation that a client applied by the client with the highest priority is the one to be used, while the other operation must be discarded.

RL-Based Resolution. To deal with the conflict, trial-and-error approaches are usually present. We used our previous work called CoReRL [26], which is an RL-based approach to identify the operation that should be applied to resolve the conflict. In particular, using the RL-based resolution approach, we attempts to learn which operation should be selected as the winner using a Q-Learning algorithm and former experiences.

4.3.6. Conflict Detection Module

Once we have designed a collaborative modeling platform based on change propagation, we need to utilize conflict detection techniques that focus on change operations. Therefore, we initially develop our conflict detection approach by utilizing the *change overlapping* technique, which can effectively identify most conflicts. However, in order to enable CoMPers to identify a wide range of conflicts pertaining to the meaning of the modeling language, we propose a conflict detection approach based on the *pattern matching* technique. This technique allows for the detection of semantic conflicts in various modeling languages through user-defined rules. It is important to note that additional conflict management techniques can be incorporated as new

components within CoMPers, taking advantage of its modular structure.

To have change overlapping technique in the CoMPers framework, we developed a conflict detection module that checks for possible conflicting combinations to identify conflicts among the incoming change operations applied in parallel. For example, in the case of the *dangling reference* illustrated in Fig 10, the change overlapping approach detects the conflict between the association added by Harry and the delete operation performed by Sally, when examining the queue of received change operations in the collaboration server.

To have pattern matching technique in the CoMPers framework, we have developed the consistency checking module within the E3MP toolkit [27] to identify the conflict situations defined using the EVL or EPL rules. The consistency checking module of E3MP examines the specified conflict detection patterns for each operation at the collaboration server. For this purpose, E3MP initially creates a tentative merged model by applying the change operation and subsequently searches the merged model to identify elements that satisfy the conflict conditions defined by the conflict pattern.

4.3.7. Conflict Resolution Module

According to the selected conflict detection approach, different techniques can be used to resolve conflicts. Figure 11 depicts the workflow of the presented technique to detect and resolve conflicts using conflict patterns. In this case, the detected conflict, after applying the incoming change operation to the tentative merged model, is resolved automatically by applying candidate resolution patterns or fixed semi-automatically through user.

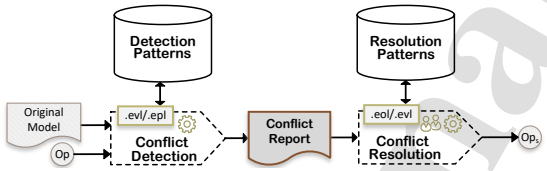


Figure 11: Workflow of Conflict Resolution for Pattern-Based Conflict Detection

Figure 12 shows the workflow of the presented approach to resolving conflicts detected by the change overlapping technique. In this case, two overlapping change operations are given as input to the conflict resolution component, and this component identifies the winning change based on the selected resolution technique.

Figure 12: Workflow of Conflict Resolution for Change Overlapping Conflict Detection

Moreover, to apply RL-based resolution approach, we follows the workflow shown in Fig. 13. This approach involves identifying the winning change operation among two conflicting operations using RL and resolution experiences. According to this workflow, two conflictual change operations are provided as input, and the RL-Based resolution module of CoMPers, based on the learning experiences and quality rewards, determines the winning change operation.

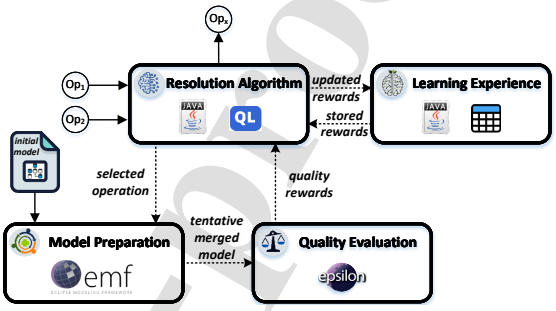


Figure 13: Workflow of Conflict Resolution Based on RL

4.3.8. Conflict Awareness Module

According to the conflict management taxonomy [8], collaborators can be aware of the occurrence of conflict by using *Notification*, *Conflict Highlighting*, and *Conflict Report*. In the CoMPers framework, we provide a textual conflict report that includes the history of change operations and conflicts. The collaboration server immediately updates this report for each client when the *duel algorithm* propagates approved change operations or encounters conflicts. The conflict log reports the operation that led to the conflict and how it was handled.

4.3.9. Duel Algorithm Module

The *duel algorithm* is the core of the operation component in the CoMPers conflict management framework. This algorithm is named *'Duel'* because every incoming change operation in the collaboration server must compete with other conflicting change operations or pass conflict detection patterns to be finalized. The overview of the duel algorithm is depicted in Fig. 14. The duel algorithm starts by receiving a change operation in the server and will continue until the changes end. Incoming changes are moved to a change queue (i.e., Q) to be checked one-by-one with a FIFO strategy. In each iteration, the algorithm takes the head of the change queue (Op_1) and checks for possible conflict based on the conflict detection strategy. As shown in Fig. 8, the conflict detection and conflict resolution strategies are chosen by the system manager in the configuration component.

In the left side of the duel algorithm in Fig. 10, where *pattern matching* is the selected conflict detection strategy, the relevant conflict detection rules are used to detect conflict by applying Op_1 to the tentative merge model. If no conflict is detected and the consistency of the change is approved, the operation can be published to the clients. Otherwise,

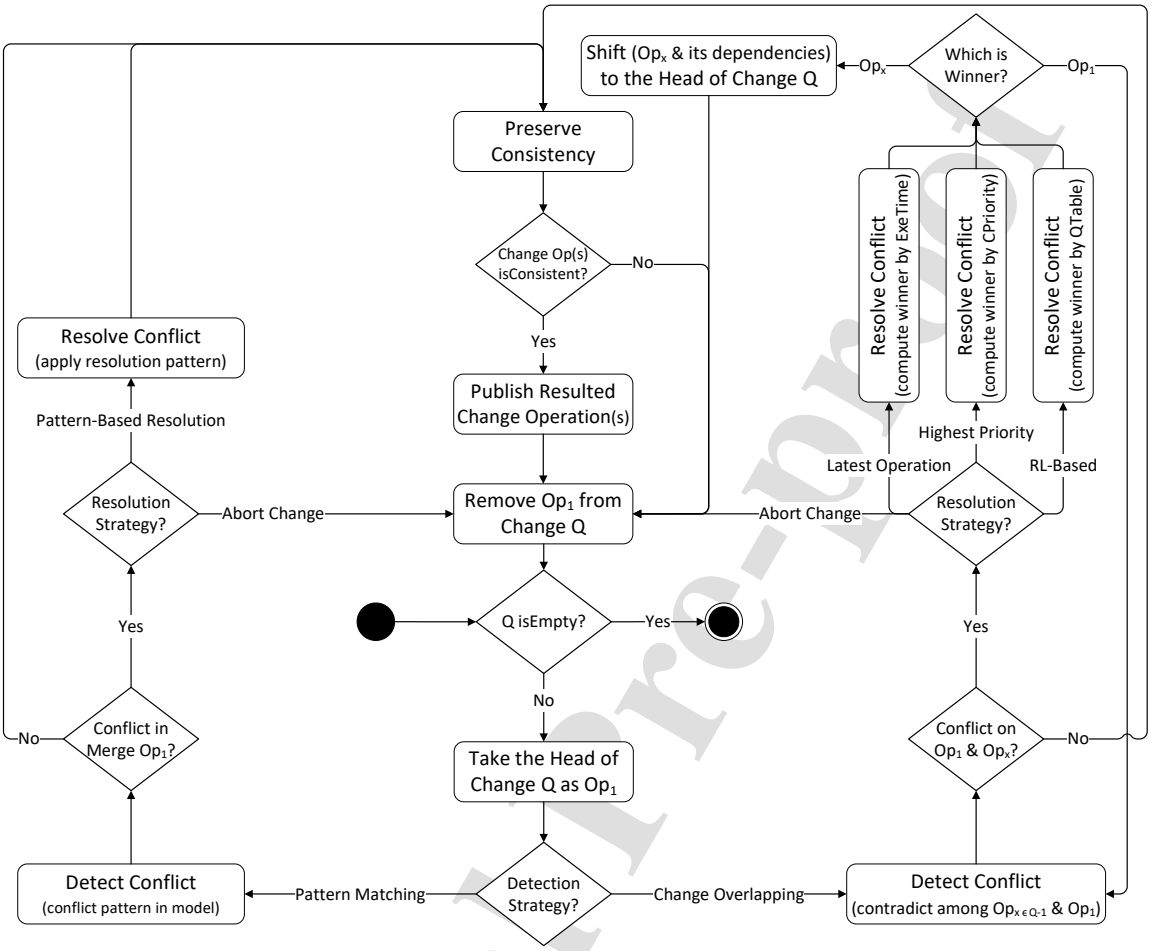


Figure 14: Overview of Duel Algorithm to Detect and Resolve Conflicts

depending on the selected resolution strategy, the change should be discarded or repaired using the relevant conflict resolution rules to produce a consistent modification that the algorithm allows to be published.

On the right side of the duel algorithm in Fig. 14, the selected conflict detection strategy is *change overlapping*. The system iterates over all other change operations in the queue to detect operations (Op_x) that contradict Op_1. In this case, Op_1 can be published if it does not contradict all others. However, if a conflict occurs, the algorithm should abort Op_1 or decide to apply only one of the two contradicting change operations, depending on the selected resolution strategy. After applying the resolution technique for Op_1 and Op_x, if Op_1 wins the competition with Op_x, the algorithm continues to check for other contradictions until there is no conflict with Op_1 and it can be published. Otherwise, the algorithm must move Op_x and its dependencies to the front of the queue and discard Op_1.

After confirming each operation (i.e., Op_1), either through pattern matching or the change overlapping technique, the algorithm proceeds to a consistency preservation step to check general inconsistency conditions and modify the operation if possible. At this point, the change is prepared to be published and synchronized across all clients. In the last step, the algorithm removes Op_1 from the change queue to initiate the iteration for the next operation.

5. Implementation

We presented the conceptual overview of the CoMPers collaborative modeling platform in Section 4.3. We will discuss our prototype implementation of this platform, which serves as a proof of concept. The CoMPers platform consists of a centralized *Collaboration Server*, as well as several *Clients* that are connected to the Server. The CoMPers collaboration server is general and acts as a message broker for the clients. Clients can be instances of any modeling

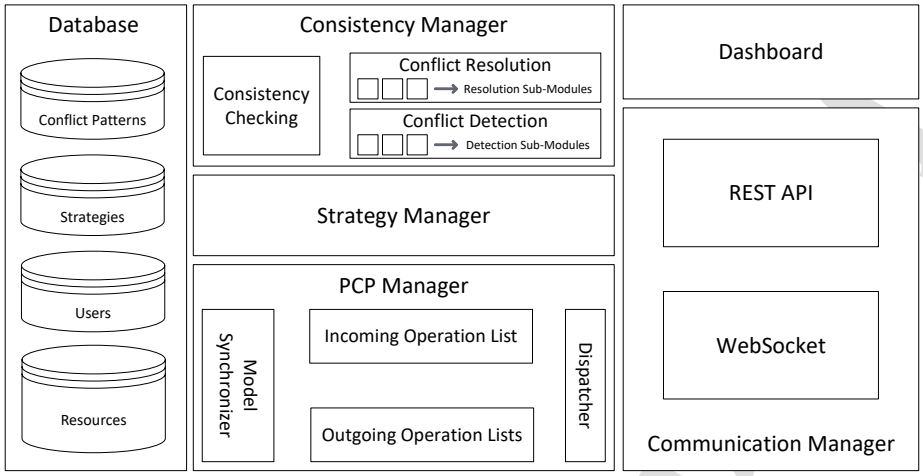


Figure 15: Architecture of CoMPers Collaboration Server

environment developed to communicate through the collaboration server. In our prototype implementation, we selected the EMF.cloud framework[4] and extended its GLSP-based Ecore editor as clients. In this section, we first discuss the CoMPers collaboration server architecture and our choices of technology in Section 5.1. Then, we briefly introduce the EMF.cloud architecture and present the extension details of the GLSP-based Ecore editor to prepare a sample CoMPers client in Section 5.2. We conclude with a discussion in Section 5.3 to emphasize that EMF.cloud is only an example choice in our implementation, and it can be replaced by any environment that is able to record changes.

5.1. The CoMPers Collaboration Server

The collaboration server plays a central role in collaborative modeling in the CoMPers platform. It is responsible for processing change operations. Specifically, the server receives, manages, and sends change operations among clients. Figure 15 illustrates the simplified core function of the CoMPers collaboration server, which consists of five sub-systems that support conflict management and change propagation presented in this work. For implementation, we chose the Java programming language. We developed the CoMPers collaboration server based on Javalin[5], which is a lightweight web framework for Kotlin and Java. Javalin runs on top of Jetty[6], which is one of the most used and stable web servers on the JVM. We opted Javalin for two reasons. Firstly, it easily supports the creation of WebSockets and REST APIs in Java. Secondly, it integrates well with our client/server infrastructure, as this is designed to be simple and blocking, which facilitates reasoning and programming. In the following, we introduce the modules of the CoMPers architecture as shown in Fig. 15.

[4] https://www.eclipse.org/emfcloud
[5] https://javalin.io
[6] https://www.eclipse.org/jetty

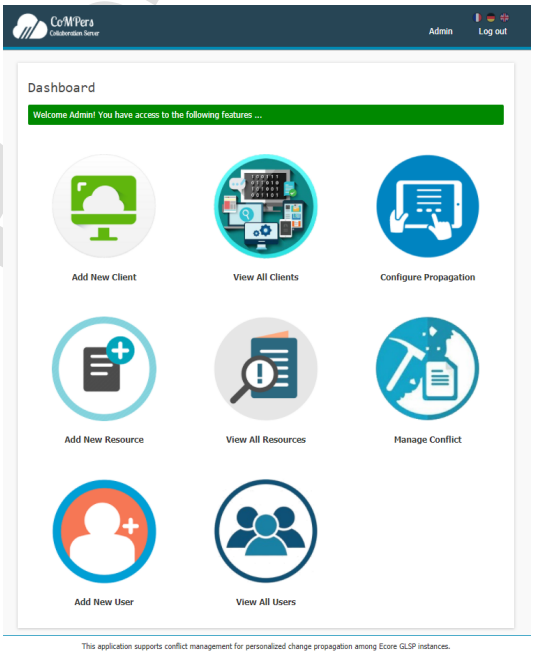


Figure 16: Dashboard of CoMPers Collaboration Server

In the collaboration server, we employ the *Communication Manager* sub-system to handle the request and response between the server and clients based on *WebSocket*. At its core, the JSON arrays of operations and clients' information are the basic data structures for collaborative modeling. The *PCP Manager* module uses Gson[7] to encode or decode data in the collaboration. Each received change operation should be checked by the *Consistency Manager*. The Consistency

[7] https://github.com/google/gson

Manager is an extensible sub-system that contains all consistency rules for EMF-based modeling, as well as engines for detecting and resolving conflicts, which are encapsulated in a purely modular core. This core is then surrounded by a shell of imperative code to manage conflict handling strategy and a web-based *Dashboard*. The dashboard can manipulate the conflict management and change propagation strategies. Figure 16 shows the profile page in the dashboard, which is designed with Velocity[8] templates to manage clients, resources, and conflict handling strategy in coordination with the *Strategy Manager* sub-system using the RESTful APIs. The implementation is available on GitHub[9] under the Apache 2.0 license. For more information, refer to our GitHub repository.

5.2. The CoMPers Client: An Extension of EMF.cloud

In our client/server architecture, collaborators apply change operations to the model via an application that connects directly to the collaboration server. In the current prototype implementation of this platform, we have chosen the EMF.cloud framework, which provides a modeling tool implemented as a web application accessible through any modern web browser. In particular, we have extended the Ecore GLSP editor, a web-based editor for drawing Ecore diagrams within the EMF.cloud framework. There are three main reasons for selecting this modeling environment. Firstly, it allows for provision of a portable and universal client that can be used across all platforms without any setup required. Secondly, it is an open-source modeling environment that supports Ecore, which is the core (meta-)model at the heart of the Eclipse Modeling Framework (EMF) [28]. Thirdly, collaborators can view and edit Ecore models in a graphical representation. Furthermore, the EMF.cloud framework is becoming increasingly attractive as a modeling environment, offering developers the ability to reuse existing tool components based on EMF in the cloud. In the following, we briefly introduce the general architecture EMF.cloud. Finally, we focus on the Ecore GLSP editor and extend it based on the personalized change propagation strategy in order to provide the CoMPers client.

5.2.1. EMF.cloud Modeling Framework

EMF.cloud is an umbrella project founded by EclipseSource[10] to provide components and technologies that make it easy to build modeling tools on the web and based on EMF. Figure 17 shows the basic architecture of the EMF.cloud modeling framework. It consists of a local *Model Server* and a *Theia-based Client*. While the Theia-based client is responsible for displaying a model to the user, the model server manages the runtime state of loaded models and allows applying changes using a command pattern. Theia is a flexible platform for building custom clients for any purpose. As shown in Fig. 17, the Theia-based client includes a web-based editor (e.g., Tree Editor) that runs based on the Theia frontend and a model connector, which is used to publish the changes between the editor and model server using Theia backend. Blue boxes are fixed components which can be used without any changes for new modeling languages and the golden are components need to be updated for each modeling language. In this case, the EMF.cloud client can be extended for different modeling languages. However, the current version of the EMF.cloud framework does not support multi-user modeling, and only one client can connect to the model server for loading and editing a model. In Section 5.2.3, we address this limitation using the collaboration server.

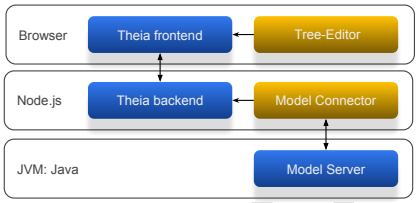


Figure 17: Architecture of the EMF.cloud Framework [29]

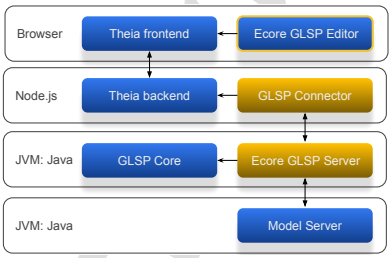


Figure 18: Architecture of the Ecore GLSP Editor in EMF.cloud [29]

5.2.2. Ecore GLSP Editor in EMF.cloud

Ecore GLSP integrates GLSP with Theia to provide a web-based editor for Ecore models. The Graphical Language Server Platform (GLSP)[11] is an extensible open-source framework for building custom diagram editors based on web technologies. As shown in Fig. 18, the Ecore GLSP editor extends the basic architecture of the EMF.cloud framework with two components: Ecore GLSP Server and Ecore GLSP Editor [30]. First, an Ecore GLSP language server is written in Java to support Ecore notations in the model server. Second, an Ecore GLSP editor is a client extension to present the Ecore diagrams using Sprotty[12]. In this case, the diagram layout will be persisted in an *.enotation* file next to the *.ecore* file.

[8] https://velocity.apache.org
[9] https://github.com/MSharbaf/CoMPers
[10] https://eclipsesource.com
[11] https://www.eclipse.org/glsp
[12] https://projects.eclipse.org/projects/ecd.sprotty

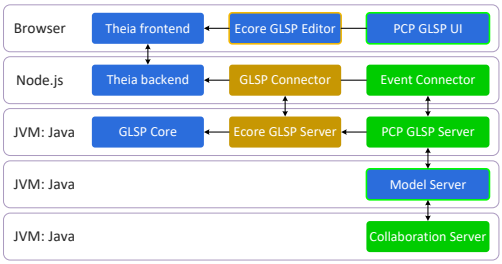


Figure 19: Architecture of Multi-user Modeling with Ecore GLSP Editor in EMF.cloud

5.2.3. Personalized Collaborative Modeling in Ecore GLSP Editor

In Section 4.2, we introduced the Personalized Change Propagation (PCP) approach to encourage communication between multiple users. To implement this approach and support multi-user modeling in the Ecore GLSP editor, we should provide proper support for the required actions in a CoMPers client, such as *Receive* and *Publish* change operations. To achieve this goal, we extended the Ecore GLSP editor based on the architecture that is presented in Fig. 19. We added a central collaboration server, which is used to coordinate the change propagation among all connected users following an advanced RESTful API protocol that uses the JSON serialization to transfer model data. Moreover, we extended the EMF.cloud model server with a change manager module to connect Ecore GLSP instances to the collaboration server. We added the *PCP GLSP Server* on top of the *Ecore GLSP Server* to encode change operations applied by a client for sending to the collaboration server, or decode and perform change operations received from the collaboration server. The light-green components in Fig. 19 are those that need to be updated for each new modeling environment. Finally, we added a Theia extension to the Ecore GLSP Editor to support the essential events (e.g., change commit and close) for PCP collaboration. Figure 20 depicts a snapshot of the Ecore GLSP editor modified for PCP. It shows the change history at the bottom of the page and the offline check-in command in the displayed menu.

5.3. Discussion

We presented the implementation of the CoMPers client by extending the Ecore GLSP editor for the EMF.cloud framework in Section 5.2. However, we emphasize that this implementation is only a proof of concept for possible clients in the CoMPers collaborative modeling platform. We believe that following the data structure required for change propagation in the CoMPers platform, each extensible modeling environment that is able to record change operations can become a client of the CoMPers platform. To achieve this goal, we should develop the change manager module in a way that can receive the change operations applied in the modeling environment and perform changes coming from the collaboration server. Moreover, this module should be able to provide the necessary information, such as resource and element identities, for publishing changes to the server.

6. Evaluation

In this section, we present the evaluation of the contributions provided in this paper. First, we demonstrate the applicability of CoMPers in supporting personalized change propagation. Then, we assess the ability of the CoMPers conflict management framework to handle conflicts. Finally, we conduct a user study to evaluate the usefulness and the ease of use of CoMPers in collaborative modeling. The evaluation results, benchmarks, and the participants form are available at our GitHub repository[13].

6.1. Applicability Demonstration

To evaluate the applicability of the proposed collaboration scheme for personalized change propagation, we conducted initial measurements using the Wind Turbine case study [10] based on our proof-of-concept implementation. In particular, we built the CoMPers collaboration server and developed a powerful extension of the EMF.cloud modeling environment to support PCP operations based on the collaboration scheme. Then, we conducted another experiment to assess the execution of proposed operations that support the basic actions of the collaboration scheme, using a range of examples. We also investigated the capability of conflict management module using 15 test cases of the benchmark for the evaluation of a model versioning system [31], which contains different examples of syntax and semantics conflicts for possible combinations [27] that may result in a conflict. The results indicate the expected outputs and true performance of the PCP operations in creating and initializing models, and allowing collaborators to configure the propagation strategies for each model. In addition, we obtained the accuracy of 100% for 15 aforementioned test cases, including 11 cases result in conflict and 4 cases were conflict free. Moreover, CoMPers can reliably propagate change operations on models with fewer than 1000 elements. The results show that checking the incoming operations and handling the conflicts in the server for collaboration with five concurrent participants requires less than one second on average.

6.2. Benchmark Evaluation of Conflict Handling

Handling the conflicts in the CoMPers conflict management framework consists of techniques for conflict detection and conflict resolution. In section 4.3.6, we provided conflict detection using *pattern matching* and *change overlapping* mechanisms and performed relevant techniques to repair the detected conflicts. With this benchmark, we aim to measure the accuracy and F-measure of our conflict detection techniques, as well as the ability of CoMPers to handle conflicts and produce a consistent model. To achieve this, we used the benchmark presented in the AMOR project [11] to evaluate the *correctness* and *completeness* of conflict detection

[13] https://github.com/MSharbaf/CoMPers/tree/main/EvaluationDetails

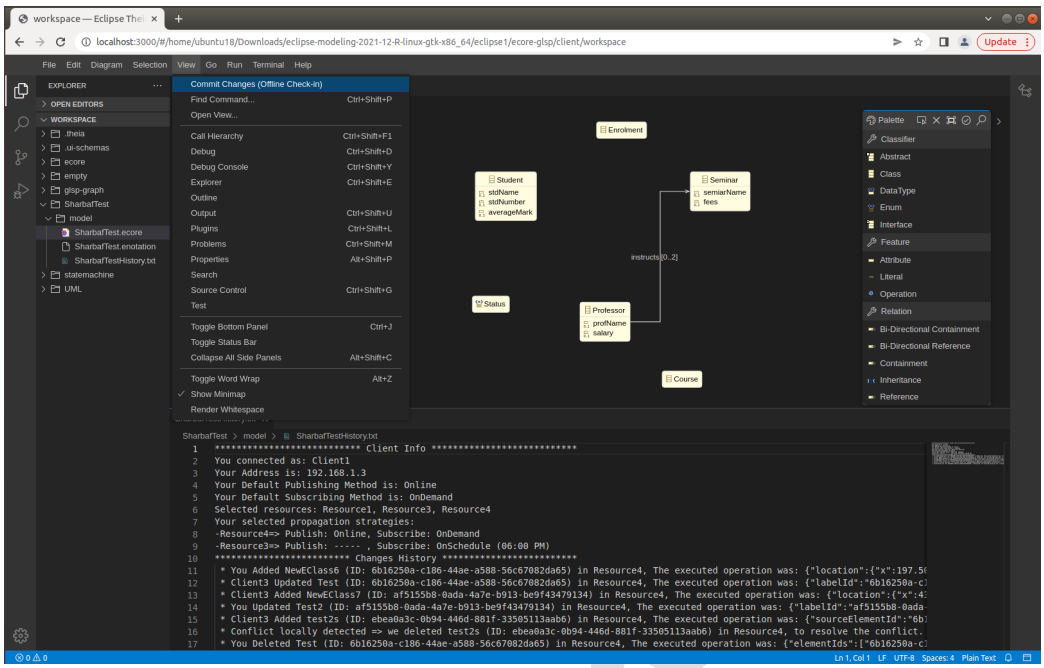


Figure 20: Ecore GLSP Editor in Multi-User EMF.cloud Environment

components. The benchmark includes 23 test scenarios for assessing the accuracy of detecting conflict in the operation-based evolution of an Ecore metamodel. Test scenarios are introduced based on the collaborative conflict lexicon [32] and cover various types of conflict among atomic change operations such as concurrent updates and contradicting delete/update, update/update, move/move, and delete/move. This benchmark includes cases in which no conflict is expected.

Measures. To measure the accuracy of conflict detection in our approach, we compute the precision and recall. The precision indicates the probability that a detected conflict is a true conflict, while recall indicates the probability that a true conflict is detected. The precision defined as the fraction of correctly detected conflicts among the set of all detected conflicts. Similarly, recall is defined as the fraction of correctly detected conflicts scenario among all conflicts that happened. Finally, we compute the accuracy and F-measure using the number of conflicts detected (tp), the number of correctly handled conflict-free scenarios (tn), the number of incorrectly raised conflicts (fp), and the number of missed conflicts (fn) as follows:

$$Percision = \frac{tp}{(tp + fp)}$$

$$Recall = \frac{tp}{(tp + fn)}$$

$$accuracy = \frac{(tp + tn)}{(tp + tn + fp + fn)}$$

$$F - measure = \frac{2 * (Precision * Recall)}{(Precision + Recall)}$$

From the perspective of model consistency, the CoMPers framework may prevent or resolve conflicts during the integration process based on the selected conflict detection strategy. Therefore, we define the accuracy of conflict handling in CoMPers as the percentage of test scenarios that result in a consistent and error-free model.

Results. Table 1 summarizes the results obtained from applying the benchmark to the CoMPers conflict detection techniques. In the case of Pattern Matching, the results are not optimal but still satisfactory. Using the relevant conflict detection rules (i.e., *dangling reference*, *contradiction in hierarchy*, *contradiction in reference*, *inheritance cycle*, *inheritance of methods*, and *equivalent associations*), the pattern matching technique could not detect 2 out of 12 expected conflicts in the 23 test scenarios of the benchmark. More specifically, conflicts for two concurrent updates of the same model element were not correctly detected by this technique. This results in a recall of 0.83 and both accuracy and F-measure of approximately 91% for conflict detection using the pattern matching technique. However, this technique surprisingly produced consistent models in all 23 test scenarios of the benchmark and achieved both accuracy and F-measure of 100% for conflict handling. In contrast, for Change Overlapping, the results are optimal. This technique has detected all expected conflicts and did not report any unexpected conflicts. Moreover, the 23 test scenarios of the benchmark led to a consistent model. Therefore, the change

Table 1
Results of the conflict detection benchmark

	#tp	#tn	#fp	#fn	#Precision	#Recall	#Accuracy	#F-measure
Pattern Matching	10	11	0	2	1.00	0.83	0.91	0.91
Change Overlapping	12	11	0	0	1.00	1.00	1.00	1.00

overlapping technique achieved an accuracy of 100% for both conflict detection and conflict handling measures. The results indicate that the change overlapping technique is complete and correct. While the pattern matching technique does not provide complete conflict handling despite producing correct models. This can lead to a frustrating user experience, as users may be aware of conflicts that don't need to be resolved.

6.3. User Experience Survey

We introduced the CoMPers framework, which aims to enable engineers who are working on the same model to customize their collaboration based on their needs and priorities. We implemented a prototype tool of the proposed approach based on the Emf.cloud modeling environment, in which users are supposed to run the modeling editor in a web browser. Then, as a general procedure of working with CoMPers, users must be connected to the collaboration server and open the target project. Furthermore, users need to create an instance model or move it from the collaboration server into the folder named *model* in the directory of the project. After opening the model, users can see the diagram of the model. In this step, users can personalize the collaboration on this model by selecting his/her strategy for getting updates and sending edits. By editing the model, the change operations are sent based on the user's propagation strategy and are reviewed according to the conflict management strategy determined by system manager. In this experiment, the system manager has chosen Change Overlapping to detect conflicts. He also set to facilitate conflict resolution based on priority. Each change that passes the conflict management process will be sent to other users according to the strategy chosen by them. Changes that fail are returned with a notification and undone in the view of their owner.

The purpose of this user survey is to assess the *usefulness* and *ease of use* of CoMPers in real-world collaborative modeling. According to Keil et al. [33], the term *usefulness* is defined as "a function of task/tool fit", while *ease of use* is considered a "task-independent construct reflecting intrinsic properties of the user interface." In the context of collaborative modeling, we define the support of teamwork and customization as two functionalities that impact the usefulness, and we describe the ease of use as the satisfaction of collaborators in using the CoMPers editor. This section evaluate the proposed approach by answering the following research questions:

- ***RQ1.*** What are the users' views on the ease of use of the proposed collaborative modeling approach?
- ***RQ2.*** What are the users' views on the usefulness of the proposed collaborative modeling approach?

6.3.1. *Participants*

We selected 10 participants from the Model-Driven Software Engineering Research Group[14] of University of Isfahan. The selected participants are graduate students (i.e., four master's student and six Ph.D. students) majoring in software engineering. We asked them to fill out the participant information forms to know their level of background knowledge. The participants meet the following requirements:

1. At least 3 years of software modeling experience.
2. Have passed the Model-Driven Software Engineering course[15].
3. Have experience in collaborative MDE and using the model management tool.

6.3.2. *Modeling Tasks*

To evaluate the usefulness and ease of use of CoMPers, we asked participants to customize their collaboration environment and concurrently modify the Ecore diagram of a university management system with two following tasks:

- ***Task1: Configuration.*** Each user should first open the modeling editor, connect to the collaboration server, and open the target project. Then, they should choose how to receive updates and send edits according to their needs. Users can work based on the default strategy set by the system manager.
- ***Task2: Modeling.*** Use the Ecore diagram to model the following information in a university management system: (1) system users such as administrators, teachers, and students. (2) system functions available to each type of user.

To this end, we divided the participants into three groups. Group 1 contains three participants that work on use cases for *User Authorization*. Group 2 includes three other participants that focus on use cases for *Student* and *Faculty Staff*. Group 3 contains four participants that work on use cases for *Professors*. We tried to make the modeling levels of the three groups as equal as possible. In addition, we asked participants in the same group that at least one participant select a different collaboration strategy. We also asked them to work together in the same and different periods of time for modeling.

[14] https://mdse.ui.ac.ir/

[15] Syllabus:https://www.york.ac.uk/students/studying/manage/programmes/module-catalogue/module/COM00111M/

Table 2
Results of user experience study

#	Question	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
1	Is it easy to create and add a new model to a collaboration?	0%	0%	20%	80%	0%
2	Is it easy to add a new element to a model?	0%	0%	0%	20%	80%
3	Is it easy to delete an element of a model?	0%	0%	0%	60%	40%
4	Is it easy to modify an element in a model?	0%	0%	0%	80%	20%
5	Is the dashboard of the CoMPers platform user-friendly?	0%	0%	0%	100%	0%
6	Is the platform appropriate for modeling with a team?	0%	0%	0%	60%	40%
7	Is the platform appropriate to propagate changes to others?	0%	0%	20%	40%	40%
8	Is an appropriate approach used to inform users of changes applied by collaborators?	0%	0%	0%	80%	20%
9	Is it easy to see the history of changes and reports of conflict?	0%	0%	0%	20%	80%
10	Would you recommend your friends use this platform for collaborative modeling?	0%	0%	20%	40%	40%
11	Is it possible to customize the method of publishing changes for each model?	0%	0%	0%	20%	80%
12	Is it possible to customize the method of receiving changes for each model?	0%	0%	0%	20%	80%
13	Is it possible to appropriately personalize the change propagation in this platform?	0%	0%	0%	60%	40%
14	Is it possible to appropriately customize the conflict management approach?	0%	0%	0%	60%	40%
15	Is it possible to customize collaboration on this platform?	0%	0%	0%	20%	80%

6.3.3. Questionnaire

To compare the participants' views, we designed a questionnaire that is depicted in Table 2. After the modeling tasks are finished, we asked participants to fill out the questionnaire. Each question in this questionnaire corresponds to an answer scale ranging from 1) "Strongly Disagree" to 5) "Strongly Agree."

6.3.4. Results and Discussion

The results of the questionnaire are summarized in Table 2. We have organized the questions in the table using horizontal lines. The first five questions are designed to assess the ease of use of the CoMPers platform, and the next ten questions evaluate the usefulness of CoMPers by investigating the support of teamwork and customization. In the following, we address the aforementioned research questions.

RQ1. What are the users' views on the ease of use of the proposed collaborative modeling approach? As indicated in the first question, 80% of participants agreed that CoMPers easily supports creating a new model. Regarding the answers to questions two to four, we have observed that 100% of participants agree that CoMPers can be easily used for applying atomic change operations. In addition, 100% of participants indicated in question five that CoMPers provides a user-friendly dashboard. Based on the first five questions, our conclusion is that users have no difficulty understanding how to work on models, and the CoMPers platform is generally user-friendly. However, there are some improvements needed in the usability of the modeling support. Currently, CoMPers only supports atomic change operations on models, i.e., add, delete and update. In this regards, adding more operations, such as copying modeling elements, implementing undo/redo functionality, and incorporating refactoring operations, will enrich the functionality and usability of CoMPers.

RQ2. What are the users' views on the usefulness of the proposed collaborative modeling approach? Considering the results obtained for questions six and seven, we observe that at least 80% of participants agree that CoMPers is an appropriate platform for performing collaboration and propagating changes on models. Based on the answers to questions eight and nine, all participants agree that changes and conflicts are effectively informed on the CoMPers platform. Additionally, question ten shows that collaborative modeling with CoMPers has been attractive for 80% of participants, to the point where they are ready to recommend it to their friends. Furthermore, the answers to questions 11 to 15 indicate that all participants agree that CoMPers is appropriately able to customize conflict management and collaborative modeling, especially in terms of sending and receiving changes among collaborators. Results show the satisfaction of participants in the coordination on models within the CoMPers framework. They believe that CoMPers

Table 3
Support of Primary Collaboration Features in Collaborative Modeling Environments

	Change Propagation			*Conflict Management*		
Tool	Scenario	Flexibility	Specification	Detection	Resolution	Awareness
OBEO[16]	Offline	—	—	—	—	—
MetaEdit+ [34]	Offline	—	—	—	—	—
EMFStore[17]	Offline	—	Implicit Spec	Change Overlapping	Manual	Conflict Report
MDEForge [35]	Offline	—	Formalism	Constraint Violation	Manual	Conflict Report
VisualParadigm[18]	Offline	—	Implicit Spec	Change Overlapping	Semi-Automatic	Conflict Report
AToMPM [36]	Online	—	—	—	—	—
WebGME [37]	Online	—	Implicit Spec	Change Overlapping	Manual	Conflict Highlighting
EMFCollab[19]	Online	—	—	—	—	—
GenMyModel [38]	Online	—	—	—	—	Notification
Pyro [39]	Online	—	Implicit Spec	Constraint Violation	Manual	Conflict Highlighting
BUMBLE-CE [40]	Online	On-the-Fly	—	—	—	—
CDO[20]	Online | Offline	Before Starting	Implicit Spec	Change Overlapping	Semi-Automatic	Conflict Highlighting
MONDO [10]	Online | Offline	Before Starting	Implicit Spec	Constraint Violation	Automatic	—
CoMPers	Online & Offline	On-the-Fly	Implicit Spec, Pattern-Based	Pattern Matching, Change Overlapping	Automatic	Conflict Highlighting

Legend: *'&': AND, '|': OR, —: Not Supported*

properly supports teamwork and customization for collaboration, making it a useful platform for collaborative modeling. However, this study did not directly evaluate the tool's actual capabilities in supporting teamwork. Further research is needed to assess the real-world effectiveness of CoMPers in facilitating collaborative modeling and conflict resolution among teams. Moreover, we have not yet evaluated CoMPers through a large-scale case study, which would help us assess its ability to support collaboration on larger models and accommodate a large number of modelers. Moreover, enhancing the support for multi-view collaborative modeling will further enhance the usefulness of the CoMPers framework.

6.4. Threats to Validity

As is commonly the case with experiments which take part people, due to its design, the user experience study has threats to validity. In this section, we follow the guideline explained by Wohlin et al. [41] to analyze the potential threats to the validity of our experiments and results.

(1) Internal validity. The internal threat is that the modeling level of the participants could affect the results of the experiment [41]. We conducted the user study with students. While all students were graduate-level, their design behavior may not be identical to that of a real-world practitioner. In addition, we divided the participants into three groups. We mitigated this threat by defining the modeling level that the participants need to achieve and tried to make the modeling levels of the three groups as equal as possible to relieve this threat. Finally, the lack of negatively phrased statements in the questionnaire may leave it vulnerable to such biases, potentially compromising the internal validity of the results.

(2) Construct validity. The construct validity reflects the degree to which the case study truly represents what needs to be studied according to the research questions [41]. Two research questions of this paper are the participants' views about usefulness and ease of use of the proposed approach. When people take part in an experiment they might try to figure out what the purpose and intended result of the experiment is. In addition, participants can bias the results of study based on the expectation of experimenters. To ensure the validity of obtained results, we let three different groups, involving people at different levels of education (i.e., master students and PhD students) which have no or different expectations to the experiment.

(3) External validity. External validity. The external validity means that the conclusion drawn from existing examples in the experimental data set is also valid for out of set examples [41]. In the user experience evaluation, the design tasks assigned to the participants were not taken from actual projects. Instead, the tasks were based on the actual design documents in order to recreate realistic collaborative design scenarios. Furthermore, we used the benchmark consists of 23 purposefully modeling scenarios, which covers a broad collection of conflict scenarios. Although we believe that these scenarios cover the most common cases concerning the detection of conflicts, the conflict management module

should be further investigated in future by new conflicting scenarios. Finally, the observed design session lasted for approximately 30 minutes, which is short compared to real-world cases. The shorter sessions could have caused bias due to participants' low familiarity with the modeling editor or target system domain. To minimize this bias, a training workshop was conducted to introduce the modeling environment, and participants were given multiple design assignments using the target system model before the experiment. Moreover, the team size was small, consisting of only ten individuals. In a larger team, which modelers are directly involved in a higher-order design conflict may become ambiguous.

7. Related Work

In the MDE community, a lot of research has been dedicated to supporting conflict management in the model merging process. As surveyed by Sharbaf et al. [8], most contributions focus on conflict detection and resolution, such as DiCoMEF [42], AMOR [17], SuperMod [43], while others propose conflict prevention techniques, such as MOMM [44] and recently work has been done by David and Syriani [45]. However, the use of conflict management techniques to address multi-user collaboration in modeling environment can be roughly classified into another research direction.

In the last decade, several modeling environments have been proposed to support multi-user collaboration. Each environment uses a specific mechanism to send and receive change operations between users. However, there is no single tool that allows users to personalize change propagation. The main strength of CoMPers is that it supports personalized change propagation for both online and offline collaborations while handling conflict for collaborative modeling.

Table 3 summarizes the support for features of *change propagation* and *conflict management* in existing popular environments in the last decade that could be used for collaborative modeling in academia or industry. We classify the identified environments based on the following features defined in the conflict management taxonomy [8]: *Scenario*, which specifies the collaboration type that the environment supports for change propagation; *Flexibility*, which defines whether the environment provides flexibility to the user to choose a particular mechanism for propagation of changes; *Specification*, which determines technology that the environment follows to describe the conflict situation; *Detection*, which specifies whether the environment provides any mechanisms to find the conflicts occurring during collaboration; *Resolution*, which defines mechanism that the environment follows to resolve the detected conflicts; and finally, *Awareness*, which depicts the ability of environment to warn the user of the conflict situation or conflictual changes. Note that we only investigated the features that the environment addresses, and those provided by external extensions or user effort are not the focus of this study.

Table 3 shows that most of popular collaborative modeling environments support either an online or offline scenario. Only the CDO and MONDO environments support both online and offline collaboration scenarios. However, in both environments, the user must select either an online or offline collaboration method for sending and receiving changes in all the models of a repository at the beginning of collaboration. In this context, BUMBLE-CE provides a persistency driver during real-time change propagation, allowing users perform a commit and push operation on-the-fly. BUMBLE-CE allows modelers to start and terminate on-demand real-time collaboration on a model and work on the head revision of the model by sending and receiving edit operations in real-time. On the other hand, CoMPers supports personalized change propagation at the model level, which allows users to collaborate in real-time, offline or a combination of both real-time and offline strategies. In CoMPers, modelers can select different sending and receiving methods for work on different models, as well as switch to other collaboration strategies during the modeling process.

In regards of conflict management, some environments such as OBEO, AToMPM, MetaEdit+, EMFCollab, and BUMBLE-CE have not implemented conflict handling at all, and GenMyModel only provides an awareness technique. Other environments, such as WebGME, EMFStore, MDEForge, Pyro, and Visual Paradigm, support a specific conflict detection approach, and most of them require human involvement to resolve conflicts. In this context, MONDO and CoMPers are the only ones that have provided automatic approaches to resolve conflicts. Mondo automates conflict resolution based on design-space exploration techniques. It identifies the set of individual and atomic editing operations on the two branches to be merged and automatically produces a set of conflict-free states by applying patterns that ensure a proper order to find valid model states in offline collaboration. However, to apply the pattern, mappings between the pattern instance and the domain metamodel need to be established by the designer. In CoMPers, we not only support pattern-based resolution but also provide more flexibility by equipping the duel algorithm with three metamodel-independent resolution strategies. Users can select the most suitable one as the default resolution strategy before beginning an online/offline collaboration. Moreover, CoMPers is the only environment that supports all mandatory aspects of conflict management and allows users to customize it based on their needs.

According to results of last systematic update on collaborative MDSE environment reported by David et al. [14], most collaborative modeling environments only support offline collaboration scenario. In addition, most existing environment worked with manual conflict resolution and rarely provide a Comprehensive conflict management mechanism. In this context, enabling techniques, such as personalized change propagation and configurable conflict management which proposed by the CoMPers framework can support these two features that has become a pressing need. Recently David et al. [46] performed a systematic survey of practices and needs in industry for collaborative MDSE. Real-time and offline collaboration, a history of changes applied to

models, collaboration at the model level, model integration, and conflicts table are the most important features of collaborative MDSE supported by CoMPers. These features address the needs frequently reported by the industry.

8. Conclusion

In this paper, we have introduced CoMPers as the main contribution of this work. We first defined a collaboration scheme to describe how CoMPers can support collaboration among a team. Following this scheme, we presented the CoMPers platform based on the personalized change propagation that helps collaborators to customize concurrent work on models. Finally, we introduced a configurable conflict management approach based on the duel algorithm and integrated it with CoMPers to ensure consistent synchronization of models. The CoMPers conflict management framework consists of two primary modules that can be customized to detect and resolve the conflicts based on change overlapping and pattern matching techniques. To evaluate our proposals, we implemented the CoMPers framework and conducted two experiments to assess the ability of CoMPers to handle conflicts and support collaborative modeling. The results confirm that consistent and concurrent modeling in the collaborative CoMPers platform is not only possible but also effective, with acceptable functionality for approximately ten collaborators.

As future work, we aim to expand the conflict detection module of CoMPers to include other conflict detection techniques, such as constraint violation and formal methods. Moreover, the suggested conflict pattern language can be expanded to automatically convert into conflict detection rules. A key area for future work is to formally verify the safety and liveness properties of the CoMPers framework. Exploring the necessity for conflict tolerance to facilitate collaborative resolution at a later stage is another direction for future enhancements of the CoMPers conflict management framework. Additionally, the CoMPers framework can be enhanced to identify and resolve inconsistencies among multiple views.

References

[1] J. Whittle, J. Hutchinson, M. Rouncefield, The state of practice in model-driven engineering, IEEE software 31 (3) (2013) 79–85.

[2] M. Franzago, D. Di Ruscio, I. Malavolta, H. Muccini, Collaborative model-driven software engineering: a classification framework and a research map, IEEE Transactions on Software Engineering 44 (12) (2017) 1146–1175.

[3] E. Syriani, Framework to model collaboratively, in: International Workshop on Collaborative Modelling in MDE, Vol. 1717, 2016.

[4] M. Brambilla, J. Cabot, M. Wimmer, Model-driven software engineering in practice, 2nd Edition, Synthesis Lectures on Software Engineering, Morgan & Claypool Publishers, San Rafael, CA, USA, 2017.

[5] N. Kanagasabai, O. Alam, J. Kienzle, Towards online collaborative multi-view modelling, in: International Conference on System Analysis and Modeling, Springer, 2018, pp. 202–218.

[6] J. Corley, E. Syriani, H. Ergin, S. Van Mierlo, Cloud-based multi-view modeling environments, in: Modern software engineering methodologies for mobile and cloud environments, IGI Global, 2016, pp. 120–139.

[7] M. Sharbaf, B. Zamani, G. Sunyé, Towards personalized change propagation for collaborative modeling, in: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, 2021, pp. 3–7.

[8] M. Sharbaf, B. Zamani, G. Sunyé, Conflict Management Techniques for Model Merging: A Systematic Mapping Review, Software and Systems Modeling (In press).
URL https://hal.archives-ouvertes.fr/hal-03787436

[9] R. A. Kastler, S. Hansen, P. J. M. Arizmendiarrieta, D4 . 4 – Prototype tool for collaboration, Tech. rep., MONDO Project (2016).

[10] A. Bagnato, E. Brosse, A. Sadovykh, P. Maló, S. Trujillo, X. Mendialdua, X. De Carlos, Flexible and scalable modelling in the mondo project: Industrial case studies, XM@ MoDELS 14 (2014) 42–51.

[11] P. Langer, M. Wimmer, A benchmark for conflict detection components of model versioning systems, Softwaretechnik-36 Trends 33 (2) (2013) 91–94.

[12] J. Hutchinson, M. Rouncefield, J. Whittle, Model-driven engineering practices in industry, in: Proceedings of the 33rd International Conference on Software Engineering, 2011, pp. 633–642.

[13] D. Di Ruscio, M. Franzago, I. Malavolta, H. Muccini, Envisioning the future of collaborative model-driven software engineering, in: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), IEEE, 2017, pp. 219–221.

[14] I. David, K. Aslam, S. Faridmoayer, I. Malavolta, E. Syriani, P. Lago, Collaborative model-driven software engineering: a systematic update, in: 2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS), IEEE, 2021, pp. 273–284.

[15] S. Barrett, P. Chalin, G. Butler, Model merging falls short of software engineering needs, in: Proc. of the 2nd Workshop on Model-Driven Software Evolution, Citeseer, 2008.

[16] K. Altmanninger, M. Seidl, M. Wimmer, A survey on model versioning approaches, International Journal of Web Information Systems (2009).

[17] P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, M. Wimmer, An introduction to model versioning, in: M. Bernardo, V. Cortellessa, A. Pierantonio (Eds.), International School on Formal Methods for the Design of Computer, Communication and Software Systems, Lecture Notes in Computer Science, vol 7320, Springer, Berlin, Heidelberg, 2012, pp. 336–398. doi:10.1007/978-3-642-30982-3_10.

[18] M. Sharbaf, B. Zamani, G. Sunyé, A formalism for specifying model merging conflicts, in: Proceedings of the 12th System Analysis and Modelling Conference, 2020, pp. 1–10.

[19] T. Mens, A state-of-the-art survey on software merging, IEEE transactions on software engineering 28 (5) (2002) 449–462.

[20] X. Zhao, C. Liu, Y. Yang, W. Sadiq, Aligning collaborative business processes — an organization-oriented perspective, IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 39 (6) (2009) 1152–1164.

[21] E. Kuiter, S. Krieter, J. Krüger, G. Saake, T. Leich, varied: an editor for collaborative, real-time feature modeling, Empirical Software Engineering 26 (2) (2021) 24.

[22] C. Debreceni, G. Bergmann, I. Ráth, D. Varró, Enforcing fine-grained access control for secure collaborative modelling using bidirectional transformations, Software & Systems Modeling 18 (3) (2019) 1737–1769.

[23] D. S. Kolovos, R. F. Paige, The Epsilon pattern language, in: International Workshop on Modelling in Software Engineering (MiSE), IEEE, 2017, pp. 54–60. doi:10.1109/MiSE.2017.8.

[24] D. S. Kolovos, R. F. Paige, F. A. Polack, On the evolution of ocl for capturing structural constraints in modelling languages, in: Rigorous Methods for Software Construction and Analysis, Springer, 2009, pp. 204–218.

[25] R. F. Paige, D. S. Kolovos, L. M. Rose, N. Drivalos, F. A. Polack, The design of a conceptual framework and technical infrastructure for model management language engineering, 14th IEEE International Conference on Engineering of Complex Computer Systems, IEEE, 2009, pp. 162–171. doi:10.1109/ICECCS.2009.14.
URL http://ieeexplore.ieee.org/document/5090524/

[26] M. Sharbaf, B. Zamani, G. Sunyé, Automatic resolution of model merging conflicts using quality-based reinforcement learning, Journal of Computer Languages (2022) 101123.

[27] M. Sharbaf, B. Zamani, Configurable three-way model merging, Software: Practice and Experience 50 (8) (2020) 1565–1599.

[28] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF: Eclipse Modeling Framework, Addison-Wesely, 2009.

[29] P. Langer, M. Koegel, Web-based Modeling Tools Theia and Che, Tech. rep., EclipseSource (2019).

[30] Helming, Jonas, and Langer, Philip and Koegel, Maximilian, Introducing the Graphical Language Server Protocol / Platform (Eclipse GLSP), https://eclipsesource.com/blogs/, online; accessed 27 July 2022 (2021).

[31] K. Altmanninger, P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, M. Wimmer, Why model versioning research is needed!? an experience report, in: Proceedings of the MoDSE-MCCM 2009 Workshop@ MoDELS, Vol. 9, 2009, pp. 1–12.

[32] P. Brosch, P. Langer, M. Seidl, K. Wieland, M. Wimmer, Colex: a web-based collaborative conflict lexicon, in: Proceedings of the 1st International Workshop on Model Comparison in Practice, 2010, pp. 42–49.

[33] M. Keil, P. M. Beranek, B. R. Konsynski, Usefulness and ease of use: field study evidence regarding task considerations, Decision support systems 13 (1) (1995) 75–91.

[34] S. Kelly, Collaborative modelling with version control, in: Federation of International Conferences on Software Technologies: Applications and Foundations, Springer, 2017, pp. 20–29.

[35] J. Di Rocco, D. Di Ruscio, A. Pierantonio, J. S. Cuadrado, J. d. Lara, E. Guerra, Using atl transformation services in the mdeforge collaborative modeling platform, in: International Conference on Theory and Practice of Model Transformations, Springer, 2016, pp. 70–78.

[36] E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. Van Mierlo, H. Ergin, Atompm: A web-based modeling environment, in: Joint proceedings of MODELS'13 Invited Talks, Demonstration Session, Poster Session, and ACM Student Research Competition co-located with the 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013): September 29-October 4, 2013, Miami, USA, 2013, pp. 21–25.

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

[38] M. Dirix, A. Muller, V. Aranega, Genmymodel: an online uml case tool, in: ECOOP, 2013.

[39] P. Zweihoff, S. Naujokat, B. Steffen, Pyro: generating domain-specific collaborative online modeling environments, in: Fundamental Approaches to Software Engineering: 22nd International Conference, FASE 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings 22, Springer, 2019, pp. 101–115.

[40] K. Aslam, Y. Chen, M. Butt, I. Malavolta, Cross-platform real-time collaborative modeling: an architecture and a prototype implementation via emf. cloud, IEEE Access (2023).

[41] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Science & Business Media, 2012.

[42] A. A. Koshima, V. Englebert, Collaborative editing of emf/ecore meta-models and models: Conflict detection, reconciliation, and merging in dicomef, Science of Computer Programming 113 (2015) 3–28.

[43] J. Schröpfer, F. Schwägerl, B. Westfechtel, Consistency control for model versions in evolving model-driven software product lines, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, 2019, pp. 268–277.

[44] U. Mansoor, M. Kessentini, P. Langer, M. Wimmer, S. Bechikh, K. Deb, Momm: Multi-objective model merging, Journal of Systems and Software 103 (2015) 423–439.

[45] I. David, E. Syriani, Real-time collaborative multi-level modeling by conflict-free replicated data types, Software and Systems Modeling (2022) 1–20.

[46] I. David, K. Aslam, I. Malavolta, P. Lago, Collaborative model-driven software engineering—a systematic survey of practices and needs in industry, Journal of Systems and Software 199 (2023) 111626.

Highlights:

1. A Customizable and Extensible Conflict Management Framework for Collaborative Modeling
2. Personalized Change Propagation for Collaborative Modeling
3. Duel Algorithm for Automatically Conflict Detection and Resolution
4. Proof-of-Concept Implementation in EMF.cloud Modeling Environment

Mohammadreza Sharbaf is an Assistant Professor in the Computer Engineering at the University of Isfahan. He is interested in Model-Driven Software Engineering (Domain-Specific Modeling Languages, Model Management, and Model Transformation), Low-Code Development Platforms, Software Development Methodologies, Design Patterns, and Semantic Web (Semantic Reasoning). His current research is focused on collaborative modeling, in particular, worked on model comparison and merging, conflict management, model versioning, and multi-view modeling. Mohammadreza received his B.Sc. from the Isfahan University of Technology, Isfahan, Iran, in 2013, and his M.Sc and Ph.D from the University of Isfahan, Isfahan, Iran in 2016 and 2022, both in Computer Engineering (Software). Now, he is the director of Model-Driven Software Engineering Research Group (MDSERG) at University of Isfahan.

Bahman Zamani received his B.Sc. from the University of Isfahan, Isfahan, Iran, in 1991, and his M.Sc. from the Sharif University of Technology, Tehran, Iran, in 1997, both in Computer Engineering (Software). He obtained his Ph.D. in Computer Science from Concordia University, Montreal, QC, Canada, in 2009. From 1998 to 2003, he was a researcher and faculty member of the Iranian Research Organization for Science and Technology (IROST)—Isfahan branch. Dr. Zamani joined the Faculty of Computer Engineering at the University of Isfahan in 2009, as an Assistant Professor. In 2018, he was promoted to Associate Professor. His main research interest is Model-Driven Software Engineering (MDSE). He is the founder of MDSE research group at the University of Isfahan, and he acted as the director of this group until September 2023. Dr. Zamani was retired on September 2022.

Gerson Sunyé is an associate professor at the University of Nantes in the domain of software engineering and distributed architectures and the head of the Nantes Software Modeling Group. He received the Ph.D. degree in Computer Science from the University of Paris 6, France, in 1999. From 1999 to 2001 he was a postdoctoral researcher at the IRISA Computer Science laboratory. He has 4 years of industry experience in software development. He received his Habilitation in 2014. He is author of several papers in international conferences and journals in software engineering. His research interests include software testing, design patterns and large-scale distributed systems.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: