Real-time collaborative multi-level modeling by conflict-free replicated data types

Istvan David 1 · Eugene Syriani 1

Received: 29 April 2022 / Revised: 4 August 2022 / Accepted: 25 September 2022 / Published online: 1 November 2022
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2022

Abstract
The need for real-time collaborative solutions in model-driven engineering has been increasing over the past years. Conflict-free replicated data types (CRDT) provide scalable and robust replication mechanisms that align well with the requirements of real-time collaborative environments. In this paper, we propose a real-time collaborative multi-level modeling framework to support advanced modeling scenarios, built on a collection of custom CRDT, specifically tailored for the needs of modeling environments. We demonstrate the benefits of the framework through an illustrative modeling case and compare it with other state-of-the-art modeling frameworks.

Keywords Collaborative modeling · Real-time collaboration · Multi-level modeling · Conflict-free replicated data types · Model-driven engineering

1 Introduction

Collaborative Model-Driven Software Engineering (MDSE) [44] aims to establish a sound interplay between physically distanced stakeholders by combining the techniques of collaborative software engineering [66] and model-driven techniques [50]. Recent systematic studies [17,28] report a substantial shift towards real-time collaboration in MDSE. While real-time collaborative MDSE opens up many opportunities, it also gives rise to unique challenges, primarily: ensuring appropriate convergence of distributed data, while still guaranteeing timely execution [55].

Optimistic replication has been suggested by [49] as a possible treatment, in which the local replicas of stakeholders are allowed to diverge temporarily. This divergence is admissible due to eventual consistency mechanisms [65] ensuring that each remote change will be observed by every stakeholder eventually and replicas converge. Strong eventual consistency (SEC) [45] augments eventual consistency with a safety guarantee: two replicas that have received the same set of change updates will be in the same state, regardless of the order of updates. Although SEC aligns well with the requirements of real-time collaborative MDSE settings, it is not trivial to implement correctly [23].

Conflict-free Replicated Data Types (CRDT) have been suggested by Shapiro et al. [52] as a scalable implementation of SEC. CRDT have been traditionally geared to support linear data, such as text. Since traditional software engineering relies on textual artifacts to persist source code, mechanisms of real-time collaborative textual editors can address the main challenges of real-time collaborative source code development. However, MDSE relies on richer data types, such as multigraphs, that are also potentially disconnected. Thus, CRDT cannot efficiently accommodate models as first-class citizens. Existing solutions either (i) focus exclusively on textual modeling and reduce collaboration to textual primitives which are well supported by current CRDT frameworks [48]; or (ii) work on models of limited complexity and do not support proper graph semantics at the data level [25]. As a consequence, current CRDT-based techniques fall short of supporting intricate modeling scenarios.

Such an intricate modeling scenario is multi-level modeling (MLM) [4]. MLM attempts to overcome the limitations of traditional modeling architectures, such as the OMG’s
Meta-Object Facility (MOF)\textsuperscript{1} [5,6,30], stemming from their restricted, two-level meta-modeling approach. While two meta-layers might be enough for simple modeling cases, such architectures fail to support scenarios in which different kinds of conformance might be required, or in which language designers need the same level of control over indirect instances two or more meta-levels below [20]. Misusing two-level modeling architectures to emulate multi-level modeling gives rise to undesirable modeling problems. Such cases have been reported by [11] who point out that 85% of wiki knowledge base classes allow for the inference of unsound conclusions. To react to these issues, MLM approaches increase the flexibility of the meta-modeling architecture by allowing an arbitrary number of meta-levels [20].

Augmenting multi-level modeling with real-time collaboration capabilities enables teamwork between stakeholders acting (i) at different levels of abstraction, or (ii) at different levels of decision making. Typical examples include (i) changing a modeling language during operation to incorporate language elements required by the technical stakeholders [33]; and (ii) restricting values of attributes at higher levels of decision making and enforcing these values through the notion of potency [61]. However, current collaboration frameworks are limited to two-level modeling architectures, and no sound practices exist to engage in real-time collaborative multi-level modeling among distributed stakeholders [17,28].

\textbf{Contributions} The main contribution of our work is a novel real-time collaborative multi-level modeling framework, called lowkey.\textsuperscript{2} The framework supports a wide range of modeling scenarios across an arbitrary number of modeling and linguistic meta-levels. We achieve this flexibility by separating the linguistic metamodel(s) from the physical metamodel. In this setting, domain-specific models always conform to their linguistic metamodels, and they both (models and metamodels) conform to the uniform physical metamodel provided by the framework. Real-time synchronization between collaborating stakeholders is achieved by persisting the physical metamodel in CRDT that ensure the consistency of domain-specific models in a domain-agnostic fashion. This makes lowkey especially suitable for supporting approaches such as multi-view modeling [46]. Furthermore, lowkey subsumes traditional modeling frameworks, such as UML [27] and EMF [54], offering a promising integration potential with existing model editors.

\textbf{Structure} The rest of this paper is structured as follows. First, in Sect. 2, we provide an example to illustrate the concepts throughout the paper. In Sect. 3, we give an overview of the background of our work and review the related work. In Sect. 4, we present lowkey, our framework for real-time collaborative modeling. In Sect. 5, we demonstrate the feasibility of the framework by applying it on the case study. In Sect. 6, we discuss the lessons learned and compare lowkey with similar frameworks. Finally, in Sect. 7, we draw the conclusions.

\section{Illustrative case}

We rely on an illustrative case of developing a collaborative editor for modeling Mind maps [13]. The case highlights multiple structural facets of (meta)modeling, such as typing, inheritance, and various forms of well-formedness through the collaborative modeling of mind maps. We show how an editor built on top of lowkey would enable this.

The metamodel of the case is shown in Fig. 1. MindMap serves as the root element, containing the various types of \texttt{Topics}, and the \texttt{Markers} that can be associated with specific \texttt{Topics}. The mindmap is hierarchical: the single \texttt{CentralTopic} further contains an arbitrary number of \texttt{MainTopics}, each containing an arbitrary number of \texttt{SubTopics}. Finally, \texttt{SubTopics} can contain \texttt{SubTopics} to arbitrary depths.

Based on the metamodel, the language engineer must provide an editor that enables real-time collaborative modeling of a MindMap instance. This entails the following operations: creating, editing, and deleting instances of metamodel concepts; creating, editing, and deleting links between elements. In addition, a mechanism for reading the state of the local model is required.

\textbf{Scenarios} Here are some modeling scenarios that may occur as a result of applying these CRUD operations.

\textbf{Cooperation} User A creates a MindMap instance and User B sets its title.

\textbf{Cooperation with of linguistic inconsistencies} User A creates a CentralTopic instance, but does not link it to the MindMap instance. User B links the CentralTopic instance to the MindMap via a composition reference.

\textbf{Conflict management} User B removes the CentralTopic instance; in parallel, however, User A creates a MainTopic instance and links it to the CentralTopic instance.

\textbf{Multi-level cooperation} Users A and B have created multiple Markers, and would like to categorize them (e.g.,

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{metamodel.png}
\caption{The metamodel of the Mind maps}
\end{figure}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{Topic} & \textbf{SubTopics} \\
\hline
\texttt{MainTopic} & \texttt{SubTopics} \\
\hline
\texttt{SubTopic} & \texttt{SubTopics} \\
\hline
\end{tabular}
\caption{The metamodel of the Mind maps}
\end{table}

\footnote{1 https://www.omg.org/mof/.
\footnote{2 https://github.com/geodes-sms/lowkey.}
as textual and graphical). The Language Designer, who works one metalevel above Users A and B, sets the potency of the Marker type from 1 to 2; thus, allowing a templating mechanism for A and B.

Multi-level integrity Subsequently, the Language Designer decides to revoke this level of flexibility from the users and sets the potency of the Marker class to 1, rendering the inheritance chain inconsistent.

Requirements Based on the requirements for effective real-time collaboration defined by Sun et al. [55], we formulate the following requirements for a solution in the context of this illustrative case. These requirements are evaluated in the feasibility study in Sect. 5.

R1. Convergence, i.e., every stakeholder’s local data must exhibit the same state after updates have been applied.
R2. Timely execution, i.e., operations must propagate and the system must reconcile within a deadline to provide a smooth user experience on the client side.
R3. Causality preservation, i.e., updates must be ordered in the same causal way by each stakeholder.
R4. User intention preservation, i.e., the original user’s intention must be preserved.
R5. Multi-level collaboration, i.e., ability to collaborate and resolve inconsistencies across different meta-levels.
R6. Meta-model reuse, i.e., tool builders have to be able to use previously defined meta-models with only minimal adaption.
R7. Tolerance of linguistic inconsistencies, i.e., the ability to synchronize and persist models into the physical representation despite linguistic non-conformance of model elements.

3 Background

We discuss the existing work related to collaborative modeling.

3.1 Collaborative MDSE

Collaborative software engineering enables effective cooperation among stakeholders [66], often in distributed settings [32]. Distributed teams introduce challenges to collaboration in terms of processes, project management, artifact sharing, and consistency [42]. These challenges are further exacerbated in the engineering of complex software-intensive systems that require collaboration between stakeholders of highly diverse expertise. MDSE [50] provides stakeholders with techniques for reasoning about the system at higher levels of abstraction than source code. As the combination of collaborative software engineering and MDSE, collaborative MDSE exhibits the traits of both disciplines. Collaborative MDSE has become a prominent feature of nowadays’ software engineering practice [10]. Version control for modeling artifacts has been extensively employed to facilitate collaboration. Such approaches rely either on lock mechanisms [35] or manual conflict resolution [59]. As a consequence, they are not suitable for real-time collaboration.

3.2 Real-time collaboration

Recent studies on collaborative model-driven software engineering show a strong shift towards real-time collaboration [17,28]. The main challenge in such a shift is ensuring appropriate convergence of distributed replicas, while still guaranteeing timely execution [55]. Convergence of replicas is ensured by the consistency model a distributed setting chooses. Strict consistency is a theoretical model for guaranteeing deterministic consistency by the total order of change updates that are exchanged instantaneously. However, due to its limited usability, various relaxations have been provided, such as sequential consistency [37], causal consistency [26], and eventual consistency [65]. Strong eventual consistency (SEC) [45] augments the liveness property of eventual consistency (all change updates will be observed eventually) with a safety guarantee: two nodes that have received the same set of change updates will be in the same state, regardless of the order of updates. SEC is an efficient resolution of the CAP theorem by [12], suggesting that distributed systems cannot provide more than two out of the three properties of strong consistency, availability, and partition tolerance. SEC removes the problem of conflict resolution on local replicas by introducing rules to ensure a unique outcome for concurrent changes, deterministically resolving any conflict. There is no need for a consensus or synchronization since any kind of change is allowed and conflicts are removed altogether. As such, this consistency model is especially appropriate for real-time collaboration. Nevertheless, SEC may be challenging or even impossible to implement for certain data types.

3.3 Conflict-free replicated data types

CRDT eliminate conflicts between the distributed stakeholders’ operations, thus avoiding the complexity of conflict resolution and roll-back. As a result, CRDT exhibit promising fault tolerance and reliability properties.

CRDT come in two flavors. State-based CRDT are structured in a way that they adhere to a monotonic semi-lattice. Shapiro et al. [52] show that state-based objects that satisfy this property are SEC; hence, they converge to a consistent state. Operation-based CRDT require that concurrent operations are commutative. Independent from the order of the received change updates, the state of the local copy converges
to the same state. To achieve such a behavior, the supporting communication protocol has to provide a causal ordering mechanism, such as global timestamps. In our approach, we have opted for the operation-based CRDT scheme because of its reduced costs when exchanging model changes between replicas.

Operation-based model representation [38] has been proposed for reasoning about streams of model operations. The C-Praxis approach [41] defines six CRUD model operations and defines how these operations interact. The CRDT of lowkey are geared towards the more complex semantics of graphs and, thus, they provide a superset of these operations. Additionally, lowkey enables working with multigraphs, hypergraphs, and disconnected graphs, allowing for more flexibility in modeling. Traditional operation-based approaches such as C-Praxis are built on MOF. Consequently, they fall short of supporting arbitrary meta-levels of modeling.

### 3.3.1 The last-writer-wins (LWW) paradigm

To ensure the convergence of local replicas, their differences have to be resolved in an automated fashion. Such a resolution mechanism can be implemented either in the application or at the data level [40]. The LWW paradigm [34] has been widely adopted as a data-level implementation of operation-based conflict resolution [47, 60]. Conflicting operations are resolved using a global ordering operator, e.g., a timestamp. Given two changes, it is the more recent one that will prevail [51]. To avoid potential data loss, each change update is stored locally and the resolution of conflicts is carried out by the local replica.

Figure 2 shows an example resolution scenario under LWW. User A (top blue) and User B (bottom green) initially have their local replicas in consistent states: the value of $x$ and its timestamp $t$. At $t = 1$, User A executes the update $x = 15$ on his local copy. A message with this updated value and the timestamp is sent to User B. However, before the message arrives, User A executes another update: $x = 20$, at time $t = 2$. Again, an update message is composed and sent to User B. Due to network delays, the second update arrives to User B earlier than the first. Upon receiving the update message, User B will reconcile this new value with his local replica. As it stands, User B has $x = 10$ timestamped with $t = 0$; and an update that says $x = 20$ timestamped with $t = 2$. Under the LWW paradigm, the latter value prevails, due to the more recent timestamp. Eventually, the first message arrives. User B has $x = 20$ timestamped with $t = 2$; and an update of $x = 15$ timestamped with $t = 1$. Under the LWW paradigm, the former value should prevail. Thus, the update is not performed on the local copy. Eventually, the replicas are in consistent states: $x = 20$.

Despite the omitted update after receiving $x = 15$, $t = 1$, the message can still be stored in User B’s local replicas. It is the responsibility of the CRDT implementation to decide whether to store outdated data or not. Some use-cases might require such behavior. However, the performance of CRDT is proportional with the data they store [57] and often requires implementing complex garbage collection mechanisms [8].

The LWW paradigm satisfies the requirements for real-time collaboration defined by Sun et al. [55]: (i) convergence, i.e., every stakeholder’s local data must exhibit the same state after updates have been applied; (ii) user intention preservation, i.e., the original user’s intention must be preserved; (iii) causality preservation, i.e., updates must be ordered in the same causal way by each stakeholder; and (iv) timely execution, i.e., operations must propagate and the system must reconcile within a deadline that provides a smooth user experience.

### 3.3.2 CRDT frameworks

Yjs\(^3\) is an open-source framework for peer-to-peer shared editing of structured data, such as rich-text, or XML. Operations are stored in a linked list, resulting in a total order of operations, thus implementing CRDT. Teletype\(^4\) provides string-wise sequence CRDT for peer-to-peer collaborative editing in the Teletype for Atom\(^5\) cooperative source code development environment. AutoCouch\([31]\) is a JSON framework combining the benefits of the Automerge CRDT library\(^6\) and the CouchDB\(^7\) database engine. Conflict-free JSON documents are replicated both on the server side and client side, while ensuring a responsive real-time user experience for web-based applications.

These frameworks are similar to lowkey in their aim to augment engineering tools with a CRDT-based collaboration service. However, they are primarily geared toward linear data types, whereas the CRDT layer of lowkey is primar-
ily aimed at supporting a wide range of modeling scenarios. Currently, no other modeling framework implements support for graph CRDT as first-class citizens. Shapiro et al. [52] formalize graph CRDT but provide no implementation. SyncMeta [25] (built on top of Yjs) provides support for modeling graph-like data structures. However, graphs are emulated by linked lists and string comparisons at the CRDT level.

### 3.4 Multi-level modeling

Multi-level modeling (MLM) is a paradigm aiming to improve the flexibility of modeling architectures by introducing arbitrary meta-levels [4]. A formal framework of modeling at arbitrary meta-levels has been developed by [3], introducing the core concepts of deep instantiation [5] and deep characterization [6]. Deep instantiation extends the traditional two-level instantiation and allows classes to be instantiated transitively. This is achieved by the notion of potency that defines how many levels of instantiation the class supports. Deep characterization allows meta-types to influence the characteristics of their instances beyond those in the level immediately below. Our framework embraces these concepts to provide collaborative support in a highly generalized fashion. As shown by [5], deep metamodeling can naturally accommodate traditional modeling frameworks built on shallow instantiation, such as UML and EMF. Consequently, our framework is a good fit with modeling tools supporting UML and EMF modeling, but lacking collaborative features.

The feasibility of multi-level modeling has been demonstrated in tools such as MetaDepth [20] and Melanee [2]. However, as pointed out recently by [36], the divergence of MLM techniques has led to numerous different interpretations of the nature of levels, their purpose, the appropriateness of implementation techniques, etc. Often, the dual ontological and linguistic instantiation of the Orthogonal Classification Architecture (OCA) [7] allows elements to be typed by linguistic concepts and domain concepts at the same time [1,5]. In this paper, we focus on a third orthogonal dimension, the physical dimension that deals with the physical representation of concepts. The physical-logical distinction was first introduced by [5]. Specifically, we build on the adapted version of the physical metamodel defined by [61]. The OCA usually omits the physical dimension, lumping it into the linguistic dimension and relying on its physical layer for representation and serialization (e.g., XMI in MOF). However, in collaboration techniques operating at the data level—such as CRDT—the management of consistency is achieved at the physical level of models. This necessitates the separation of the physical and linguistic dimensions. The explicit physical dimension enables ensuring physical consistency while allowing linguistic inconsistencies. Temporary linguistic inconsistencies are often desirable, especially when complex modeling operations are used, or when atomic modeling steps do not result in a linguistically valid model (see Scenario 2 in Sect. 5.2). For the sake of conciseness, we choose to treat the physical dimension of [5] and the linguistic dimension of the OCA explicitly, and treat the ontological dimension implicitly. However, because CRDT operates at the data level, our framework is able to treat both linguistic and ontological conformance.

### 4 A framework for real-time collaborative metamodeling

We present the different components of lowkey, our real-time collaborative modeling framework.

#### 4.1 Architecture

Figure 3 outlines the architecture of the framework. In a typical modeling setting, users are provided with (domain-specific) Editors that enable interacting with the models of the linguistic modeling layer, through a generated domain-specific API. The linguistic modeling layer enforces well-formedness rules defined by the static semantics of the language. Each element at the linguistic modeling layer conforms to a physical metamodel, as described by Van Mierlo et al. [61]. The physical modeling layer is responsible for the uniform representation of objects of the linguistic models and metamodels. Instances of the physical metamodel are serialized and propagated to the collaborating stakeholders through their network interface which communicates with the server. Other clients receive these instances from the server and merge them into the local working data represented in CRDT. The command language of the physical modeling layer enables a uniform treatment of local updates (from the linguistic modeling layer), and remote updates (from the network interface).

#### 4.2 Linguistic modeling layer

The linguistic modeling layer provides mechanisms for modeling at arbitrary levels of abstraction. As a consequence, this layer contains every domain model and its metamodels the Editor manipulates. To this end, a domain-specific API is generated for the metamodels. In the illustrative case, the MindMap class and its instance(s) are situated at this modeling layer; and methods for CRUD operations are generated.

The main responsibility of this layer is to enforce the language-specific well-formedness rules defined by the metamodel and static semantics. Typical examples of language-specific well-formedness rules include multiplicities, transi-
4.3 Physical modeling layer

The main responsibility of the physical modeling layer is to provide a metamodel that every linguistic concept can conform to, regardless of which linguistic meta-level they are situated at. For example, the physical metamodel has to accommodate both the MindMap class; its instance mindmap_0; and the Class class the MindMap class corresponds to. In addition, the physical modeling layer may impose language-independent well-formedness rules, such as each model having to exhibit a graph structure.

4.3.1 Physical metamodel

We adopt the physical metamodel from previous work [61], as shown at the bottom of Fig. 4.

Node is the foundational concept of the metamodel, which can be organized into Models. Models, in turn, are Nodes themselves, allowing for the hierarchical composition of Models.

A consequence of metamodeling—and multi-level modeling in particular—is that instantiable model elements can play the role of both instances and types [3]. To accommodate this dichotomy, Clabjects serve as the physical metatype for every linguistic class and instance. For example, in the mindmap metamodel, both the CentralTopic class and its instance(s) are mapped onto the Clabject physical type.

Associations link Clabjects to each other. The Association inherits from the Clabject, and transitively from the Node. Due to the latter, Associations can be typed by other Nodes. Due to the former, Associations can be abstract and link other Associations. We allow this flexibility to accommodate the models of various modeling frameworks and formalisms that typically implement a subset of these modeling options. UML, for example, restricts Associations from being abstract, and only allows linking Clabjects. Composition and Aggregation are specialized types of Association with conventional semantics.

Attributes store information of specific Nodes. Specifically, in Attributes, the type property maintains information...
about the linguistic type the Attribute corresponds to. For example, storing textual information (such as the name of a Topic in the illustrative case) in the Eclipse Modeling Framework (EMF) would mean the type of the Attribute is EString. Attributes are specializations of the Node type. The inherited typed_by relationship provides a mechanism for typing an Attribute at the physical level.

4.3.2 Need for an explicit physical metamodel

Instantiation mechanisms of current modeling frameworks typically consider two levels: classes and their instances (objects). Although this covers the majority of practical use cases, some scenarios might require multiple levels of type-instance relationships [22]. Such a type-instance hierarchy enables various beneficial mechanisms, such as deep instantiation and deep characterization [5]. The role of a physical metamodel is to represent the objects of such a type-instance hierarchy uniformly, irrespective of what metalevel a specific object is situated at; and to provide services such as the serialization of these objects. By this separation of concerns, objects of the type-instance hierarchy become independent of their physical representation, and their roles are determined by purely linguistic concepts, such as being metatypes or instances of each other. Therefore, we refer to this level of objects as the linguistic metamodel.

The physical metamodel of traditional modeling frameworks is typically coupled with the topmost meta-level of their linguistic metamodels. For example, Ecore serves as the core metamodel of EMF and also defines the rules governing the serialization of EMF models into XMI files.

By providing mechanisms for multi-level metamodeling, and clearly separating the physical metamodel from the linguistic metamodel, our framework is able to (i) support advanced metamodeling scenarios; and (ii) accommodate traditional modeling frameworks, for example, by restricting the flexibility of the physical metamodel. Similar avenues have been explored by multiple modern modeling frameworks, such as Melanee [2], metaDepth [20], and the Modelverse [62].

4.3.3 Command language

The physical metamodel can be accessed and interacted with through a command language. The brief definition of the language is shown in Listing 1. The command language defines four CRUD operations. Therefore, integrating an editor with the physical metamodel requires the appropriate facility that translates domain-specific modeling operations to the operations of the command language.

We have opted for providing an external textual DSL because such languages are more trivially serialized and propagated through standard network protocols than binary data.

4.4 CRDT layer

The CRDT layer is responsible for persisting instances of the physical metamodel. As shown in Fig. 4, each element of the physical metamodel is associated with exactly one CRDT. At run-time, as the classes of the physical metamodel get instantiated, a corresponding CRDT is instantiated as well. The model element maintains a reference to its persisting CRDT instance during execution. Our CRDT implementations follow the LWW paradigm and have been mainly implemented by following the specifications outlined by Shapiro et al. [51]. In the following, we briefly elaborate on these types.

4.4.1 Timestamps for total order

To ensure the commutativity of operation-based CRDT, an operator for total order is required. The most natural choice in a distributed setting is a global timestamping mechanism, as it makes the fewest assumptions about the system. lowkey uses the Unix epoch time in nanoseconds to timestamp model updates with. As an alternative, Lamport clocks [37] or vector clocks [53] can be used as suggested by Shapiro et al. [51]. Timestamped model updates are subsequently forwarded to the collaborating stakeholders. The global nature of the timestamp ensures that each local replica sorts the updates in the same order, ensuring the safety property of SEC (c.f. Sect. 3.3). The appropriate granularity of timestamps is paramount in supporting complex CRDT. Consider the underlying data structure in the illustrative example in Fig. 2 being a graph, in which vertices represent entities with associated attributes. Assigning a timestamp to the whole graph would not allow independent changes at finer-grained levels, such as vertices. Thus, we assign a timestamp to each instance of the elements of the physical metamodel, and update it on each CRUD operation.

Another useful operator could be the priority of messages, e.g., to define hierarchical stakeholder roles in which higher ranked roles can overwrite the changes of lower-ranked roles. An important difference between timestamps and priorities is the level of granularity. Timestamps (e.g., at the level of nanoseconds) provide better chances to unambiguously order two operations, as compared to priorities (e.g., assigned from a range between 1–5). Therefore, the ordering operator should be carefully designed to ensure the total order. Furthermore, compound ordering operators can be used as well. For example, the shortcomings of the priority operator can be circumvented by combining it with timestamps. Such compound operators are best defined by linking them through the absorption identity over the lattice of change operations [15] to ensure a valid and sound composition.
4.4.2 LWWRegister

The LWWRegister is the simplest CRDT in lowkey, containing a single value. To implement LWW semantics, its value \( v \) is equipped with a timestamp \( t \). Thus, an LWWRegister \( r \) is defined as \( r = (v, t) \).

The \textit{update} operation permits to modify the value of the register. Given a new timestamped value \( (v', t') \), the \textit{update} operation on register \( r = (v, t) \) is defined as \( r := (v', t') \) iff \( t' > t \), and NOP otherwise.

4.4.3 LWWSet

The LWWSet \( S \) contains an arbitrary number of timestamped values \((v, t)\) under set semantics. That is, for \( S = \{(v, t)\} \), it holds that \( S_1 = (v_1, t_1), S_2 = (v_2, t_2) \in S : v_1 \neq v_2 \).

We have implemented the LWWSet as an LWW-element-Set [51]. LWW-element-SetS are composed of the \textit{add-set} \( A \) and the \textit{remove-set} \( R \). The two sets are disjoint with the \textit{add-set} containing values added to \( S \), and the \textit{remove-set} containing values removed from \( S \). This structure ensures the commutativity of \textit{add} and \textit{remove} operations: \( \text{add}(v, t) \circ \text{remove}(v, t') \equiv \text{remove}(v, t') \circ \text{add}(v, t) \).

\textit{lookup} The \textit{lookup} operation indicates if an element is in the set. A value is considered to be in the LWWSet iff it can be found in the \textit{add-set}, and it cannot be found in the \textit{remove-set} with a higher timestamp. It is defined as \( \text{lookup}(v) \equiv \text{true} \) if \( \exists v, t \ni (v, t) \in A \land (v, t') \in R, t' > t \).

\textit{add} The \textit{add} operation inserts new values in the \textit{add-set}. It is defined as \( \text{add}(v, t) : S.A \rightarrow S.A \cup \{(v, t)\} \).

\textit{remove} The \textit{remove} operation deletes values from the set by adding them to the \textit{remove-set}. It is defined as \( \text{remove}(v, t) : S.R \rightarrow S.R \cup \{(v, t)\} \).

4.4.4 LWWMap

The LWWMap is defined as an extension of the LWWSet. The data is stored as a key/value pair \((k, v, t)\).

\textit{lookup, query} The \textit{lookup} operation is modified so that it looks up the key instead of the value. That is, \( \text{lookup}(k) \equiv \text{true} \) if \( \exists k, t \ni (k, v, t) \in A \land (k, v, t') \in R, t' > t \). The \textit{query}(\( k \)) method returns the value for a key \( k \).

\textit{add, remove} Analogously to the \textit{add} and \textit{remove} of the LWWSet: \( \text{add}(k, v, t) : S.A \rightarrow S.A \cup \{(k, v, t)\} \); and \( \text{remove}(k, v, t) : S.R \rightarrow S.R \cup \{(k, v, t)\} \).

\textit{update} This operation is the only substantial difference the LWWMap introduces to the LWWSet. Updating a key \((k, v, t)\) entry with a value \( v' \) timestamped with \( t' \) is defined as \( \text{update}(k, v, v', t, t') = \text{add}(k, v', t') \circ \text{remove}(k, v, t - \epsilon) \). First, the entry with key \( k \) is removed, and the time of removal is timestamped with a value that is older than the new timestamp \( t' \) by the minimal time interval \( \epsilon \) the system is able to detect. In lowkey \( \epsilon = 1 \text{ ns} \). Subsequently, the entry with key \( k \) and the new value \( v' \) is added with the timestamp \( t' \).

4.4.5 LWWGraph

The LWWGraph \( G \) is the extension of the LWWMap that enables persisting attributes of the graph. This mechanism is used for storing the vertices and edges of the graph. Vertices are stored in an LWWSet, denoted by \( V \). Each element of \( V \) is an LWWVertex. To enable storing attributes and metadata of vertices, each LWWVertex is an extension of the LWWMap. Analogously, the edge set \( E \) is an LWWSet, containing LWWEdge instances. Formally: \( G = (V, E) \), where

- \( V, E \vdash \text{LWWSet} \);
- \( V = \{(v, t)\}, v \vdash \text{LWWVertex} \) and \( t \) is a timestamp;
- \( E = \{(e, t)\}, e \vdash \text{LWWEdge} \) and \( t \) is a timestamp;
- \( \forall e \in E : e \text{.query("source")}, e \text{.query("target")} \in V \) (denoted \( e \text{.source} \) and \( e \text{.target} \);
- \( \text{LWWGraph, LWWVertex, LWWEdge} \vdash \text{LWWMap} \).

Due to the invariant property of \( E \subseteq V \times V \), operations on \( V \) and \( E \) are not independent. Shapiro et al. [51] suggest multiple ways to manage this issue. In lowkey, we chose prioritizing the \textit{removeVertex} operation to ensure CRDT behavior. That is, the operation is only allowed to be executed if it does not leave a dangling edge behind.

\textit{lookup and query} Since both the LWWVertex and the LWWEdge extend the LWWMap, their lookup and query methods are identical to the ones discussed in Sect. 4.4.4.

\textit{addEdge, addVertex} These operations reuse the API of the LWWSet directly. Adding an edge is achieved by adding the edge to the add-set \( A \) of the edge set \( E \) of graph \( G \).
Thus, \(\text{addEdge}(e, t) : G.E.A \rightarrow G.E.A \cup \{(e, t)\}\); and \(\text{addVertex}(v, t) : G.V.A \rightarrow G.V.A \cup \{(v, t)\}\). \(\text{removeEdge}, \text{removeVertex}\) These operations reuse the API of the \(\text{LWWSet}\) directly. Removing an edge is achieved by adding the edge to the remove-set \(R\) of the edge set \(E\) of graph \(G\). Thus, \(\text{removeEdge}(e, t) : G.E.R \rightarrow G.E.R \cup \{(e, t)\}\); and \(\text{removeVertex}(v, t) : G.V.R \rightarrow G.V.R \cup \{(v, t)\}\). Furthermore, to ensure the CRDT behavior: \(\text{removeVertex}(v, t) \Rightarrow \exists e \in E : e\.source=v \lor e\.target=v\).

Additional methods of the \(\text{LWWGraph}\) are defined for querying various properties of the graph; and for adding and removing vertices and edges by name and identifier. Additional methods of the \(\text{LWWEdge}\) and \(\text{LWWVertex}\) types are defined, e.g., for querying incoming and outgoing edges of a vertex, cascading the removal of dangling edges upon a vertex removal, and assigning direction to edges.

### 4.5 Network architecture

**lowkey** follows a client-server network architecture with multiple clients connecting to the same server, as outlined already in Sect. 4.1. The network architecture is built on top of the ZeroMQ\(^8\) asynchronous messaging library. It offers efficient scalability and latency properties; thus it aligns well with the requirements of real-time collaboration.

#### 4.5.1 Server and client components

As shown in Fig. 3, our framework provides two network components for connecting remote collaborating stakeholders: the server and the client.

The server component is responsible for two tasks: (i) collecting from, and distributing updates among clients; and (ii) providing newly joined clients with the snapshot of the system so they have an initial local replica.

The client component is responsible for providing networking capabilities to modeling tools and editors. The **Client interface** is accessible from the API of the framework and it is the tool builder’s responsibility to properly implement its required functionality. Specifically, the **Client interface** is properly implemented by defining the action the **Sub** socket executes periodically during its polling loop.

#### 4.5.2 Communication patterns

Different responsibilities of the server are implemented with different communication patterns, as summarized in Fig. 3. **Exchanging updates** To receive updates, a client must first **subscribe** to the updates by connecting to the **Pub** (publisher) socket of the server with its own **Sub** (subscriber) socket. The Pub-Sub pattern establishes a one-way asynchronous communication channel with the client receiving and processing messages in a polling loop. The **Pub** socket broadcasts messages to every client connected to the server. To send updates, a client must first connect to the **Pull** socket of the server with its **Push** socket. Similar to the Pub-Sub pattern, the Pull-Push pattern establishes a one-way connection. Note that the **Push** socket is geared towards supporting pipelining mechanisms, hence it does not broadcast messages.

**Distributing system snapshots** Upon connecting to the server, the client needs to obtain a snapshot of the system. This is achieved by connecting to the **Router** socket of the server via the **Dealer** socket of the client. The Router-Dealer pattern implements a request-reply mechanism with both ends acting asynchronously. As opposed to the patterns used for exchanging updates, this pattern does not maintain a connection after the request-reply pair of messages has been exchanged.

To provide a snapshot to its clients, the server is equipped with a memory that stores the history of previously exchanged messages (model updates). Once the newly connected client requests the snapshot, the history of messages is replayed to it. The state of the system is then built up locally by the client. We opted for this mechanism to keep it aligned with our choice of operation-based CRDT semantics (Sect. 3.3), and to shift the workload to the client instead of the server. Since operation-based and state-based CRDT are able to emulate each other, it would be possible to communicate the whole state at once, but this would come at the price of increased network traffic.

#### 4.5.3 Update messages

Every update message has the following signature:

\(\langle \text{clientId}, \text{command}, \text{timestamp} \rangle\), where the **clientId** is a UUID assigned to the client upon its creation, used for avoiding double delivery problems; **command** corresponds to the command language of the physical metamodel shown in Listing 1; and **timestamp** is the timestamp of creation, as discussed in Sect. 4.4.1. Messages are serialized as text and sent through the TCP/IP stack.

Upon reception, the command encapsulated in the update message is executed on the local replica. In the current implementation of lowkey, the scope of the change is the root model in the modeling session. Advanced modeling scenarios might necessitate more complex scope management. For example, multi-view modeling requires support for namespaces in order to allow independent views to define syntactically similar or identical concepts [14]. Such features are left for future work.

---

\(^8\) https://zeromq.org/.
5 Feasibility evaluation

Based on our illustrative case, we evaluate the feasibility of lowkey from two points of view. In both cases, our objective is to assess whether lowkey can accommodate the challenges outlined in the case study. First, in Sect. 5.1, we provide a language engineer’s view by presenting the process of metamodeling in lowkey. Then, in Sect. 5.2, we provide a domain expert’s view by demonstrating four collaborative modeling scenarios.

Study setup We conduct the evaluation on three desktop machines with Python 3.7 and the required libraries installed on them. We deploy the lowkey framework as an editable local package. First, in Sect. 5.1, we model the domain concepts of the Mind map editor on one machine and generate a domain-specific API the other two machines will use during collaboration. Second, in Sect. 5.2, we deploy this domain-specific API to two machines along with the Mind map editor and conduct our experiments guided by the requirements formulated in Sect. 2.

5.1 Language engineer’s view: metamodeling in lowkey

Figure 5 shows how the metamodel of the illustrative case (Fig. 1) is defined within the lowkey framework in relation to the physical model (Fig. 4). The linguistic metamodel represents the same domain as the metamodel in Fig. 1, but instead of using UML to express it, here, we use the physical metamodel of the lowkey framework.9 The color coding shows how every concept physically conforms to the Clabject. This is due to the linguistic metamodel being an instance of the physical metamodel. In addition, the linguistic metamodel linguistically conforms to the linguistic meta-metamodel. The latter is another instance of the physical metamodel; thus, its elements physically conform to the specific elements of the physical metamodel (shown by color-coding). Specific Mind map instances are created in the linguistic instance model that linguistically conforms to the linguistic metamodel, and physically conforms to the physical metamodel. As emphasized by the arrow notation, the mindmap_0 object is a linguistic instance of the MindMap class; which is, in turn, a linguistic instance of the Class class. Each of these are physical instances of the Clabject class. Similarly, the title of the mindmap “todolist” is a linguistic instance of title: String = “”; which is, in turn, a linguistic instance of the Attribute class.

\[\text{We remark, that multi-level modeling approaches traditionally rely on the orthogonal linguistic and ontological dimensions, and do not consider the physical dimension. In this paper, we only consider the linguistic and physical dimensions to allow an easier discussion. Our approach can be safely extended with ontological aspects, similarly to the work of [61].}\]

5.2 Domain expert’s view: collaboration in lowkey

After the linguistic metamodel has been defined, the domain experts can build the linguistic instance model in a collaborative fashion. Figure 6 illustrates this collaboration by showing the objects of the linguistic metamodel and the CRDT at the local replicas. For space considerations, we omit the technical details, such as clients connecting to the server; and only show the CRDT for Client A. For testing, evaluation, and demonstration purposes, we implemented a simple Mind map editor via the command line. The commands of the editor are shown in Listing 3. In the following, we outline typical collaboration scenarios based on our experiments with the editor.

1. Cooperation

\[\text{Thanks to the clear physical conformance relationships, every element at the physical instance level will be persisted as a CRDT, irrespective of their linguistic meta-level. Therefore, stakeholders can safely collaborate by editing any of the three linguistic metamodels.}\]

Generating a domain-specific API To enable interacting with the linguistic metamodel, an API has to be produced. This process is fully automated with a template-based code generator that produces a Python class for every Clabject, and generates the appropriate accessors (e.g., get, set, add, remove) for attributes and references. Listing 2 shows the method signatures of the Python code generated for the MindMap class.10

```
1 class MindMap(Clabject):
2     # Attribute: title, Type: String,
3     # Multiplicity: 1
4     def getTitle(self)
5     def setTitle(self, title)
6         # Reference: topic, Type: CentralTopic
7         # MultiplicityFrom: 1..1, MultiplicityTo: 1..1
8         # IsComposition: True
9         def getTopic(self)
10         def setTopic(self, topic: CentralTopic)
11         def removeTopic(self)
12
13         # Reference: markers, Type: Marker
14         # MultiplicityFrom: 0..1, MultiplicityTo: 0..*1
15         # IsComposition: True
16         def getMarkers(self)
17         def addMarker(self, marker)
18         def removeMarker(self, self, marker)
19
```

Listing 2 Excerpt of the generated model API.

\[\text{The full example is available at https://github.com/geodes-sms/lowkey.}\]
Real-time collaborative multi-level modeling by conflict-free replicated data types

Fig. 5 The three-level hierarchy of linguistic models corresponding to the physical metamodel in the mindmap case (color figure online)

Listing 3 MindMap DSL of the Editor

1 READ -- Returns the mindmap model in a readable form
2 OBJECTS -- Lists every object in the local session
3 CREATE (type) (name) -- Creates an instance with name of the domain-specific type
4 LINK (source).(port) TO (target) -- Links object target to object source via port
5 UPDATE (name) (attribute) (newValue) -- Updates attribute with name to newValue
6 DELETE (name) -- Deletes object name

– Client A creates a MindMap instance by issuing the CREATE MindMap mindmap_0 command. As shown in Fig. 3, the command is parsed into Python source code that uses the Domain-specific API (Listing 2) of the linguistic level. The role of the linguistic level is to enforce linguistic well-formedness rules. Since the model with the single mindmap_0 object is well-formed, the command is further translated to the command language of the physical modeling layer (Listing 1).
– The client sends this command to the Server, using their respective Push-Pull sockets. The message is stored in the list of updates at the Server.
– At the CRDT level, an LWWVertex is instantiated to store the local data at the client side. The key-value pairs of the LWWVertex store every information required to reconstruct the linguistic object, i.e., (type, MindMap); (name, mindmap_0); and (title, mindmap_0). In the current implementation of lowkey, an LWWGraph is automatically instantiated to accommodate the LWWVertex and LWWEdge instances created throughout the collaboration.
– The Server broadcasts the change to the collaborators, i.e., Client B. Subsequently, an object from the MindMap class is instantiated with the same ID as the one at User A’s side.
– Client B changes the title attribute of the mindmap_0 object by issuing the UPDATE command. The change is applied locally and forwarded to the Server after the required conversion to the command language. Client
Fig. 6  Collaboration in lowkey outlining changes in the CRDT shown for Client A
A receives the change from the Server and updates its local replica.

- At the CRDT level, the \texttt{LWWVertex} is updated by changing the \textit{title} from \texttt{mindmap\_0} to \texttt{todolist}, reflecting the changes by Client B. As explained in Sect. 4.4, the previous and current values are stored in the \textit{Add-set} of the \texttt{LWWVertex} (inherited from the \texttt{LWWMap}). Querying the CRDT instance yields \texttt{todolist} as the value with the latest timestamp.

2. Cooperation with linguistic inconsistencies

- Client A creates a CentralTopic instance by the name \textit{tasks}. The \texttt{centraltopic\_0} object is not linked to the \texttt{mindmap\_0} object, violating the 1-1 relationship between the MindMap and CentralTopic classes, prescribed by the static semantics of the metamodel (Fig. 1). In this example, we assume a less restrictive linguistic modeling layer, in which static semantics are not enforced. The \textit{physical modeling layer}, however, enforces physical well-formedness rules, e.g., the model has to correspond to a graph, which is allowed to be disconnected. (This rule is defined by the aggregation between the \textit{Model} and \textit{Node} types in Fig. 4.) The editor-level user operation, thus, is propagated to the \textit{physical modeling layer}, translated into the command language: CREATE -typedBy CentralTopic -name \textit{tasks}. Because of the valid physical model, the message is forwarded to the server and distributed to Client B.

- At the CRDT level, a new \texttt{LWWVertex} is created to store the CentralTopic instance.

- Client B links the CentralTopic instance to the MindMap instance. The distribution of the tasks of object creation (Step 5) and linking (Step 6) creates a truly collaborative setting, demonstrating cases in which stakeholders of specialized expertise take care of partial tasks, not necessarily resulting in a valid linguistic model on every atomic operation. Client B creates an association of type \textit{topic} between the two objects, named \texttt{topic\_0}. The operation results in a valid linguistic model, the changes are propagated in the usual way, and eventually, the change is reflected in the replica of Client A.

- At the CRDT level, a new \texttt{LWWEdge} is created to store the association. The \texttt{LWWEdge} is added to the \texttt{LWWGraph} with the previous two \texttt{LWWVertex} as its source and destination.

3. Conflict management

- Client A works on objects that Client B has deleted. First, Client B deletes the \texttt{centraltopic\_0} object, and as a consequence, the object is marked deleted in the local replica. As explained in Sect. 4.4, CRDT use soft delete to be able to restore data if needed. The update message is sent to the server. In the meantime, Client A creates an instance of the \textit{MainTopic} class and links it to the \texttt{centraltopic\_0} object via the \textit{mainTopics\_0} reference. The two update messages are propagated to the server. Client A started its changes before the updates about Client B’s changes have reached him. Subsequently, both clients’ updates are sent to the other client, causing inconsistency in the local replicas. However, both inconsistencies are resolved immediately by the LWW semantics of CRDT. Since Client A’s updates have been created later, the timestamping mechanism will resolve the inconsistencies by (i) leaving Client A’s replica intact and (ii) re-adding the \texttt{centraltopic\_0} object to Client B’s replica.

- At the CRDT level, the local replica of Client B removes the \texttt{LWWVertex} storing the CentralTopic instance: the vertex is removed from the vertex set of the \texttt{LWWGraph}. This is achieved by adding the vertex to the \textit{Remove-set} of the graph, as explained in Sect. 4.4.4. Before this information is propagated, Client A performs editing operations resulting in a new \texttt{LWWVertex} and a new \texttt{LWWEdge}. The source of the newly created \texttt{LWWEdge} is the \texttt{LWWVertex} removed by Client B, hence the conflict. However, due to the more recently created (and timestamped) \texttt{LWWEdge}, the \texttt{LWWVertex} is not removed in the local replica of Client A. After Client B is notified about the operations of Client A, the \texttt{LWWVertex} storing the CentralTopic instance is re-added to its local replica, restoring the consistency between the clients.

4. Multi-level cooperation

- Users A and B have created multiple instances of the \textit{Marker} class and would like to categorize them into textual and graphical markers. One way to achieve this is by allowing a templating mechanism to these users so that they can create specific sub-classes of the \textit{Marker} class. These classes can be then instantiated with a specific \textit{symbol}.

- The Language designer increases the \textit{potency} of the \textit{Marker} class from 1 to 2. The change is distributed to the users.

- User A creates a new instance of the \textit{Marker} class by the name TextMarker. The \textit{potency} of the newly created clabject is set to 1, allowing one more instantiation. Subsequently, he modifies the previous \textit{marker\_0} in a way that it is typed by the newly created TextMarker class. The \textit{potency} of \textit{marker\_0} is set to 0, preventing any further instantiation.
5. Multi-level integrity

- After defining the `TextMarker` type and creating the `marker_0` instance of it, the language designer decides to revoke this level of flexibility from the users and sets the potency of the `Marker` class to 1. The change is distributed to the users.

- The potency of the `TextMarker` instance is now automatically set to 0 by `lowkey`. However, this compromises the integrity of the instantiation chain as the 0 potency of the `TextMarker` prohibits the instantiation of `marker_0`. `lowkey` resolves this problem by prioritizing the integrity of the instantiation chain and marking `marker_0` as deleted. While the physical integrity of the model is ensured, linguistic well-formedness might be violated. For example, the `TextMarker` object does not define a `symbol` attribute, which might violate linguistic well-formedness rules. As explained in Sect. 4.2, such rules are defined by language engineers and tool builders at the linguistic modeling layer.

- Should the Language Designer set the potency of the `Marker` class to 2 again, `marker_0` instance becomes available to the Domain experts again.

5.3 Summary

In this feasibility evaluation, we have demonstrated the usage of `lowkey` in advanced modeling scenarios. Concerning the requirements formulated in Sect. 2, we have demonstrated that replicas converge (R1) and that, in a timely fashion (R2). Both causality (R3) and user intention is preserved (R4). The language engineer’s point of view (Sect. 5.1) has shown the development of domain-specific modeling languages with multi-level modeling capabilities (R5). `lowkey` enabled reusing the metamodel of the case (Fig. 1) that was previously defined in an external tool (R6). The domain expert’s point of view (Sect. 5.2) has demonstrated how real-time collaboration is achieved by CRDT, irrespectively of the level of linguistic abstraction (R5). In addition, we have highlighted how the combination of CRDT and the Physical metamodel allows for seamless collaboration in the presence of linguistic inconsistencies and non-conformance (R7).

Limitations and threats to validity Our evaluation, being a feasibility study, is limited in some aspects. First, in Sect. 5.1, we used a simple editor developed for our purposes. This setup does not allow for assessing the ease of integration with more complex legacy editors. The main problems that might emerge while integrating with an external editor are primarily associated with architectural choices and the lack of technological convergence. These limitations pose threat to the external validity of our feasibility evaluation. However,
these problems are largely mitigated by the relatively loose restrictions the 0MQ-based architecture of lowkey imposes.

Second, in Sect. 5.1, we used only three collaborators, while this number might be larger in realistic settings. User intention [55] becomes harder to ensure as the number of collaborators increases. The lower number of collaborators in our evaluation might threaten conclusion validity. However, these threats are mainly restricted to usability edge cases and do not affect the structural correctness of collaboratively developed models.

Finally, the relatively short example scenarios do not allow observing scalability issues of CRDT caused by the continuously growing number of accumulated messages. However, scalability concerns have not been the focus of this evaluation and we consider the topic future work.

6 Discussion

In this section, we assess how lowkey compares to other modeling and collaborative frameworks. Then, we reflect on various aspects of the approach.

6.1 Comparison

We compare lowkey with frameworks that are closest in their aim and feature set, shown in Table 1. These are typically either modeling tools with multi-level modeling capabilities or tools with real-time collaborative features. Our objective is to assess how lowkey compares in terms of the key functionality of (i) metamodeling, (ii) multi-level modeling, and (iii) real-time collaboration. We find that some tools partially overlap with lowkey in terms of functionality; however, the combination of the three key features is unique to lowkey.

Metamodeling, i.e., the ability to construct new metamodels and modeling languages, is supported by the majority of the sampled tools. MetaEdit+ [35] is a widely adopted metamodeling framework. Locking at the class level (not attributes) is the primary collaborative mechanism, but there is no support for real-time collaboration. MetaEdit+ represents a class of modeling tools that gained substantial industrial adoption, and could benefit from a real-time collaborative framework such as lowkey. The two exceptions are TGRL [48] and the MONDO framework [24]. TGRL is a tool for requirements modeling in a collaborative way. MONDO provides collaborative mechanisms for domain-specific modeling.

Multi-level modeling, i.e., the ability to define metamodels at an arbitrary number of meta-levels, is supported by the mechanisms of deep characterization [6] and deep instantiation [5]. AtomoPM [58] is a web-based multi-view modeling tool that allows for defining metamodels at arbitrary number of levels and instantiating them by bootstrapping mechanisms. However, deep characterization and deep instantiation are not supported. The Modelverse [62] is a modelware backend for storing and simulating models. It achieves multi-level modeling by using a physical metamodel similar to lowkey [61]. The same approach has been used by deep metamodeling frameworks MetaDepth [20] and Melanee [2]. Similar to lowkey, these tools use graphs at the meta-circular level, i.e., the topmost linguistic meta-level.

Real-time collaboration is becoming increasingly adopted in model editors [17]. Tools such as WebGME [39], SpacEclipse [29], FlexiSketch [67], and to some extent SyncMeta [25] support metamodeling by shallow instantiation, augmented with real-time collaboration capabilities. As a consequence of shallow instantiation, these tools are restricted to a three-level meta-hierarchy, such as OMG’s MOF or EMF [54]. A variety of consistency models are employed in these tools to support collaboration. Strong consistency, employed in AtomoPM, SpacEclipse, and FlexiSketch, ensures that all participating nodes hold the exact same state of the model at all times. However, due to its underlying mechanisms, it significantly hinders the scalability and user experience of collaborative modeling tools [37]. In AtomoPM, real-time collaboration is supported in two ways: at the levels of the abstract and concrete syntax. In both cases, changes in the abstract syntax are shared; in the latter case, changes in the representation are shared as well. Collaboration in SpacEclipse and AtomoPM requires manual conflict resolution, while FlexiSketch uses preventive conflict management techniques. WebGME relies on eventual consistency, that provides the weaker guarantee that changes will be eventually observed across each node [65]. However, conflict resolution is not automated. Novel types of real-time collaborative tools, such as TGRL and SyncMeta employ strong eventual consistency (SEC) that combines the benefits of strong and eventual models [51] and avoids conflicts altogether. TGRL and SyncMeta implement real-time collaboration using the Teletype and Yjs CRDT frameworks, respectively.

Summary As shown in Table 1, lowkey provides a unique combination of features for real-time collaborative multi-level modeling. Typically, modeling frameworks either provide multi-level modeling capabilities without support for real-time collaboration (e.g., Modelverse, MetaDepth); or provide real-time collaboration capabilities without support for multi-level modeling (e.g., WebGME, FlexiSketch). Closest to our work is AtomoPM, which provides limited facilities for multi-level modeling, and supports real-time collaboration by conservative consistency model and without automated conflict resolution.

6.2 Physical metamodel

One of the main benefits of the physical metamodel is the uniform representation of objects and models, irrespective
of the linguistic meta-level they are situated at. This mechanism allows for the co-existence of models with different syntaxes and semantics. As a consequence, the collaboration between different modeling tools becomes a more manageable endeavor.

We have chosen graphs as the meta-circular level. That is, the physical metamodel corresponds to graphs, and all linguistic models correspond to graphs as well. As discussed in Sect. 6.1, graphs have been shown to be an appropriate and versatile choice for such purposes. We have implemented a directed multigraph formalism, i.e., edges have an unambiguous source and target vertex, and multiple edges are allowed between vertices. Directed edges enable navigability of associations in linguistic models, and the multigraph nature enables defining multiple different associations between the same pair of classes. Additionally, the physical metamodel supports disconnected graphs. We found this property useful in enabling the temporal tolerance of linguistic inconsistencies. Additional graph properties can be implemented and enforced, depending on the use-cases to be supported by the framework.

Some scenarios might necessitate extending the physical metamodel with custom-built types. The feasibility of such techniques has been discussed in previous work [19].

### 6.3 Temporal tolerance of linguistic inconsistencies

The separated physical metamodel allows for collaboration in the presence of linguistic non-conformance (vertical inconsistencies), enabling advanced collaboration scenarios. For example, ensuring the consistency of a model during the collaboration of stakeholders with different expertise might require tolerating linguistic non-conformance. This has been demonstrated in Scenario 2 of Fig. 6. Here, the expertise of Client A is the instantiation of objects and the expertise of Client B is organizing dangling objects into models. Already in this simple example, we were able to introduce a non-conformance between the instance model and the metamodel when Client A did not link the newly created instance to the root object immediately. In practical applications, this issue is vastly exacerbated, as the number of stakeholders, domains, formalisms, and tools increases. The CRDT persisting the physical metamodel ensure horizontal consistency [64], i.e., stakeholders have a consistent view of the system, even if these views are linguistically incorrect. This enables a smooth collaboration in the presence of linguistic non-conformance.

Tool builders can use lowkey for building safer applications with improved inconsistency management mechanisms. As demonstrated, temporal inconsistencies between modeling operations are naturally occurring phenomena in modeling scenarios. The separation of physical and linguistic concerns enables well-formedness and consistency rules to be captured at the most appropriate levels of abstraction. For example, one might constrain their models at the physical level by enforcing conformance to graphs and at the linguistic level to class diagrams. Additionally, the domain-specific API generation capabilities of lowkey-as demonstrated in Sect. 5.1-further improve the safety of modeling tools by allowing for a rich intermediate representation. Finally, tool integrators might want to consider lowkey as the mediating layer in their tool chains. On the one hand, lowkey is agnostic of the internal representations of the tools to be integrated, allowing to accommodate, in principle, any modeling formalism. On the other hand, the multi-level metamodeling architecture allows for ensuring consistency rules at arbitrary levels of abstraction. The architecture of lowkey makes it especially suitable for developing complex applications and tool chains. Tools to be integrated simply need to establish communication with the lowkey Server through the 0MQ messaging broker used in our architecture. The complexity of this step is substantially mitigated by 0MQ’s support for a wide array of programming languages.

### 6.4 Support for complex modeling operations

Throughout this paper, we have assumed atomic change operations (e.g., CRUD operations in the running example). In practical modeling scenarios, more complex domain-specific operations are often required to be supported, e.g., in the automation of large refactoring on the model(s). In the running example, a promote operation could be defined to move a subtopic with all its children directly under the central topic, converting the subtopic into a maintopic automatically. The need for such strategies in multi-level modeling has been thoroughly discussed by [21]. lowkey provides the necessary mechanisms for implementing complex operations in real-time collaborative multi-level modeling settings. While complex modeling operations are defined at the level of the editor, lowkey allows for treating these operations as first-class citizens at the CRDT API level, improving the safety of the software application.

Depending on (i) the synchronicity of message transmission between collaborators, and (ii) the timestamping strategy, there are at least three distinct ways to treat complex operations in a CRDT-based collaborative setting.

First, atomic operations can be aggregated at the client side and transmitted all-at-once as soon as the complex operation is composed, with each atomic operation receiving the same timestamp. This strategy is a natural choice in modeling editors in which atomic operations of a complex functionality are conceptually executed at the same time, with the complex operation being the unit of change. In other words, complex operations are treated as transactions. The benefit of this strategy is that the aggregation logic of atomic operations reflects the intentions of the originator: it is the originator
who decides which operations to compose into a complex operation. Second, similarly to the previous alternative, atomic operations can be aggregated at the client side and transmitted all-at-once, however, with each atomic operation receiving its own timestamp. Conceptually, the transmission of atomic operations is delayed until a complex operation is built up at the client side. This allows for beneficial mechanisms at the originator’s side, such as local undo-redo and rollback. At the receiving end, atomic operations—each with its own timestamp—have to be merged into the local replica. However, the notion of aggregation is lost, unless the clients have previously agreed on the same aggregation semantics.

Finally, atomic operations can be transmitted immediately and aggregated at the receiving clients’ side. Clearly, this is the opposite of the first alternative where the originator controls the aggregation. Here, it is the receiving clients who decide what to aggregate and how. This strategy is often used in stream processing and complex event processing [16], where clients might be interested in their own, domain-specific complex phenomena. For example, a safety engineer who is part of a collaboration might want to execute safety checks after specific complex sequences of changes, but that complex sequence might not interest others.

6.5 Opportunities in multi-view modeling

We anticipate multi-view modeling (MVM) [46] being one of the main application areas of our approach. Views are projections of one or multiple underlying models [14], presenting stakeholders with only the essential information they require for their work. Views typically pertain to domains and expertise (e.g., the mechanical and the electrical views in the design of a mechatronic system), but they can pertain to specific use cases (e.g., electro-mechanic view) or expertise (e.g., chassis design). The architecture and services of lowkey align well with the requirements of MVM. It is able to accommodate multiple different metamodels, their instances, and their views in a uniform fashion, thus allowing for change propagation between linguistically and semantically different views. Model- and screen sharing [63] are straightforward to implement, as tool builders can outsource the data layer of their tools to lowkey.

Other approaches relying on an ensemble of multiple models—such as multi-paradigm modeling (MPM) [43], multi-modeling [9] and blended modeling [18]—can benefit from this approach as well. MPM advocates modeling every aspect of the system at the most appropriate level(s) of abstraction using the most appropriate formalism(s).

The physical metamodel provides a basis for synchronization among stakeholders, while different formalisms can be implemented at the linguistic level. Similar techniques have been employed in the MPM tool Modelverse [62].

6.6 Accommodating mainstream modeling frameworks

As demonstrated by [5], multi-level modeling frameworks can emulate traditional modeling frameworks using the notion of potency. Potency is a constraint that specifies how many times a class can be instantiated transitively. As shown in Fig. 4, the physical metamodel of lowkey defines a potency to its CJob element. By that, lowkey subsumes traditional modeling frameworks that operate on the mechanism of shallow instantiation. We see an opportunity in developing libraries for lowkey implementing the meta-facilities of these traditional frameworks. The process of adopting lowkey in already existing modeling tools relying on such frameworks, e.g., MOF or EMF, can be vastly improved and automated.

6.7 Limitations

Performance The performance of CRDT is subject to the number of objects present in the specific application [56]. lowkey CRDT implement a soft delete mechanism, i.e., objects are never removed from the specific CRDT, but rather, “marked” as removed. The LWWSet, for example, contains the removed elements in its remove-set (see Sect. 4.4). As a consequence, the performance of the CRDT layer will gradually decrease as the number of objects increases. Our preliminary measurements show linear degradation. Garbage collection mechanisms have been suggested for managing such limitations of CRDT-based applications [8].

Intention preservation [55] plays a key role in achieving an intuitive human–computer interaction and a smooth user experience in collaborative settings. While the LWW paradigm usually preserves the user’s intention, some corner cases might result in model changes that are less intuitive. The user makes decisions on changing the model based on the model’s materialized view in the modeling tool or browser. However, there might be change updates from other collaborators on their way that might arrive after the user carried out his changes. These updates retroactively change the model in a way that affects the user’s reasoning. Since there is no way to account for such messages, the user’s intention in such corner cases cannot be guaranteed. While manual conflict resolution would solve this issue, it would also render real-time collaboration infeasible.

Timestamping mechanism The current prototype implementation of lowkey uses the time.time_ns() Python function for timestamping changes. This function returns time as an integer number of nanoseconds. However, this approach is prone to clock drift, which could render CRDT inconsistent. Our working assumption is that nanosecond-level drift does not affect the dynamics of collaborative modeling between humans, as the interactions in such settings are at the level of seconds. Nonetheless, lowkey can be extended by
additional mechanisms to regularly synchronize clocks using the Network Time Protocol (NTP)\(^{11}\) and other mechanisms described in Sect. 4.4.1.

7 Conclusion

In this paper, we have presented a real-time collaborative framework for a wide range of advanced modeling scenarios, supported by techniques of multi-level modeling. Our framework provides a unique combination of modeling capabilities and real-time collaboration. It is built on a custom implementation of CRDT, geared towards graph models, providing promising real-time capabilities and scalability. We have defined a mapping of physical metamodels onto CRDT and demonstrated the approach through an illustrative case.

We have identified multiple lines of future work. We are planning to develop state-of-the-art collaborative multi-view modeling mechanisms based on the framework. To enable easier integration with legacy models, we will provide profiles for UML and EMF models, effectively reproducing the respective meta-levels of the OMG superstructure and Ecore. We plan to build a family of modeling editors integrated with existing modeling frameworks to augment them with real-time collaborative capabilities. In more technical terms, we are looking to improve the performance of the framework by implementing advanced garbage collection mechanisms and a network stack with less overhead. We will develop a benchmark for collaborative MDSE frameworks to evaluate their scalability (with respect to connected clients and messages exchanged), performance (response time of a local operation being propagated to each remote client), and usability in typical modeling scenarios.

Acknowledgements The authors would like to express their gratitude for the insightful comments of the reviewers that helped improve the initial manuscript.

References

1. Álvarez, J.M., Evans, A., Sammut, P.: Mapping between levels in the metamodel architecture. In: «UML» 2001—The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, October 1–5, 2001, Proceedings, Springer, LNCS, vol. 2185, pp. 34–46 (2001)
2. Atkinson, C., Gerbig, R.: Flexible deep modeling with Melanee. In: Modellierung, GI, LNI, vol. 255, pp. 117–122 (2016)
3. Atkinson, C., Kühne, T.: Meta-level independent modelling. In: European Conference on Object-Oriented Programming, vol. 12, p. 16 (2000)
4. Atkinson, C., Kühne, T.: The essence of multilevel metamodeling. In: The Unified Modeling Language, Modeling Languages, Concepts, and Tools, LNCS, vol. 2185, pp. 19–33. Springer, Berlin (2001)
5. Atkinson, C., Kühne, T.: Rearchitecting the UML infrastructure. ACM Trans. Model. Comput. Simul. 12(4), 290–321 (2002)
6. Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. Softw. Syst. Model. 7(3), 345–359 (2008)
7. Atkinson, C., Gutheil, M., Kennel, B.: A flexible infrastructure for multilevel language engineering. IEEE Trans. Softw. Eng. 35(6), 742–755 (2009)
8. Bauwens, J., Boix, E.G.: Memory efficient CRDTs in dynamic environments. In: International Workshop on Virtual Machines and Intermediate Languages, pp. 48–57. ACM (2019)
9. Boronat, A., Knapp, A., Meseguer, J., Wirsing, M.: What is a multi-modeling language? In: Recent Trends in Algebraic Development Techniques, 19th International Workshop, LNCS, vol. 5486, pp. 71–87. Springer, Berlin (2008)
10. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice, Second Edition. Morgan & Claypool Publishers (2017)
11. Brasilheiro, F., Almeida, J.P.A., de Carvalho, V.A., Guizzardi, G.: Applying a multi-level modeling theory to assess taxonomic hierarchies in wikidata. In: Bourdeau, J., Hendler, J., Nkambou, R., Horrocks, I., Zhao, B.Y. (eds.) Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, 2016, Companion Volume, ACM, pp. 975–980 (2016)
12. Brewer, E.: CAP twelve years later: How the “rules” have changed. Computer 45(2), 23–29 (2012)
13. Buzan, T.: The ultimate book of mind maps: unlock your creativity, boost your memory, change your life. HarperCollins UK (2006)
14. Corley, J., Syriani, E., Ergin, H., Van Mierlo, S.: Modern Software Engineering Methodologies for Mobile and Cloud Environments. IGI Global, chap Cloud-based Multi-View Modeling Environments, vol. 7, pp. 120–139 (2016)
15. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order, 2nd edn. Cambridge University Press, Cambridge (2002)
16. David, I., Ráth, I., Varró, D.: Foundations for streaming model transformations by complex event processing. Softw. Syst. Model. 17(1), 135–162 (2018)
17. David, I., Aslam, K., Faridmoyar, S., Malavolta, I., Syriani, E., Lago, P.: Collaborative model-driven software engineering: a systematic update. In: Model Driven Engineering Languages and Systems, pp. 273–284. ACM (2021)
18. David, I., Latifaj, M., Pietron, J., Zhang, W., Ciccozzi, F., Malavolta, I., Raschke, A., Steghöffer, J.P., Hebig, R.: Blended modeling in commercial and open-source model-driven software engineering tools: a systematic study. Softw. Syst. Model. To appear (2022a)
19. David, I., Syriani, E., Masson, C.: Extensible conflict-free replicated datatypes for real-time collaborative software engineering. In: Proceedings of the 17th Conference on Computer Science and Intelligence Systems (FedCSIS), IEEE, Sofia, Bulgaria (2022b)
20. de Lara, J., Guerra, E.: Deep Meta-modelling with MetaDepth. In: Objects, Models, Components, Patterns, LNCS, vol. 6141, pp. 1–20. Springer, Berlin (2010)
21. de Lara, J., Guerra, E.: Refactoring multi-level models. ACM Trans. Softw. Eng. Methodol. 27(4), 17:1–17:56 (2018)
22. de Lara, J., Guerra, E., Cuadrado, J.S.: When and how to use multilevel modelling. ACM Trans. Softw. Eng. Methodol. 24(2), 12:1–12:46 (2014)
23. De Perre, K., Myter, F., Troyer, C.D., Scholliers, C., Meuter, W.D., Boix, E.G.: Putting order in strong eventual consistency. In: International Federated Conference on Distributed Computing Techniques. Lecture Notes in Computer Science, pp. 36–56. Springer, Berlin (2019)
24. Debrenci, C., Bergmann, G., Bür, M., Ráth, I., Varró, D.: The MONDO collaboration framework: secure collaborative modeling

\(^{11}\) http://www.ntp.org/.
over existing version control systems. In: Foundations of Software Engineering, pp. 984–988. ACM (2017)
25. Dernot, M., Nicolaescu, P., Erdtmann, S., Klamka, R., Jarke, M.: Near real-time collaborative conceptual modeling on the web. In: Conceptual Modeling, LNCS, vol. 9381, pp. 344–357. Springer, Berlin (2015)
26. Du, J., Iorgulescu, C., Roy, A., Zwaenepoel, W.: GentleRain: cheap and scalable causal consistency with physical clocks. In: Symposium on Cloud Computing, pp. 4:1–4:13. ACM (2014)
27. Fowler, M., Scott, K.: UML distilled—A Brief Guide to the Standard Object Modeling Language, 2nd edn. Addison-Wesley-Longman (2000)
28. Franzago, M., Ruscio, D.D., Malavolta, I., Muccini, H.: Collaborative model-driven software engineering: a classification framework and a research map. IEEE Trans. Softw. Eng. 44(12), 1146–1175 (2018)
29. Gallardo, J., Bravo, C., Redondo, M.A.: A model-driven development method for collaborative modeling tools. J. Netw. Comput. Appl. 35(3), 1086–1105 (2012)
30. Gonzalez-Perez, C., Henderson-Sellers, B.: Metamodeling for Software Engineering. Wiley, London (2008)
31. Grosch, P., Krafft, R., Wölkli, M., Bieniusa, A.: Autocouch: a JSON CRDT framework. In: Workshop on Principles and Practice of Consistency for Distributed Data, pp. 6:1–6:7. ACM (2020)
32. Herbsleb, J.D.: Global software engineering: the future of socio-technical coordination. In: International Conference on Software Engineering, pp. 188–198. IEEE (2007)
33. Izquierdo, J.L.C., Cabot, J.: Collabora: a collaborative (meta) modeling tool. PeerJ Comput. Sci. 2, e84 (2016)
34. Johnson, P.R., Thomas, R.: Maintenance of duplicate databases. RFC 677, 1–10 (1975)
35. Kelly, S.: Collaborative modelling with version control. In: Software Technologies: Applications and Foundations. LNCS, pp. 20–29. Springer, Berlin (2017)
36. Kühne, T.: Multi-dimensional multi-level modeling. Softw. Syst. Model. 21(2), 543–559 (2022)
37. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21(1), 558–565 (1978)
38. Le Noir, J., Delande, O., Exertier, D., da Silva, M.A.A., Blanc, X.: Operation based model representation: experiences on inconsistencies detection. In: Modelling Foundations and Applications—7th European Conference, LNCS, vol. 6698, pp. 85–96. Springer, Berlin (2011)
39. Maróti, M., Kecskés, T., Kereskényi, R., Broll, B., Völgyesi, P., Jurácz, L., Levedzki, T., Lédeczi, Á.: Next generation (meta)modelling: web- and cloud-based collaborative tool infrastructure. In: Multi-Paradigm Modeling, CEUR-WS, pp. 41–60 (2014)
40. Meiklejohn, C., Van Roy, P.: Lasp: a language for distributed, coordination-free programming. In: Principles and Practice of Declarative Prog., pp. 184–195. ACM (2015)
41. Michaux, J., Blanc, X., Shapiro, M., Sutra, P.: A semantically rich approach for collaborative model edition. In: Proceedings of the 2011 ACM Symposium on Applied Computing (SAC), pp. 1470–1475. ACM (2011)
42. Mistrík, I., Grundy, J., van der Hoek, A., Whitehead, J.: Collaborative software engineering: challenges and prospects. In: Collaborative Software Engineering, pp. 389–403. Springer, Berlin (2010)
43. Mosterman, P.J., Vangheluwe, H.: Computer automated multi-paradigm modeling: an introduction. Simulation 80(9), 433–450 (2004)
44. Muccini, H., Bosch, J., van der Hoek, A.: Collaborative modeling in software engineering. IEEE Softw. 35(6), 20–24 (2018)
45. Preguiça, N.M., Marquês, J.M., Shapiro, M., Letia, M.: A commutative replicated data type for cooperative editing. In: International Conference on Distributed Computing Systems, pp. 395–403. IEEE (2009)
46. Reineke, J., Tripakis, S.: Basic problems in multi-view modeling. In: Tools and Algorithms for the Construction and Analysis of Systems. LNCS, pp. 217–232. Springer, Berlin (2014)
47. Roh, H., Jeon, M., Kim, J., Lee, J.: Replicated abstract data types: building blocks for collaborative applications. J. Parallel Distrib. Comput. 71(3), 354–368 (2011)
48. Saini, R., Mussbacher, G.: Towards conflict-free collaborative modelling using VS code extensions. In: Model-Driven Engineering Languages and Systems: Companion Proceedings, pp. 35–44. ACM (2021)
49. Saito, Y., Shapiro, M.: Optimistic replication. ACM Comput. Surv. 37(1), 42–81 (2005)
50. Schmidt, D.C.: Model-driven engineering. IEEE Comput. 39(2), 25–31 (2006)
51. Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M.: A comprehensive study of convergent and commutative replicated data types. Tech. rep., Inria–Centre Paris-Rocquencourt: INRIA(2011a)
52. Shapiro, M., Preguiça, N.M., Baquero, C., Zawirski, M.: Conflict-free replicated data types. In: Stabilization, Safety, and Security of Distributed Systems—13th Int. Symposium, Lecture Notes in Computer Science, vol. 6976, pp. 386–400. Springer, Berlin (2011b)
53. Singhal, M., Kshemkalyani, A.D.: An efficient implementation of vector clocks. Inf. Process. Lett. 43(1), 47–52 (1992)
54. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: Eclipse Modeling Framework. Pearson Education (2008)
55. Sun, C., Jia, X., Zhang, Y., Yang, Y., Chen, D.: Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. ACM Trans. Comput. Hum. Interact. 5(1), 63–108 (1998)
56. Sun, D., Sun, C.: Operation context and context-based operational transformation. In: Conference on Computer Supported Cooperative Work, pp. 279–288. ACM (2006)
57. Sun, D., Sun, C., Ng, A., Cai, W.: Real differences between OT and CRDT in correctness and complexity for consistency maintenance in co-editors. Proc. ACM Hum. Comput. Interact. 4 (2020)
58. Syriani, E., Vangheluwe, H., Mannadiar, R., Hansen, C., Mierlo, SV., Ergin, H.: AToMPM: a web-based Modeling Environment. In: Model-Driven Engineering Languages and Systems, vol. 1115, pp. 21–25. CEUR-WS.org (2013)
59. Taentzer, G., Ermel, C., Langer, P., Wimmer, M.: Conflict detection for model versioning based on graph modifications. In: Graph Transformations—5th International Conference, ICGT, LNCS, vol. 6372, pp. 171–186. Springer, Berlin (2010)
60. Thomas, R.H.: A majority consensus approach to concurrency control for multiple copy databases. ACM Trans. Database Syst. (TODS) 4(2), 180–209 (1979)
61. Van Mierlo, S., Barroca, B., Vangheluwe, H., Syriani, E., Kühne, T.: Multi-level modelling in the modelverse. In: Workshop on Multi-Level Modelling, CEUR-WS, pp. 83–92 (2014)
62. Van Tendeloo, Y., Vangheluwe, H.: The modelverse: a tool for multi-paradigm modelling and simulation. In: Winter Simulation Conference, pp. 944–955. IEEE (2017)
63. Van Tendeloo, Y., Vangheluwe, H.: Unifying model- and screen sharing. In: Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 127–132. IEEE (2018)
64. Vanherpen, K., Denil, J., David, L., Meulenaere, P., Mosterman, P.J., Töngren, M., Qamar, A., Vangheluwe, H.: Ontological reasoning for consistency in the design of cyber-physical systems. In: Int. Workshop on Cyber-Physical Production Systems, pp. 1–8. IEEE (2016)
65. Vogels, W.: Eventually consistent. Commun. ACM 52(1), 40–44 (2009)
66. Whitehead, J.: Collaboration in software engineering: a roadmap. In: International Conference on Software Engineering, pp. 214–225. IEEE (2007)

67. Wüest, D., Seyff, N., Glinz, M.: FlexiSketch: a mobile sketching tool for software modeling. In: Mobile Computing, Applications, and Services, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 110, pp. 225–244. Springer, Berlin (2012)

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Istvan David is a Postdoctoral Research Fellow at the Department of Computer Science and Operations Research at the University of Montreal, Canada. He received his PhD in Computer Science from the University of Antwerp, Belgium. His research focuses on collaborative modeling, multi-paradigm modeling and model transformations. Istvan is an IVADO Postdoctoral Research Scholarship laureate on the topic of inference of simulation models in digital twins by reinforcement learning. Contact him at http://istvandavid.com.

Eugene Syriani is an Associate Professor at the Department of Computer Science and Operations Research at University of Montreal. He received his Ph.D. in Computer Science in 2011 from McGill University. His main research interests fall in software design based on the model-driven engineering approach, the engineering of domain-specific languages, model transformation and code generation, simulation-based design, collaborative modeling, and user experience. Contact him at http://www-ens.iro.umontreal.ca/~syriani.