JOT
JOURNAL OF
OBJECT TECHNOLOGY

Journal of Object Technology | **RESEARCH ARTICLE**

Hybrid Collaborative Modeling: Problem Analysis, Requirements, and Architectural Principles

Léo Olivier*, **Marcos Didonet Del Fabro***, **and Sébastien Gérard**‡
*CEA-List, Université Paris-Saclay, France
‡IRT Jules Verne, France

ABSTRACT Modeling is inherently a collaborative activity, as it provides stakeholders with a common language for structuring domain-specific knowledge. This makes support for diverse collaboration scenarios a central requirement for modeling tools; however, most existing tools are designed primarily for either real-time or asynchronous collaboration. An emerging workflow that has so far received limited attention, yet is widely valued by practitioners, is *hybrid collaborative modeling*. In this setting, stakeholders alternate between synchronous and asynchronous work, with real-time subgroups coordinating locally while overall progress is maintained asynchronously across time zones and organizations. This article lays the foundations for the architectural requirements required to support hybrid collaborative modeling. We first introduce a classification framework for collaboration architectures and use it to compare existing collaborative modeling tools, highlighting their limited support for hybrid workflows. Building on this analysis, we present a novel architectural approach: *Local-First Collaborative Modeling*. Finally, we present a reference implementation approach based on replicated data types, intended to demonstrate feasibility and to guide future implementations.

KEYWORDS Model-Based Systems Engineering, Collaborative modeling, Local-First software, Operation-based versioning, Conflict-free Replicated Data Types

1. Introduction

A recent survey of industrial modeling practitioners across diverse domains highlights two key findings (David et al. 2023). First, modeling is inherently a collaborative activity: 93% of respondents reported engaging in model-level collaboration, and 98% consider it essential to their projects. Second, practitioners express strong expectations regarding collaborative features in modeling tools. They expect support for both real-time[1] (90%) and asynchronous (95%) collaboration, and regard model merging as essential (95%). While manual conflict resolution is still widely used (65%), automated resolution is considered equally important (77%), despite practitioners having had few opportunities to use it in practice due to limited tool support.

We analyze this gap between practitioners' expectations and the actual capabilities of modeling tools as a consequence of the limitations of the software architectures used to implement these tools. From a design perspective, existing architectures exhibit structural limitations that constrain the forms of collaboration they can effectively support. From an organizational perspective, they struggle to accommodate the evolving work practices and collaboration habits of practitioners.

The dominant collaborative architecture in modeling tools is the centralized server model, with 43 out of 48 surveyed tools adopting this approach according to (Franzago et al. 2018). While a server simplifies coordination and consistency management by acting as a single source of truth, it also constitutes a single point of failure: if it becomes unavailable or corrupted, participants lose the ability to collaborate and, in the worst case, even to access the shared model. Unsurprisingly, failure recovery is identified as a key requirement by 73% of practi-

JOT reference format:
Léo Olivier, Marcos Didonet Del Fabro, and Sébastien Gérard. *Hybrid Collaborative Modeling: Problem Analysis, Requirements, and Architectural Principles*. Journal of Object Technology. Vol. 25, No. 3, 2026. http://dx.doi.org/10.5381/jot.2026.25.3.a27

[1] Throughout this article, the terms *synchronous* and *real-time* are used interchangeably.

tioners (David et al. 2023). This architectural choice carries significant organizational implications. In practice, organizations rarely host their own servers and instead rely on cloud providers (*e.g.*, Amazon Web Services). Yet the way these providers handle and store customer data is often opaque, raising concerns about sovereignty and intellectual property protection, particularly in the event of a data breach. In multi-organization collaborations, disagreements over where and by whom shared models are stored can become a significant barrier.

The growing demand for collaborative features in modeling tools should be read in the broader context of shifting user practices over the past decade (Kleppmann et al. 2019). Practitioners are increasingly mobile and switch devices depending on context. Modeling tools must therefore provide continuity across devices. A practitioner should be able to work from an office desktop, a laptop at home over Wi-Fi, or a smartphone on a train with poor or even no connectivity. Tools should support seamless synchronization and data merging. Accordingly, users also expect cross-platform collaboration (operating systems, web environments, *etc.*). Moreover, collaboration now extends beyond human users. Human-Autonomy Teaming (HAT) (McNeese et al. 2018) is becoming a reality: AI assistants are increasingly integrated into modeling and development tools, participating in analysis, refinement, and decision-making tasks (Rädler et al. 2024; Dagenais & David 2025).

Consider, for example, a global aerospace manufacturer co-developing an avionics platform with suppliers across Europe, North America, and Asia. The core systems team in Europe co-edits architecture and safety models in real-time during design reviews, while suppliers in other time zones independently update interface models, traceability links, and verification artifacts offline. Connectivity constraints (travel, secure facilities, and bandwidth limits) make continuous access to a central server unreliable. In parallel, automated analysis agents run nightly to update requirement coverage, flag constraint violations, and propose model refactorings.

When teams reconnect, all changes (human or AI) must converge to a consistent model. This scenario exemplifies *hybrid collaborative workflows*: participants alternate between synchronous and asynchronous work, with real-time subgroups coordinating locally while overall progress is maintained asynchronously across time zones and organizations. Such workflows are increasingly representative of modern engineering practice.

From the needs reported by practitioners and the evolution of engineering practices, we derive two main conclusions. First, modeling tools must support multiple collaboration modalities to reflect the diversity of real-world scenarios. Second, model-centric development poses collaboration challenges that are not fully addressed either by source code-centric workflows (*e.g.*, version control systems, code review, pull requests), which primarily support asynchronous work, or by centralized server architectures, which enable real-time collaboration. Practitioners instead call for modeling environments capable of supporting *both* collaboration workflows.

Addressing these expectations further entails meeting architectural requirements such as fault-tolerance, model merging, and automated conflict resolution, thereby outlining a concrete design agenda for next-generation collaborative modeling tools.

Contributions In this article, we lay the foundations for the architectural principles required to support hybrid collaborative modeling. First, we introduce a classification framework for collaboration architectures. Using this framework, we establish a comparative table of existing collaborative modeling tools and highlight their limited support for hybrid workflows. We then propose a novel architectural approach: *Local-First Collaborative Modeling*. Inspired by the Local-First software design paradigm (Kleppmann et al. 2019), this approach allows users to edit shared models locally (even without network connectivity) while automatically merging their changes upon reconnection, without compromising real-time collaboration among online participants. Finally, we outline a reference implementation approach based on replicated data types, intended to demonstrate feasibility and guide future implementations

Plan The remainder of this article is structured as follows. Section 2 reviews existing research on collaborative modeling. Section 3 introduces our classification framework for collaboration architectures and applies it to position current collaborative modeling tools within the resulting design space. Section 4 presents the requirements and implementation proposal for Local-First Collaborative Modeling. Section 5 discusses the implications of the proposed architectural principles and outlines future research directions. Finally, Section 6 concludes.

2. Related Work

In this section, we review related work on collaborative modeling. We first provide an overview of the field, then discuss the two main collaboration modalities (asynchronous and synchronous), and finally examine existing research on hybrid collaborative modeling.

2.1. Overview of Collaborative Modeling

Models provide a common language for structuring domain-specific knowledge, enabling sharing and co-construction among stakeholders (Bézivin 2005; Seidewitz 2003). Model-based Systems Engineering (MBSE) treat models as first-class artifacts used to capture and manage the complexity of system design, development, and maintenance (Estefan et al. 2007). MBSE relies on (meta)models to define the abstract syntax of Domain-Specific Languages (DSLs), which specify the structure of models.

Collaborative modeling lies at the intersection of several fundamental challenges in modeling research: *model management* (lifecycle support, versioning, and repositories), *communication and interaction* (participatory modeling, awareness features, wikis, chat), and *consistency management* (conflict detection and visualization, model merging) (Di Ruscio et al. 2017; Franzago et al. 2017). While its importance is well established, collaborative modeling continues to be poorly supported in practice, and addressing this gap is key to accelerating the industrial adoption of MBSE (Abrahão et al. 2017; Bucchiarone et al. 2020; Kuhn et al. 2012; Liebel et al. 2018).

Collaborative modeling tools implement different collaboration modalities, which may or may not align with practitioners' needs depending on evolving practices, project complexity, development processes, and team size (Demuth et al. 2015). The suitability of a tool, therefore, depends not only on its modeling capabilities, but also on the kind of collaboration scenarios it enables.

To date, collaborative modeling solutions have mainly focused on two collaboration scenarios, often treated as mutually exclusive. As a result, tools can broadly be classified into two main families: (i) tools supporting asynchronous collaboration, typically through a model-aware Version Control System (VCS), and (ii) tools supporting synchronous, real-time collaboration, generally relying on a centralized server. According to an earlier study (Franzago et al. 2018), about half of the surveyed tools support an asynchronous workflow (24 out of 48; 50%), while 40% support a synchronous workflow (20 out of 48). Only a small minority support both (4 out of 48; 8%). While this study predates several recent developments, our analysis of more recent tools (Section 2.4 and 3, and Table 1) confirms that explicit and principled support for hybrid workflows remains limited and often constrained.

2.2. Asynchronous Collaborative Modeling

At the intersection of collaborative modeling and model management, model-aware VCSs allow participants to work on their local copy of the shared model and merge their changes later, through a repository. Contributors can work in parallel on separate branches while benefiting from a complete history of modifications, allowing them to trace and understand the evolution of modeling artifacts over time (Altmanninger et al. 2009).

Unlike traditional text-based version control systems such as Git[2] or SVN[3], these tools are specifically designed to account for the graph structure and rich semantics of models (Kelly 2018; Exelmans et al. 2023). The existence of multiple concurrent versions of the model introduces the problem of conflict resolution and merge procedures. When performed manually, this is a slow and error-prone process that imposes a significant cognitive load to participants. To tackle this challenge, many model versioning tools have been proposed to automatically detect conflicts, visualize differences, and assist users in resolving them (Sharbaf et al. 2022; Koshima & Englebert 2015; Koegel & Helming 2010). In parallel, extensive research has focused on specialized model comparison and merging algorithms that better account for model semantics (Brun & Pierantonio 2008; Debreceni et al. 2017; Westfechtel 2010; Zadahmad Jafarlou & Syriani 2026). However, the support for asynchronous work generally comes at the expense of synchronous collaboration.

2.3. Synchronous Collaborative Modeling

Real-time collaborative modeling allows participants to co-edit a shared model while immediately[4] seeing each other's changes. Well-established requirements for this workflow exist (Sun et al. 1998). Research has addressed key challenges such as update propagation (Sharbaf et al. 2021), integration in different environments (*e.g.*, web-based platforms (Nicolaescu et al. 2018)), network architectures (*e.g.*, peer-to-peer (Krusche & Bruegge 2014), federation (Alfonso & Cabot 2026), cloud-based), model consistency management (*e.g.*, eventual consistency (David & Syriani 2023)), and participant awareness mechanisms (Saini & Mussbacher 2021).

The vast majority of existing tools rely on a centralized server architecture to provide real-time collaboration (Franzago et al. 2018). The server is responsible for linearizing[5] update requests, which removes the need for explicit conflict management on the client side. Such architectures do not support offline work: updating the shared model requires a permanent connection to the server. As a result, participants with high latency suffer a degraded user experience, including failed requests or repeated rollbacks.

2.4. Hybrid Collaborative Modeling

Several studies acknowledge that *hybrid collaborative modeling* is an important yet underexplored research direction (Pietron 2020; Choudhury et al. 2025; Franzago et al. 2018; David et al. 2023; Di Ruscio et al. 2017). Some recent work have begun to explore this workflow.[6]

Pietron *et al*. propose an operation-based versioning approach in which update operations on shared models are stored in a partially-ordered history represented as a directed acyclic graph (DAG), enabling versioning and transitions between real-time and offline collaboration modes in graphical modeling tools (Pietron et al. 2021). However, conflict resolution must be carried out manually by the participants.

Olivier *et al*. introduce a set of operation-based *Conflict-free Replicated Data Types* (CRDTs) (Shapiro et al. 2011), including a directed multigraph (or multidigraph) embedding a semantic conflict resolution policy. They use it to implement a Local-First collaborative UML Class Diagram (Olivier et al. 2025). However, their method requires manual re-encoding of the target metamodel, *i.e.*, composing replicated data types by hand to reflect every structural aspect of the metamodel, which limits its applicability to complex modeling languages. In addition, they do not discuss the requirements for generalizing their approach.

C-Praxis supports collaborative model editing by representing models as sequences of metamodel-independent operations over the abstract syntax (*e.g.*, creating/deleting model elements, adding/removing references between them) (Michaux et al. 2011). It builds on Telex, an optimistic replication middleware, allowing participants to alternate between frequent synchronization and disconnected work. Conflicts are handled through application-level semantic constraints and schedule selection. However, convergence is obtained by ordering or aborting conflicting changes, so some concurrent contributions may be dis-

[2] https://git-scm.com
[3] https://subversion.apache.org
[4] Modulo network latency and synchronization delays. Some authors prefer to speak of "near real-time" (Nicolaescu et al. 2018).
[5] The server processes update requests sequentially, as they arrive, without accounting for concurrent edits.
[6] Some approaches support both workflows, but not hybrid workflows. For example, MONDO supports both synchronous and asynchronous collaboration, but participants cannot transition seamlessly between the two modes.

carded and participants may experience rollbacks when tentative schedules are revised.

DesignSpace is a cloud-based artifact integration and service platform for model-driven engineering that supports distributed collaboration and artifact synchronization across tools (Demuth et al. 2015). It acts as an engineering hub rather than a modeling tool, and while it provides infrastructure for consistency checking, conflict resolution is typically realized by the participants themselves.

CoMPers supports hybrid collaborative modeling through personalized change propagation (Sharbaf et al. 2025). While it enables concurrent editing without locking, its configurable conflict detection and resolution mechanisms are realized at the centralized server.

Although these approaches demonstrate the feasibility of supporting both synchronous and asynchronous collaboration in modeling environments, they vary substantially in design. None provides a principled and unified set of architectural requirements to guide key decisions (*e.g.*, conflict management, consistency), thereby highlighting the need for a common foundation for hybrid collaborative modeling systems.

3. A Framework for Classifying Collaborative Architectures

In this section, we first introduce a framework to classify different collaborative architectures. Then, we apply this framework to a representative set of collaborative modeling tools, demonstrating its effectiveness in characterizing heterogeneous architectures and making their trade-offs explicit.

3.1. Classification Framework

Our framework builds on prior analyses of collaboration architectures (Pietron 2020; Pietron et al. 2023; Masson et al. 2017), but retains only the abstract dimensions that define the core architectural properties of collaboration principles.[7]

Figure 1 outlines our classification framework, in the form of a feature diagram (Czarnecki & Wasowski 2007). Abstract nodes correspond to collaboration dimensions, while concrete nodes represent the modalities of these dimensions. The feature diagram defines the *design space* of collaborative settings. A collaboration scenario specifies which modalities it needs, while an architecture provides those it supports. Whether an architecture is suitable for a scenario depends on whether the scenario's required modalities are covered by the architecture's supported ones.

We define *collaboration* as a process in which several actors, the *participants*, co-edit a *shared resource*. In our context, the shared resource is typically a model, which consists of an abstract syntax, defining its structure, with one or more concrete syntaxes, defining its representations (*e.g.*, diagrams). As a process, collaboration produces a *history*, understood as a set of *change events* linked by causal dependencies (Lamport 1978). A *version* of the shared resource is a prefix of the history.[8]

Collaboration is classified by temporality as either *synchronous*, when participants share a common time window, or *asynchronous*, when they do not. We propose a third category: *hybrid* collaboration. In hybrid settings, participants may alternate seamlessly between real-time and asynchronous phases. Several subgroups can work in real-time, while coordination between these groups takes place asynchronously.

A shared resource can be *centralized* or *distributed*.[9] In the first case, there is only one authoritative version of the shared resource, and participants must request changes to it. In the latter, each participant maintains a replicated copy (*i.e.*, replica) of the shared model.

A change event is characterized by its level of abstraction and its granularity. Changes to the abstract syntax are *semantic*; changes to the concrete representation are *syntactic*. An event may record an operation[10] or a state snapshot.

Changes from the same participant are totally ordered, whereas changes from different participants may be only partially ordered (*i.e.*, concurrent) when they work in parallel, such as in asynchronous workflows (Schwarz & Mattern 1994). The history of recorded events may or may not preserve this partial order. Sequential histories result either from ignoring the distributed nature of collaboration (by arbitrarily linearizing events), or from completely preventing parallel work.

A *conflict* arises when concurrent change events would lead the shared resource to divergent states (they do not commute). *Conflict management* refers to the mechanisms used to address such situations. These mechanisms may prevent conflicts by avoiding concurrency (*e.g.*, through locking or coordination protocols), delegate resolution to participants (manual merge), resolve conflicts automatically (*e.g.*, using CRDTs or model-aware merging), or use a mixed approach. When conflict resolution is automatic, it may rely either on a reordering of concurrent events (*structural* resolution) or on an interpretation of their semantics, that is, the nature and intent of the changes involved (*semantic* resolution). Structural resolution imposes an order on concurrent events, whereas semantic resolution exploits properties of the changes themselves to compute a meaningful merged result that preserves user intent.[11] Once resolution has occurred, the involved participants must observe a consistent state of the shared resource.

[7] In particular, we deliberately exclude access control, persistence mechanisms, and communication protocols from our scope. In comparison, (Pietron et al. 2023) develop fine-grained feature diagrams that cover a broad range of implementation choices and technological variants, illustrated on general-purpose collaborative software (Google Docs, Overleaf, Git) rather than modeling tools. By contrast, our framework focuses on a smaller set of fundamental dimensions, making it easier to apply, more stable with respect to technological evolution, and better suited for systematic comparison across tools. Combined with our comparative table, it provides a concrete landscape of existing modeling tools.

[8] For a partially ordered history, the notion of prefix corresponds to a downward-closed set of events.

[9] For the sake of simplicity, we consider federated resources as distributed.

[10] Operations can be classified into (pure) queries, which do not change the state, and (pure) updates, which have a side effect (Perrin 2017).

[11] Semantic conflict resolution is not restricted to semantic change events. For instance, TGRL (Saini & Mussbacher 2021) records changes at the textual concrete-syntax level, yet resolves conflicts semantically by computing appropriate insertion positions for concurrent character edits.

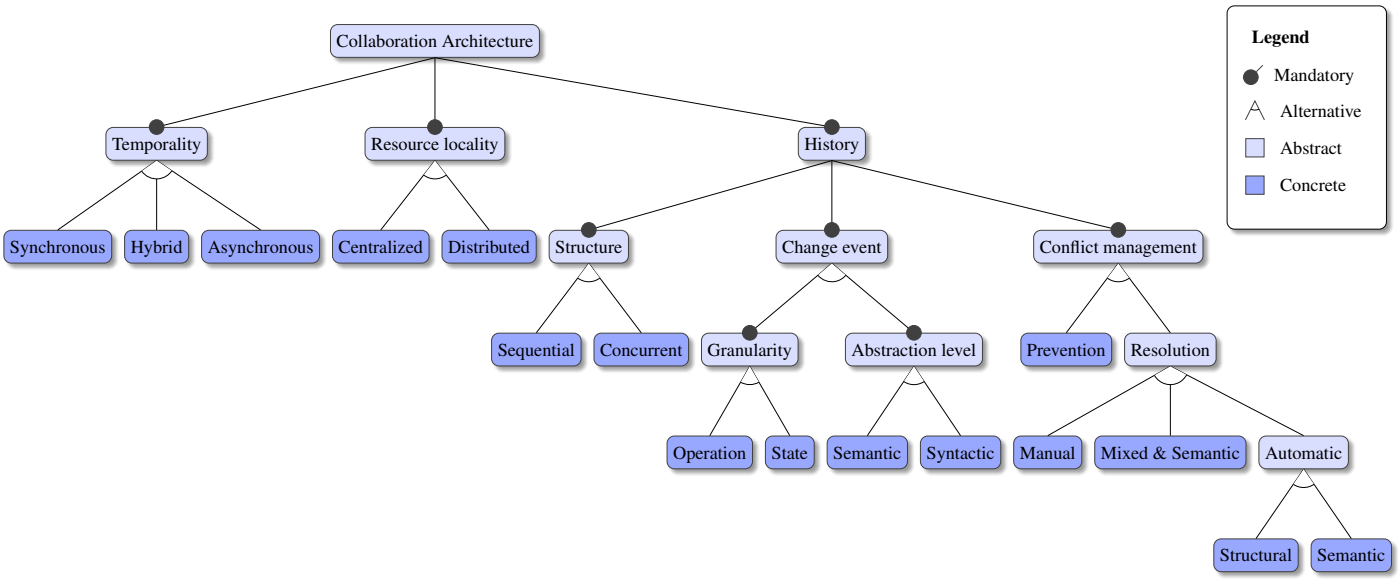

Figure 1 Feature diagram of the design space of collaborative architectures.

3.2. Comparative Table of Existing Collaborative Modeling Tools

This section demonstrates the effectiveness of the classification framework introduced in Section 3 by applying it to representative collaborative modeling tools. Table 1 confirms that explicit hybrid support remains limited. The mapped solutions are fragmented across architectural choices.

The comparative table includes tools reported in major surveys and mappings of collaborative MBSE (Franzago et al. 2018; Choudhury et al. 2025; David et al. 2023), while ensuring diversity across architectural families (centralized, distributed, and hybrid), history structures (sequential vs concurrent), and conflict-management strategies (prevention, manual resolution, mixed, automatic). Concretely, the table combines established platforms (WebGME, SiriusWeb, AToMPM, MetaEdit), distributed CRDT-oriented solutions (Lowkey, TGRL, SyncMeta), and hybrid candidates discussed in recent literature (DesignSpace, CoMPers, Pietron *et al*., Olivier *et al*., and MONDO via its online/offline modes). This sampling strategy allows meaningful cross-tool comparison within a single design space.

Our classification is intended as an analytical instrument for reasoning about collaboration architectures. It reports the claimed architectural support for each dimension and modality, based on the corresponding publications and technical documentation, rather than potential capabilities that are not explicitly designed or validated. For example, Lowkey could in principle be adapted for asynchronous collaboration, but it is classified here according to its intended synchronous design.

4. Local-First Collaborative Modeling

To effectively support hybrid workflows, collaborative modeling would benefit from a software design paradigm that integrates collaboration features at its core. One such paradigm is *Local-First*: a set of principles that prioritize responsiveness, data availability, and user ownership of data, even in distributed and offline scenarios (Kleppmann et al. 2019).

In this section, we first introduce the Local-First paradigm and its core principles, then derive architectural requirements for collaborative modeling tools that fully embrace this paradigm. Finally, we formulate an implementation proposal using replicated data types.

4.1. Local-First Software Design Paradigm

The concept of Local-First software was introduced by Kleppmann *et al*. in their 2019 essay (Kleppmann et al. 2019). It advocates an architecture in which applications operate primarily on local data while seamlessly synchronizing across devices and collaborators. The seven principles emphasize that software should be fast (no network dependency for responsiveness), multi-device (data available across user devices), offline-capable, collaborative (supporting real-time and asynchronous work), long-lived (data outlasting any single service provider), secure and private by default, and user-controlled (users retain ownership of their data). Their vision is that Local-First software can empower users with greater control over their data and overcome the traditional opposition between asynchronous and real-time collaboration.

4.2. Architectural Design Requirements

We define *Local-First collaborative modeling* as an architecture where participants edit shared models stored primarily on their own devices. Data synchronization occurs opportunistically when connectivity is available. Merging of concurrent versions happens automatically and uses a semantic conflict resolution policy to preserve participants' work as much as possible. This allows for real-time collaboration between online participants, while also supporting offline work with later merging of changes.

Building on modelers' needs reported in Table 2, we derive the following architectural requirements (**R1**-**R5**) for hybrid collaborative modeling:

Tool	Temporality	Resource Locality	History structure	Change abstraction level	Change granularity	Conflict management	Conflict resolution
(Olivier et al. 2025)	Hybrid	Distributed	Concurrent	Semantic	Operation	Resolution	Automatic (semantic)
C-Praxis[a]	Hybrid	Distributed	Concurrent	Semantic	Operation	Resolution	Automatic (semantic)
CoMPers[b]	Hybrid	Centralized	Concurrent	Semantic	Operation	Resolution	Mixed[1]
DesignSpace[c]	Hybrid	Distributed	Concurrent	Semantic	State	Resolution	Manual
Lowkey[d]	Synchronous	Distributed[2]	Sequential	Semantic	Operation	Resolution	Automatic (structural)
WebGME[e]	Synchronous[5]	Centralized	Concurrent	Semantic	State	Resolution	Manual
SiriusWeb[f]	Synchronous	Centralized	Sequential	Semantic	Operation	Prevention[3]	N/A
MetaEdit+[g]	Synchronous	Centralized	Sequential	Semantic	Operation	Prevention[4]	N/A
AToMPM[h]	Synchronous	Centralized	Sequential	Semantic	Operation	Prevention[3]	N/A
TGRL[i]	Synchronous	Distributed	Concurrent	Syntactic	Operation	Resolution	Automatic (semantic)
BUMBLE-CE[j]	Synchronous	Centralized	Sequential	Semantic	Operation	Prevention[3]	N/A
SyncMeta[k]	Synchronous	Distributed	Concurrent	Semantic	Operation	Resolution	Automatic (semantic & structural)
ModelFed[l]	Synchronous	Distributed	Sequential	Semantic	Operation	Prevention[3]	N/A
MONDO[m] (online mode)	Synchronous	Centralized	Sequential	Semantic	Operation	Prevention[3]	N/A
MONDO[m] (offline mode)	Asynchronous	Distributed	Concurrent	Syntactic	State	Resolution or Prevention[4]	Automatic (semantic)
DiCoMEF[n]	Asynchronous	Distributed	Sequential[6]	Semantic	Operation	Resolution	Manual
CDO[o]	Asynchronous[7]	Centralized	Concurrent	Semantic	Operation	Resolution or Prevention[4]	Mixed

[1] Support both manual and automatic conflict resolution. Automatic conflict resolution can be structural (*e.g.*, highest-priority-wins) or semantic (using a machine learning algorithm).
[2] Rely on a central server to dispatch updates.
[3] Server linearizes update requests.
[4] Locking mechanism on model elements.
[5] Partially support offline editing.
[6] Each participant maintains local copies of the model/metamodel and local repositories, but synchronization is controller-mediated. Accepted changes form a common sequential operation history.
[7] Partially support online editing through live transactions. However, seamless transition between online and offline modes is not supported.

[a] (Michaux et al. 2011): Peer-to-peer collaborative model editing framework based on Telex.
[b] (Sharbaf et al. 2025): Hybrid collaborative modeling through personalized change propagation.
[c] (Demuth et al. 2015): Cloud-based artifact integration and service platform for model-driven engineering.
[d] (David & Syriani 2023): Multi-level modeling using CRDTs.
[e] (Maróti et al. 2014): Web-based (meta)modeling tool for DSLs.
[f] (Giraudet et al. 2024): Web-based language workbench.
[g] (Kelly 2018): Repository-based meta(modeling) tool.
[h] (Syriani et al. 2013): Web-based multi-paradigm modeling tool.
[i] (Saini & Mussbacher 2021): Real-time goal-oriented requirement modeling in VS Code using CRDTs.
[j] (Aslam et al. 2023): Cross-platform real-time collaborative modeling platform based on EMF.Cloud.
[k] (Nicolaescu et al. 2018): Web-based conceptual modeling framework using CRDTs.
[l] (Alfonso & Cabot 2026): Federated protocol for collaborative cross-platform modeling.
[m] (Debreceni et al. 2017): Secure collaborative modeling over existing VCSs. It supports both synchronous and asynchronous collaboration, but not hybrid workflows. Participants cannot transition seamlessly between the two modes.
[n] (Koshima & Englebert 2015): Model-aware version control system for EMF (meta)models.
[o] CDO: distributed shared model repository for EMF (meta)models.

Table 1 Comparative table of collaborative modeling tools.

Practitioner need	Evidence from literature	Req.
Ability to work without continuous connectivity and resilience to server failures	95% expect offline support; 73% consider failure recovery critical (David et al. 2023)	**R1**
Support for hybrid workflows	98% expect collaboration support; 90% synchronous and 95% asynchronous (David et al. 2023)	**R2**
Fine-grained understanding of model evolution	95% expect history support; 85% version branching (David et al. 2023); traceability and evolution needs reported (Altmanninger et al. 2009; Exelmans et al. 2023)	**R3**
Reduction of cognitive load during collaboration and avoidance of manual merge overhead	77% consider automated conflict resolution important (David et al. 2023)	**R4**
Preservation of contributors' work and predictability of collaboration outcomes	Reported risks of lost updates and error-prone merges in collaborative modeling (Di Ruscio et al. 2017; Sharbaf et al. 2022)	**R5**

Table 2 Traceability from practitioner needs to architectural requirements

R1 **Local storage of models**. The system maintains a local replica of the shared model locally at each participant. This avoids reliance on opaque cloud storage and gives users direct control over their data, reducing risks related to confidentiality and intellectual property. It also ensures availability of the model even in the absence of network connectivity or in case of server failure.

R2 **Seamless transition between workflows**. The system supports synchronous (online) and asynchronous (offline) collaboration, and seamless transitions between these modes.

R3 **Fine granularity of changes**. Several works have shown that, for model versioning, it is preferable to record the operations applied rather than a succession of state snapshots (Pietron 2020; Exelmans et al. 2023). This finer granularity provides a more expressive history, enabling features such as branching, retrospective analysis, and simulation.

R4 **Automatic model merging**. Changes from different participants are merged automatically. Participants are not required to coordinate manually to restore consistency and collaboration is not interrupted by explicit conflict-resolution steps.

R5 **Preservation of contributions**. To avoid losing work during a conflict resolution, the merge should preserve the effect of each participant's changes (*i.e.*, avoid rollbacks or "lost updates") and remain transparent in how conflicts are resolved. This typically requires a *semantic* conflict resolution policy.

4.3. Implementation proposal

4.3.1. Overview In conventional modeling toolchains, a DSL is first defined by means of a metamodel, from which executable code is then generated (typically in languages such as Java or C++). The metamodel's structural elements and constraints are encoded in sequential data types. Conceptually, the resulting implementation corresponds to a multidigraph or tree of classes linked to one another. Our proposal is to generate replicated data types with well-defined behavior under concurrency, thereby making metamodel-based DSLs suitable for distributed collaborative environments.

To address model locality (**R1**) and automatic model merging (**R4**), we propose to encode metamodels as *Conflict-free Replicated Data Types* (Shapiro et al. 2011). Instances of these data types are therefore replicated objects representing the models. CRDTs are said to be *wait-free* because, unlike traditional data-replication approaches based on consensus protocols, their update operations do not require any prior coordination to be issued. The state therefore always remains available to be modified, even offline, which directly satisfies **R2**.

A CRDT satisfies *eventual consistency*: two replicas that have observed the same set of updates are in the same state. Replicas may temporarily diverge while operating in isolation, but they converge as soon as they exchange their updates, *i.e.* they are *eventually consistent*. Propagation may occur peer-to-peer or through generic syncing servers[12], making CRDTs highly flexible. Replication can be state-based or operation-based; to satisfy **R3**, we suggest to use operation-based CRDTs.

Since conflict resolution is integrated into the CRDT specification, no external merge mechanism is required. Conflicts are resolved at the appropriate level of the object hierarchy, either within the CRDT or through its parent's resolution policy. Hence, a CRDT is a reusable object that embeds the conflict specification, detection, and resolution of the data it represents. While designing correct replicated data types can be difficult, this difficulty is mitigated by the ability to compose them into more complex CRDTs (Bauwens & Gonzalez Boix 2023).

Although any replicated data type that guarantees convergence can be described as a CRDT, convergence alone is not sufficient. A *structural* conflict-resolution policy, such as assigning fixed priorities to participants and ordering concurrent updates accordingly (the well-known "Last-Writer-Wins" policy), technically achieves convergence, but often results in unpredictable data loss (Saito & Shapiro 2005). Updates may be reordered in ways that violate their preconditions, causing implicit rollbacks. Moreover, the more participants work asynchronously and in isolation, the higher the probability of conflicts. In such situations, non-semantic conflict resolution degrades the modeling experience and undermines trust in asynchronous collaboration. Semantic conflict resolution, which inspects the meaning of the conflicting operations and computes an appropriate merged result, is therefore essential to satisfy **R5**. They guarantee that conflicts involving the same operations and causal context are always resolved in the same deterministic way.

[12] Note that in that case, the server is only responsible for relaying updates, not for storing the model or performing any merge procedure.

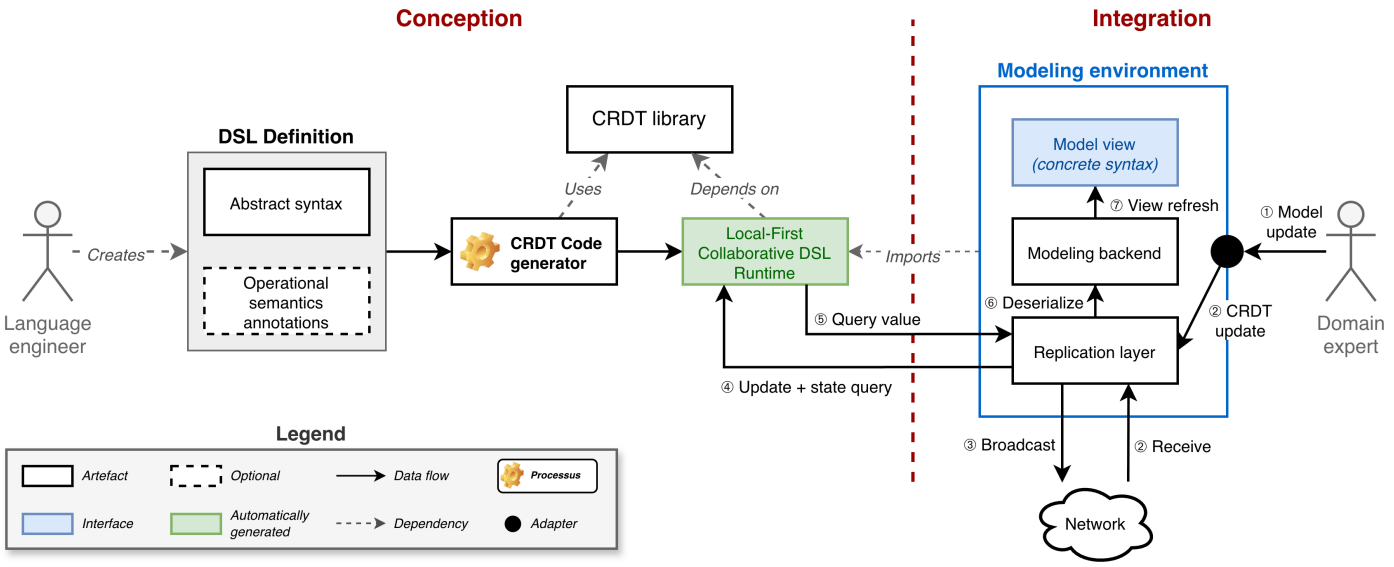

Figure 2 Implementation proposal overview. On the left-hand side (conception), a language engineer defines the metamodel of a DSL and may optionally annotate it to guide the generation of operational semantics. The metamodel is then provided to a code generator, which leverages a CRDT library to produce a Local-First collaborative DSL runtime. On the right-hand side (integration), a domain expert works within a familiar modeling environment to edit a shared model through the standard user interface. Editing actions are intercepted and translated into CRDT operations, which are applied locally and propagated to other participants when connectivity is available. The resulting CRDT state is queried and materialized into a representation understood by the modeling backend, which in turn refreshes the model view.

4.3.2. Semantic conflict-resolution with CRDTs Previous uses of CRDTs in modeling have not fully exploited their potential. Lowkey (David & Syriani 2023) relies on a Last-Writer-Wins policy.[13] TGRL (Saini & Mussbacher 2021) represents models as serialized strings using a text CRDT, resulting in a syntactic approach that fails to capture the semantics of modeling operations. SyncMeta (Nicolaescu et al. 2018) is constrained by the predefined policies and data types exposed by the underlying CRDT library.[14]

A stronger approach is to design modeling-specific CRDTs, with conflict-resolution policies that take the semantics of modeling operations into account. For example, the *Update-Wins labelled directed multigraph CRDT* proposed in (Olivier et al. 2025) addresses conflicts between the concurrent addition of an edge and the deletion of its source or target node. This data type supports nested CRDTs in its graph elements, allowing it to model complex structures and their associated content. Under its "Update-Wins" semantics, a delete operation removes the node and its existing content at the time of the deletion, but does not discard concurrent additions: newly added relations or updates to its content revive the node. As a result, neither the addition nor the deletion is lost, dangling relations are avoided, and the resulting state remains predictable to participants. While (Olivier et al. 2025) provides empirical validation via property-based fuzz testing, it does not include a formal proof of the multidigraph CRDT specification. We provide such a proof in Appendix A.1. Stricter policies are also possible, for example by fully ignoring a node deletion concurrent to an update of its content. Indeed, the fine-grained control CRDTs offer over conflict resolution allows for a wide range of policies, depending on the modeling context and the desired user experience.

CRDTs also support manual resolution when required. The Multi-Value register data type (DeCandia et al. 2007), for example, preserves all concurrent updates instead of arbitrarily selecting one, allowing participants to manually choose the appropriate value. This approach is well suited for conflicts where domain judgment is necessary.

Finally, some modeling constraints require that a property have a unique value at all times. While a Last-Writer-Wins register often adopted in that case, more suitable designs are possible. One option is a *Fair register*, which relies on a dynamic, round-robin style priority rather than fixed participant precedence. Another approach is to define a semantic total order over the set of possible values (Zawirski et al. 2016). This approach can also be combined with Multi-Value semantics when the value domain is only partially ordered.

Fair Register CRDT We formalize the Fair register[15] introduced above as an alternative to Last-Writer-Wins registers. We provide a formal proof of correctness in Appendix A.2. The state of a CRDT replica is (abstractly) modeled by its causal his-

[13] Last-Writer-Wins does not refer a precisely defined policy in the literature. In case of conflicts, ties are either broken using wall-clock time or the lexicographic order on replica identifiers.

[14] Production-ready libraries such as Loro, Yjs, or Automerge do not allow the definition of new replicated data types tailored for modeling, such as graphs.

[15] The Fair register implements a weak notion of fairness in which no participant is systematically preferred by the tie-breaking rule (Lamport 1978). A reference implementation is available at: Moirai GitHub repository - FairPolicy.

tory (Shapiro et al. 2011), *i.e.*, a partially ordered set of updates. We denote by $s \subseteq T \times U$ the CRDT state, consisting of pairs of timestamp-update (t, u), where t is a Lamport clock and $u \in \{\mathsf{write}(v) \mid v \in Val\}$. Updates delivered by a participant are appended to its local causal history. Queries are evaluated over s by the eval function. $\mathsf{origin} : T \to \{1, \ldots, n\}$ returns the replica identifier of the issuing issuing participant of t.

When queried, the Fair register returns the most recent write in its causal history. A conflict arises when two writes are concurrent, *i.e.*, when both may be considered the latest write and an arbitration is therefore required to select one. To resolve such conflicts, we define a total order on timestamps that combines Lamport clocks with a fair tie-breaking rule:

$$t \prec t' \iff t < t' \vee \big(t = t' \wedge \mathsf{rank}(t) < \mathsf{rank}(t')\big)$$

The rank function implements a rotating leader selection over participants:

```
1: function RANK(t)
2:     leader ← t mod n
3:     idx ← origin(t)
4:     return (idx + n − leader) mod n
```

The evaluation function selects the maximal write according to this order:

$$\mathsf{eval}(\mathsf{read}, s) = \begin{cases} v & \text{if } (t, \mathsf{write}(v)) \in s \\ & \quad \wedge\ t = \max_{\prec}\{t' \mid (t', \_) \in s\} \\ \bot & \text{if } s = \emptyset \end{cases}$$

Hence, CRDTs can be designed with a wide range of conflict-resolution policies, from strict to permissive, structural to semantic, and automatic to manual. This flexibility allows them to be tailored to the specific requirements of different modeling contexts and user preferences, making them a powerful tool for implementing Local-First collaborative modeling.

4.3.3. Integration path with existing modeling tools

We propose a three-step integration path for deploying a Local-First collaborative architecture in existing modeling environments. The steps are: (1) design modeling-specific CRDTs, (2) generate CRDT-based model implementations from metamodels, and (3) integrate a replication layer into existing toolchains. Figure 2 illustrates this approach.

The first step is partially addressed by previous work (Olivier et al. 2025). Our framework, written in Rust, MOIRAI[16], provides a library of composable, operation-based CRDTs and is explicitly designed to be extended with new replicated data types. MOIRAI offers reusable building blocks for defining domain-specific CRDTs, while remaining agnostic to the networking, storage, or deployment architecture.

The second step is to exploit CRDT composability through code generation. Instead of generating sequential model implementations (as in traditional toolchains such as EMF[17]), the generator maps *metamodel constructs to compositions of CRDTs*. This generator would be metamodeling-specific; practical initial targets include UML[18], SysML[19], and Ecore[20]. In practice, a language engineer defines the DSL metamodel in a standard workbench, then invokes the generator to produce a Local-First implementation conforming to that metamodel. Because the generated CRDT runtime implements the structural constructs of the metamodel, its instances correspond to replicated models expressed in the language defined by the metamodel. To cover different collaboration requirements, metamodel annotations can parameterize the generated CRDT choices and associated operational semantics.

The third step is the integration into existing modeling tools (*e.g.*, EMF.Cloud[21], SiriusWeb) through an *intermediary replication layer*. Most tools expose APIs to observe editing events. An adapter can intercept model updates, translate them into CRDT operations, and submit them to the replication layer. The layer then (i) applies operations to the local CRDT state, (ii) propagates local updates to other replicas, and (iii) materializes the current replicated state into a format understood by the modeling backend (*e.g.*, XMI, JSON). The backend deserializes this state and refreshes the model view.

4.3.4. A metamodel-independent mapping to CRDTs

We present a generic, metamodel-independent mapping to CRDTs. We do not aim to prescribe a unique conflict-resolution policy for all metamodels. Instead, we present a default mapping that preserves contributions as much as possible by using an Add-Wins policy where appropriate, in line with **R5**. Table 3 summarizes the six metamodel-independent model-building operations identified in (Blanc et al. 2008), based on the MOF reflective API.

Operation	Description
$\mathsf{create}(e, c)$	Creates a model element e of class c.
$\mathsf{delete}(e)$	Deletes an existing model element e.
$\mathsf{addProperty}(e, p, val)$	Sets property p of e to value val.
$\mathsf{removeProperty}(e, p)$	Unsets property p of element e.
$\mathsf{addRef}(e_1, r, e_2)$	Adds reference r from e_1 to e_2.
$\mathsf{removeRef}(e_1, r, e_2)$	Removes reference r from e_1 to e_2.

Table 3 Metamodel-independent model-building operations.

Conflict patterns Let $\|$ denote concurrent execution. A conflict occurs when two concurrent updates do not commute in the current state. By examining all pairs of non-commutative operations, we identify the following potential conflicts:

- $\mathsf{addRef}(e_1, r, e_2) \parallel \mathsf{removeRef}(e_1', r', e_2')$ if $r = r' \wedge (e_1 = e_1' \vee e_2 = e_2')$.
- $\mathsf{addProperty}(e, p, val) \parallel \mathsf{removeProperty}(e', p')$ if $e = e' \wedge p = p'$.

[16] https://github.com/CEA-LIST/Moirai
[17] https://eclipse.dev/emf
[18] https://www.omg.org/spec/UML
[19] https://sysml.org
[20] https://eclipse.dev/emf
[21] https://eclipse.dev/emfcloud

– $\mathsf{addProperty}(e, p, val) \parallel \mathsf{addProperty}(e', p', val')$ if $e = e' \wedge p = p' \wedge val \neq val'$.
– $\mathsf{addProperty}(e, p, val) \parallel \mathsf{delete}(e')$ if $e = e'$.
– $\mathsf{addRef}(e_1, r, e_2) \parallel \mathsf{delete}(e')$ if $e_1 = e' \vee e_2 = e'$.

Composition Given a MOF-based metamodel, we can design a composition of CRDTs that support these model-building operations. Such a composition forms a tree of causal histories (Bauwens & Gonzalez Boix 2023). Query evaluation is therefore recursive. A query is first interpreted at the root, which determines, according to its conflict-resolution policy, which children are live and which portions of their histories remain visible (it *causally resets* it). The query is then propagated to the selected children, and the process continues down to the leaves. For example, in an Update-Wins labelled multidigraph CRDT, removing a vertex causes subsequent queries to ignore all updates stored in the corresponding child history that causally precede the removal, while still preserving updates that are concurrent with it.

Mapping A metamodel is a tuple $M = \langle C, P, R, \mathsf{prop}, \mathsf{ref} \rangle$, where C is a set of classes, P a set of properties, and R a set of reference types. The function $\mathsf{prop} : C \to \mathcal{P}(P)$ maps each class to the properties available for its instances, while $\mathsf{ref} : C \to \mathcal{P}(R \times C)$ returns the reference types that may originate from instances of a class, together with their admissible target classes.

The corresponding replicated object is a composite Update-Wins labelled multidigraph CRDT $G_M = \langle V, E, A, \mathsf{elem}, \mathsf{type} \rangle$. V is a set of model-element identifiers (the vertices) and E a set of reference identifiers (the edges). The set $A \subseteq V \times V \times E$ contains the references between model elements (the arcs of the graph). The function $\mathsf{elem} : V \to \mathit{Record} \times C$ maps each element identifier to a replicated object representing an instance of class C and its properties. More precisely, this object is a *record CRDT*, *i.e.* a map from property names to replicated fields. For each class $c \in C$, the associated record is built from M:

$$\mathsf{record}_M(c) = \mathsf{Record}\{p \mapsto \mathsf{field}(p) \mid p \in \mathsf{prop}(c)\}$$

The auxiliary function field determines the CRDT assigned to each property. Operations on distinct record fields commute naturally. Concurrent operations on the same field may conflict, in which case resolution is delegated to the CRDT chosen for that field, according to the mapping given in Table 4.

Finally, $\mathsf{type} : E \to \mathit{Register}\langle R \rangle$ maps each reference identifier to its reference type, stored in a register CRDT such as the Fair Register introduced earlier. A graph instance conforms to the metamodel when each arc respects the reference declarations of M: for every $(v_1, v_2, e) \in A$, if $\mathsf{elem}(v_1) = (c_1, \_)$, $\mathsf{elem}(v_2) = (c_2, \_)$, and $\mathsf{type}(e) = r$, then $(r, c_2) \in \mathsf{ref}(c_1)$. Consequently, classes determine the possible vertex types, properties determine the fields of the record CRDT stored at each vertex, and references determine the typed edges of the graph.

MOF data type	CRDT	Conflict-resolution policy
Number	Counter	Concurrent increments and decrements commute.
Boolean	Flag	Concurrent enable and disable operations are resolved using Enable-Wins semantics.
String	Sequence of characters	Concurrent insertions are ordered deterministically; removals affect only observed characters.
Enumeration	Register	Concurrent assignments are preserved (Multi-Value) or arbitrated by a Fair policy.

Table 4 Mapping from MOF primitive data types to CRDTs.

Conflict resolution Our default policy favors preservation of concurrent contributions:

– *Concurrent addition and removal of a reference:* the reference is retained under add-wins semantics, with the type assigned by the concurrent addition.
– *Concurrent assignment and removal of a property value:* the removal clears the effects of causally preceding assignments, while concurrent assignments remain visible.
– *Concurrent assignments of distinct values to the same property:* the conflict is resolved by the CRDT associated with that property, according to Table 4.
– *Concurrent deletion of an element and update of one of its properties:* the deletion removes the effects of causally preceding updates on the element, while the concurrent property update preserves the element identifier, its class, and assigns value v to property p.
– *Addition of a reference to a concurrently deleted element:* to prevent dangling references, the added reference is recorded but remains hidden as long as one of its endpoints is absent from the resolved vertex set. It becomes visible again whenever both endpoints are present.[22]

The mapping guarantees convergence and supports all structurally constructible models of a metamodel. However, structural constraints beyond local typing (*e.g.*, acyclicity, multiplicities, containment, and abstractness) require additional modeling-specific CRDTs or invariant-preservation mechanisms.

5. Discussion & Future Work

Local-First collaborative modeling is a promising approach, but it requires further research and development to be fully realized.

5.1. Modeling-specific CRDTs

The design of modeling-specific CRDTs is a central research challenge, as their semantics directly shape the user experience

[22] Consider three participants p_1, p_2, and p_3 co-editing a model with two elements, m_1 and m_2. Participant p_1 deletes m_1, while p_2 concurrently adds a reference from m_1 to m_2. When the deletion is observed, m_1 is absent and the reference is hidden to avoid a dangling edge, although the reference addition remains recorded. If p_3 concurrently updates a property of m_1, update-wins semantics revives m_1 in a default state with that property set. The hidden reference then becomes visible again, since both endpoints are present.

of conflict resolution. We do not envision a single universal solution, but rather a family of CRDTs offering different semantic trade-offs depending on the modeling context. Given the diversity of modeling formalisms, specialized replicated data types are required. For example, although CRDTs have been developed for simple graphs (Shapiro et al. 2011), directed multigraphs (Olivier et al. 2025), DAGs (Borth et al. 2025), and hypergraphs (Bansal 2022), no *typed graph* CRDT has yet been proposed. A typed graph defines a set of vertex and edge types, specifying exactly how those vertices can be linked, including edge multiplicities and other constraints such as acyclicity, abstractness, *etc*. (Taentzer & Rensink 2005). Consequently, typed graphs can directly encode the structure and invariants of a metamodel.

5.2. Limitations of eventual consistency

Because they are designed to process a partial order rather than a sequence of updates, most CRDTs do not share the same operational semantics as their sequential counterparts. Even seemingly simple operations, such as moving an element in a list CRDT, requires a thorough investigation and are difficult to implement correctly (Da & Kleppmann 2024). Important collaborative features, such as undo/redo and access control, remain open problems in eventually consistent systems (Preguiça 2018).

Moreover, it is not always possible to preserve all contributions without violating model invariants. Some global invariants require coordination, and cannot be enforced by CRDTs alone. However, we argue that this is not a fundamental limitation. Collaborative modeling, as a process of co-construction, should allow for temporary inconsistencies (Wieland et al. 2012). Conformity checking and issue awareness mechanisms can be implemented at a higher level than the replicated layer, ensuring that the model eventually satisfies its invariants without sacrificing the benefits of Local-First collaboration.

Finally, eventual consistency is not a universal solution. Empirical evidence shows that only 53% of practitioners explicitly express a need for eventual consistency in modeling tools (David et al. 2023). However, eventual consistency should not be viewed as a feature in itself, but rather as an architectural trade-off that enables the support of both real-time and asynchronous collaboration. As a result, certain collaboration scenarios remain better served by coordination-based mechanisms such as locking. Our proposal therefore does not aim to replace existing workflows, but to complement them by supporting the emerging class of hybrid collaborative modeling scenarios for which eventual consistency and Local-First principles are particularly well suited.

5.3. Integration

Requirement **R3**, *i.e*. fine-grained changes and explicit operation recording, has significant architectural implications. Capturing semantic operations requires deeper integration with modeling tools, making recorders more invasive than snapshot-based approaches. Indeed, one of Git's strengths lies in its independence from the editing tool, as it operates purely on file states.

Fortunately, modern CRDT approaches (Baquero et al. 2017), such as the one implemented in MOIRAI, decouple data-type semantics from both the replication protocol and the storage layer. This separation enables flexible deployment configurations: CRDTs can be replicated in peer-to-peer settings or across federations of servers (Alfonso & Cabot 2026), while their partially ordered logs of updates may be persisted using heterogeneous storage backends. Such flexibility opens the possibility of reusing model-aware version control systems, such as the one proposed in (Exelmans et al. 2023), as storage infrastructures for Local-First collaborative modeling.

5.4. Beyond the abstract syntax

So far, our discussion has focused on conflict resolution at the level of abstract syntax. However, conflicts may also arise in concrete syntaxes (*e.g*., in element layout, graphical styling, *etc*.) and these deserve dedicated investigation. The use of CRDT-based techniques at this layer remains to be explored, but they offer a promising direction for supporting consistent and predictable collaboration across multiple representations.

6. Conclusion

In this article, we have shown, through a classification framework of collaboration architectures, that emerging *hybrid collaboration workflows* remain poorly supported by current modeling tools, despite corresponding to a clear and growing need among practitioners.

To address this gap, we proposed adapting the *Local-First software design paradigm* to collaborative modeling in order to natively support hybrid workflows. We derived a set of architectural requirements and outlined a concrete integration path based on *replicated data types*. Adopting this architectural stance has several important implications. First, collaboration becomes independent of permanent server availability: each participant maintains a replica of the shared model and can continue working offline. Second, concurrency is treated as the norm rather than the exception, shifting the focus from preventing conflicts to designing predictable and meaningful resolution semantics. Third, conflict resolution is elevated from a merge utility to a first-class semantic concern, embedded directly in the modeling language and its underlying data structures.

A first implementation, realized in the MOIRAI framework, already covers part of the approach and has enabled the practical identification and validation of some architectural features. Future work will focus on extending this prototype to support the full Local-First collaborative modeling architecture, evaluating it through real-world MBSE case studies, and studying its impact on collaborative workflows and user experience.

A. Formal proofs

A correct CRDT satisfies the properties of the Strong Eventual Consistency (SEC) criterion (Shapiro et al. 2011):

– **Convergence**: Any two replicas whose causal histories are equal are in the same state; *i.e*., all queries over these histories return the same values.

– **Eventual visibility**: Every update operation issued by a correct replica is eventually included in the causal history of every other correct replica.[23]

A.1. Correctness of the multidigraph CRDTs

We provide a formal proof of correctness for the (labelled) multidigraph CRDT specifications, covering both the simple ("Add-Wins") variant and the nested ("Update-Wins") variant, in which vertices and edges may themselves contain nested CRDTs. We do not recall the algorithms here; readers can refer to (Olivier et al. 2025) for details. A reference implementation is publicly available in the MOIRAI repository.[24]

The eventual visibility property is ensured by assuming a reliable causal broadcast protocol (Cachin et al. 2011). We therefore focus on proving convergence using VeriFx, an automated verification language for CRDTs (De Porre et al. 2024). A valid execution is a causal history such that, for each update u, its preconditions are satisfied in the state obtained from its causal past before u is added to the history. For the data types considered, the precondition requires that the vertices referenced by an update adding an arc already exist.

A.1.1. Add-Wins multidigraph

Theorem A.1 (Convergence of the Add-Wins multidigraph). For any valid execution of the AW-multidigraph, the convergence property of SEC holds.

Proof. The theorem is machine-checked in VeriFx.[25] The formalization proves that concurrent updates commute, *i.e.* their effect on the state is independent of their delivery order, and that the evaluation function is deterministic. Together, these properties imply convergence for all replicas that have delivered the same set of updates. □

In addition, the VeriFx specification proves that evaluating any valid causal history yields a well-formed multidigraph satisfying the expected structural invariants and Add-Wins semantics.

A.1.2. Update-Wins labelled multidigraph

A composite replicated data type, such as the Update-Wins labelled multidigraph, maintains a set of owned child CRDTs, which may themselves be composite and contain further nested CRDTs.

To avoid reasoning directly about arbitrarily deep hierarchies of nested CRDTs, we abstract all nested components as a single correct resettable operation-based CRDT. This abstraction is parameterized by the properties required from the nested component, namely SEC convergence and correct reset behavior. It therefore reduces the verification problem to a single level of nesting.

Theorem A.2 (Convergence of the Update-Wins labelled multidigraph). For any valid execution of the UW-multidigraph, and for any correct resettable CRDT nested within it, the convergence property of SEC holds.

Proof. The theorem is machine-checked in VeriFx.[26] The formalization proves that concurrent updates commute, including updates that affect nested CRDTs, and that reset operations preserve the convergence guarantees of the nested component. These properties imply SEC convergence for replicas that have delivered the same set of updates. □

A.2. Correctness of the Fair register CRDT

Theorem A.3 (Correctness of the Fair Register). For any well-formed causal history of write operations, the Fair Register satisfies SEC convergence. Moreover, every read returns at most one value, and this value was produced by a write operation.

Proof. The theorem is machine-checked in VeriFx.[27] The formalization proves that the ordering used to select the visible write is total over valid timestamps, and that the read function is deterministic, returns only written values, and returns at most one value. These properties imply that replicas which have delivered the same set of writes compute the same register value, hence SEC convergence. □

The VeriFx specification also establishes the fairness property: no replica is permanently excluded from winning conflicts, since each replica has the highest priority for infinitely many Lamport counter values.

Acknowledgments

This work was partially funded by project ANR Carnot Decloud.

References

Abrahão, S., Bourdeleau, F., Cheng, B., Kokaly, S., Paige, R., Störrle, H., & Whittle, J. (2017, Sep). User experience for model-driven engineering: Challenges and future directions. In *Models* (pp. 229–236). doi: 10.1109/MODELS.2017.5

Alfonso, I., & Cabot, J. (2026). Modelfed: A federated protocol for collaborative cross-platform modeling. In *Er* (pp. 45–62). Springer.

Altmanninger, K., Seidl, M., & Wimmer, M. (2009, Aug). A survey on model versioning approaches. *Int. J. Web Inf. Syst.*, *5*(3), 271–304. doi: 10.1108/17440080910983556

Aslam, K., Chen, Y., Butt, M., & Malavolta, I. (2023). Cross-platform real-time collaborative modeling: An architecture and a prototype implementation via emf.cloud. *IEEE Access*, *11*, 49241–49260. doi: 10.1109/ACCESS.2023.3276872

Bansal, A. (2022). Hypergraphs conflict-free partially replicated data types. In *Dexa* (pp. 417–432). Springer. doi: 10.1007/978-3-031-12423-5_32

Baquero, C., Almeida, P. S., & Shoker, A. (2017). Pure operation-based replicated data types. *CoRR*, *abs/1710.04469*. Retrieved from http://arxiv.org/abs/1710.04469

[23] Informally, this means that non-crashed replicas cannot remain disconnected indefinitely, the network cannot remain permanently partitioned, and every update issued by a replica is eventually delivered to all other non-crashed replicas.

[24] Moirai GitHub repository - graph

[25] VeriFx formalization of the AW-multidigraph.

[26] VeriFx formalization of the UW-multidigraph.

[27] VeriFx formalization of the Fair Register.

Bauwens, J., & Gonzalez Boix, E. (2023, Jul). Nested pure operation-based crdts. In *Ecoop* (pp. 1–26). doi: 10.4230/LIPIcs.ECOOP.2023.2

Bézivin, J. (2005, May). On the unification power of models. *Softw. Syst. Model.*, *4*(2), 171–188. doi: 10.1007/s10270-005-0079-0

Blanc, X., Mounier, I., Mougenot, A., & Mens, T. (2008). Detecting model inconsistency through operation-based model construction. In *Icse* (pp. 511–520). ACM. doi: 10.1145/1368088.1368158

Borth, E., Lersch, P., & Bieniusa, A. (2025). Directed acyclic graph crdts. In *Papoc* (pp. 30–37). ACM. doi: 10.1145/3721473.3722141

Brun, C., & Pierantonio, A. (2008). Model differences in the eclipse modeling framework. *UPGRADE*, *9*(2), 29–34.

Bucchiarone, A., Cabot, J., Paige, R. F., & Pierantonio, A. (2020, Jan). Grand challenges in model-driven engineering: an analysis of the state of the research. *Softw. Syst. Model.*, *19*(1), 5–13. doi: 10.1007/s10270-019-00773-6

Cachin, C., Guerraoui, R., & Rodrigues, L. (2011). *Introduction to reliable and secure distributed programming* (2nd ed.). Springer.

Choudhury, A., Malavolta, I., Ciccozzi, F., Aslam, K., & Lago, P. (2025, Feb). The technological landscape of collaborative model-driven software engineering. *Softw. Syst. Model.*, *24*(5), 1595–1619. doi: 10.1007/s10270-025-01274-5

Czarnecki, K., & Wasowski, A. (2007). Feature diagrams and logics: There and back again. In *Splc* (pp. 23–34). IEEE.

Da, L., & Kleppmann, M. (2024). Extending json crdts with move operations. In *Papoc* (pp. 8–14). ACM. doi: 10.1145/3642976.3653030

Dagenais, K., & David, I. (2025, Oct). Complex model transformations by reinforcement learning with uncertain human guidance. In *Models* (pp. 209–220). doi: 10.1109/MODELS67397.2025.00025

David, I., Aslam, K., Malavolta, I., & Lago, P. (2023). Collaborative model-driven software engineering — a systematic survey of practices and needs in industry. *J. Syst. Softw.*, *199*, 111626. doi: 10.1016/j.jss.2023.111626

David, I., & Syriani, E. (2023, Aug). Real-time Collaborative Multi-Level Modeling by Conflict-Free Replicated Data Types. *Softw. Syst. Model.*, *22*(4), 1131–1150. doi: 10.1007/s10270-022-01054-5

Debreceni, C., Bergmann, G., Búr, M., Ráth, I., & Varró, D. (2017). The mondo collaboration framework: secure collaborative modeling over existing version control systems. In *Esec/fse* (pp. 984–988). ACM. doi: 10.1145/3106237.3122829

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., … Vogels, W. (2007). Dynamo: amazon's highly available key-value store. In *Sosp* (pp. 205–220). ACM. doi: 10.1145/1294261.1294281

Demuth, A., Riedl-Ehrenleitner, M., Nöhrer, A., Hehenberger, P., Zeman, K., & Egyed, A. (2015). Designspace: an infrastructure for multi-user/multi-tool engineering. In *Sac* (pp. 1486–1491). ACM. doi: 10.1145/2695664.2695697

De Porre, K., Ferreira, C., & Gonzalez Boix, E. (2024). *Verifx: Correct replicated data types for the masses.* Retrieved from https://arxiv.org/abs/2207.02502

Di Ruscio, D., Franzago, M., Malavolta, I., & Muccini, H. (2017, May). Envisioning the future of collaborative model-driven software engineering. In *Icse-c* (pp. 219–221). doi: 10.1109/ICSE-C.2017.143

Estefan, J. A., et al. (2007). Survey of model-based systems engineering (mbse) methodologies. *Incose MBSE Focus Group*, *25*(8), 1–12.

Exelmans, J., Pietron, J., Raschke, A., Vangheluwe, H., & Tichy, M. (2023). A new versioning approach for collaboration in blended modeling. *J. Comput. Lang.*, *76*, 101221. doi: 10.1016/j.cola.2023.101221

Franzago, M., Di Ruscio, D., Malavolta, I., & Muccini, H. (2017). Collaborative model-driven software engineering: reflections on the past and visions of the future. In *Icse-c* (pp. 1–5).

Franzago, M., Di Ruscio, D., Malavolta, I., & Muccini, H. (2018, Dec). Collaborative model-driven software engineering: A classification framework and a research map. *IEEE Trans. Softw. Eng.*, *44*(12), 1146–1175. doi: 10.1109/TSE.2017.2755039

Giraudet, T., Bats, M., Blouin, A., Combemale, B., & David, P.-C. (2024). Sirius Web: Insights in Language Workbenches - An Experience Report. *J. Object Technol.*, *23*(1), 1–20. doi: 10.5381/jot.2024.23.1.a6

Kelly, S. (2018). Collaborative modelling with version control. In *Staf* (pp. 20–29). Springer.

Kleppmann, M., Wiggins, A., van Hardenberg, P., & McGranaghan, M. (2019). Local-first software: you own your data, in spite of the cloud. In *Onward!* (pp. 154–178). ACM. doi: 10.1145/3359591.3359737

Koegel, M., & Helming, J. (2010). Emfstore: a model repository for emf models. In *Icse* (pp. 307–308). ACM. doi: 10.1145/1810295.1810364

Koshima, A. A., & Englebert, V. (2015). Collaborative editing of emf/ecore meta-models and models: Conflict detection, reconciliation, and merging in dicomef. *Sci. Comput. Program.*, *113*, 3–28. doi: 10.1016/j.scico.2015.07.004

Krusche, S., & Bruegge, B. (2014). Model-based real-time synchronization. *Softwaretechnik-Trends*, *34*(2).

Kuhn, A., Murphy, G. C., & Thompson, C. A. (2012). An exploratory study of forces and frictions affecting large-scale model-driven development. In *Models* (pp. 352–367). Springer. doi: 10.1007/978-3-642-33666-9_23

Lamport, L. (1978, Jul). Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, *21*(7), 558–565. doi: 10.1145/359545.359563

Liebel, G., Marko, N., Tichy, M., Leitner, A., & Hansson, J. (2018, Feb). Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. *Softw. Syst. Model.*, *17*(1), 91–113. doi: 10.1007/s10270-016-0523-3

Maróti, M., Kecskes, T., Kereskényi, R., Broll, B., Völgyesi, P., Jurácz, L., … Ledeczi, A. (2014, Jan). Next generation (meta)modeling: Web- and cloud-based collaborative tool infrastructure. *CEUR Workshop Proc.*, *1237*, 41–60.

Masson, C., Corley, J., & Syriani, E. (2017). Feature model for

collaborative modeling environments. In *Models*. Retrieved from https://api.semanticscholar.org/CorpusID:39560632
McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2018, Mar). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. *Hum. Factors*, *60*(2), 262–273. doi: 10.1177/0018720817743223
Michaux, J., Blanc, X., Shapiro, M., & Sutra, P. (2011). A semantically rich approach for collaborative model edition. In *Proceedings of the 2011 acm asac* (p. 1470–1475). ACM. Retrieved from https://doi.org/10.1145/1982185.1982500 doi: 10.1145/1982185.1982500
Nicolaescu, P., Rosenstengel, M., Derntl, M., Klamma, R., & Jarke, M. (2018, May). Near real-time collaborative modeling for view-based Web information systems engineering. *Inf. Syst.*, *74*, 23–39. doi: 10.1016/j.is.2017.07.008
Olivier, L., Morcos, K., Del Fabro, M. D., & Gerard, S. (2025, Oct). A Local-First Collaborative Modeling Approach with Replicated Data Types. In *Copamo (models-c)*. Retrieved from https://cea.hal.science/cea-05322894
Perrin, M. (2017). *Distributed systems: concurrency and consistency*. Elsevier.
Pietron, J. (2020). Enhancing collaborative modeling. In *Models-c* (pp. 1–6). ACM. doi: 10.1145/3417990.3419490
Pietron, J., Füg, F., & Tichy, M. (2021). An operation-based versioning approach for synchronous and asynchronous collaboration in graphical modeling tools. In *Staf workshops* (Vol. 2999, pp. 88–89). CEUR-WS.org. Retrieved from https://ceur-ws.org/Vol-2999/fpvmdata4mdepaper3.pdf
Pietron, J., Raschke, A., Exelmans, J., & Tichy, M. (2023, Oct). Collaboration and versioning framework – a systematic top-down approach. In *Models-c* (pp. 767–777). doi: 10.1109/MODELS-C59198.2023.00124
Preguiça, N. M. (2018). Conflict-free replicated data types: An overview. *CoRR*, *abs/1806.10254*. Retrieved from http://arxiv.org/abs/1806.10254
Rädler, S., Berardinelli, L., Winter, K., Rahimi, A., & Rinderle-Ma, S. (2024, Sep). Bridging mde and ai: a systematic review of domain-specific languages and model-driven practices in ai software systems engineering. *Softw. Syst. Model.*, *24*(2), 445–469. doi: 10.1007/s10270-024-01211-y
Saini, R., & Mussbacher, G. (2021, Oct). Towards conflict-free collaborative modelling using vs code extensions. In *Models-c* (pp. 35–44). doi: 10.1109/MODELS-C53483.2021.00013
Saito, Y., & Shapiro, M. (2005, Mar). Optimistic replication. *ACM Comput. Surv.*, *37*(1), 42–81. doi: 10.1145/1057977.1057980
Schwarz, R., & Mattern, F. (1994, Mar). Detecting causal relationships in distributed computations: in search of the holy grail. *Distrib. Comput.*, *7*(3), 149–174. doi: 10.1007/BF02277859
Seidewitz, E. (2003, Sep). What models mean. *IEEE Softw.*, *20*(5), 26–32. doi: 10.1109/MS.2003.1231147
Shapiro, M., Preguiça, N., Baquero, C., & Zawirski, M. (2011, Jan). *A comprehensive study of Convergent and Commutative Replicated Data Types* (Tech. Rep. No. RR-7506). Inria. Retrieved from https://inria.hal.science/inria-00555588
Sharbaf, M., Zamani, B., & Sunyé, G. (2021, Oct). Towards personalized change propagation for collaborative modeling. In *Howcom (models)*. Retrieved from https://hal.science/hal-03327055
Sharbaf, M., Zamani, B., & Sunyé, G. (2022). Conflict management techniques for model merging: A systematic mapping review. *Softw. Syst. Model.*. doi: 10.1007/s10270-022-01050-9
Sharbaf, M., Zamani, B., & Sunyé, G. (2025). Compers: A configurable conflict management framework for personalized collaborative modeling. *J. Syst. Softw.*, *219*, 112227. doi: 10.1016/j.jss.2024.112227
Sun, C., Jia, X., Zhang, Y., Yang, Y., & Chen, D. (1998, Mar). Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Trans. Comput.-Hum. Interact.*, *5*(1), 63–108. doi: 10.1145/274444.274447
Syriani, E., Vangheluwe, H., Mannadiar, R., Hansen, C., Van Mierlo, S., & Ergin, H. (2013). Atompm: A web-based modeling environment. In *Models companion* (pp. 21–25).
Taentzer, G., & Rensink, A. (2005). Ensuring structural constraints in graph-based models with type inheritance. In *Fase* (pp. 64–79). Springer. doi: 10.1007/978-3-540-31984-9_6
Westfechtel, B. (2010). A formal approach to three-way merging of emf models. In *Iwmcp* (pp. 31–41). ACM. doi: 10.1145/1826147.1826155
Wieland, K., Langer, P., Seidl, M., Wimmer, M., & Kappel, G. (2012). Turning conflicts into collaboration - concurrent modeling in the early phases of software development. *CSCW*, *22*(2-3), 181–240. doi: 10.1007/s10606-012-9172-4
Zadahmad Jafarlou, M., & Syriani, E. (2026). Domain-specific conflict resolution and model merge. *J. Syst. Softw.*, *232*, 112694. doi: 10.1016/j.jss.2025.112694
Zawirski, M., Baquero, C., Bieniusa, A., Preguiça, N., & Shapiro, M. (2016). Eventually consistent register revisited. In *Papoc*. ACM. doi: 10.1145/2911151.2911157

About the authors

Léo Olivier is PhD student at CEA-List, working on local-first software and collaborative modeling environments. You can contact the author at leo.olivier@cea.fr.

Marcos Didonet Del Fabro is a research engineer at CEA List. His activities focus on developing and researching intelligent and collaborative modeling architectures. You can contact the author at marcos.didonetdelfabro@cea.fr.

Sébastien Gérard is expert at IRT Jules Vernes. He leads the flagship program on MBSE for industrial robotics domain. His research interests include complex systems, model-based engineering, autonomisation of systems, knowledge AI-empowered engineering and visual modeling, collaborative tools, digital transformation, and its impact on society. You can contact the author at sebastien.gerard@irt-jules-vernes.fr.