ModelFed: a Federated Protocol for Collaborative Cross-Platform Modeling

Iván Alfonso[1], Jordi Cabot[1,2]

[1] Luxembourg Institute of Science and Technology, Esch-sur-Alzette, Luxembourg
{ivan.alfonso, jordi.cabot}@list.lu
[2] University of Luxembourg, Esch-sur-Alzette, Luxembourg

Abstract. Modeling is inherently a collaborative process, involving multiple stakeholders with different expertise and technical levels, often using a myriad of modeling tools, each one tailored to their individual profiles. This poses a major interoperability challenge, even more when we want this collaboration to happen in real-time and without relying on a centralized node. Indeed, a centralized architecture introduces scalability, privacy and reliability concerns.
In this sense, this paper presents ModelFed, a federated protocol for real-time collaboration across heterogeneous and distributed modeling platforms. ModelFed, currently focused on data models but extendable to others, enables participant platforms to maintain their own data, policies, and model representations while synchronizing changes efficiently. We evaluate our approach in a multi-platform modeling environment, demonstrating consistent model synchronization and practical efficiency for distributed stakeholders.

Keywords: Collaborative modeling · cross-platform · model federation.

1 Introduction

Information systems development is naturally a collaborative activity involving various stakeholders working together to build a system [19]. This is also true for approaches like Model-Driven Engineering (MDE), where conceptual models serve as the foundation for designing, using, analyzing, and transforming models during the software development process. Indeed, in conceptual modeling, multiple stakeholders with diverse knowledge and expertise (from the development team to non-technical stakeholders and domain experts) often must contribute collaboratively to defining the system-to-be.

Although this need is recognized in many modeling methods and frameworks, that support collaborative modeling, they typically require all actors to use the same tool. This greatly limits the collaboration by imposing restrictions on all actors (not only on the tool to be used, but also on the modeling language and even the syntax), forcing all participants to learn those even if, due to their profile, each participant would prefer to use a different tool and language [12].

Beyond cross-platform modeling, an ideal collaboration environment should also be not only distributed but fully decentralized. As solutions relying on a

central server introduce single failure points, hamper scalability, and raise privacy and control concerns for participants.

This paper proposes such a collaborative modeling architecture. Our approach is inspired by the concept of Fediverse [17], an interconnected and decentralized ecosystem of social platforms. In Fediverse, independent platforms can interact with each other, enabling users to communicate and share content. One of its key advantages is that each platform retains control over its own data and infrastructure, improving privacy, data ownership, and security [17]. The Fediverse primarily ensures interoperability between platforms by using the ActivityPub[3] protocol, which enables the federation of independent servers and reduces dependence on large social media companies. However, while ActivityPub defines general purpose activities (such as *Create*, *Update*, and *Delete*), it lacks domain-specific semantics required for collaborative modeling, such as representing model elements (e.g. classes and relationships).

With a vision similar to ActivityPub and the Fediverse, but focused on model federation, we propose ModelFed and Modelverse, a decentralized ecosystem that enables collaborative modeling across different platforms. ModelFed, the protocol that enables platforms to communicate, extends ActivityPub by introducing new concepts tailored to modeling collaboration, particularly for the creation and maintenance of structural models. Modelverse adopts a federated architecture, where modeling platforms maintain their own data and policies but synchronize changes via a shared protocol, eliminating the need for a central server while ensuring interoperability.

The remainder of this paper is structured as follows. Section 2 establishes foundational concepts and background. Section 3 introduces our running example, which illustrates key concepts throughout the paper. We then present our core contributions: Section 4 details the Modelverse ecosystem, Section 5 describes the ModelFed vocabulary extension, and Section 6 explains the runtime implementation. Tool support and prototyping are discussed in Section 7, followed by experimental validation in Section 8. Section 9 examines the extensibility of the protocol for new domains. Finally, Section 10 reviews related work, and Section 11 concludes with future directions.

2 Background

2.1 Federated systems

The term federation in software and systems engineering refers to the collaboration between multiple systems, databases, services, or platforms without requiring centralization of all data or resources in a single location [25]. Federated systems share information, resources, or functionalities in a coordinated manner while maintaining their autonomy.

The main advantages of federated systems are their interoperability, as they enable seamless communication and synchronized information sharing; scalability, achieved through resource decentralization, which avoids reliance on a single

[3] `https://www.w3.org/TR/activitypub/`

point of failure; security, since each system can store and protect sensitive information locally, minimizing exposure risks; and autonomy, as systems maintain their independence while collaborating within the federated framework [5].

2.2 Fediverse and ActivityPub

The Fediverse is a very popular example of a federated system in which a federated network of interconnected servers is used for social networking and content publishing. Each server operates independently, but communicates with others using a shared protocol called ActivityPub. As a federated system, there is no central corporation, platform, back-end, or website that coordinates and controls all activity. Instead, users can interact on servers (following, sharing, and messaging) regardless of which platform or provider they use.

ActivityPub is the protocol that enables communication between social networks in the Fediverse. Established by the World Wide Web Consortium (W3C), it facilitates interoperability by defining two decentralized APIs: (1) client-to-server API, which enables interactions between users and platforms, and (2) server-to-server API, which specifies the procedures that servers follow to federate or distribute activities across different servers.

The data format and vocabulary of ActivityPub are based on ActivityStreams[4], a standardized JSON-LD[5] based format used to represent social activities and interactions in a structured manner. The core concepts defined in this vocabulary include:

- Activities: Actions that may happen, are happening, or have already occurred. Common activity types include Create, Block, Delete, Like, and Update.
- Objects: The specific entities involved in an activity, such as Article, Note, Image, Video, or Place.
- Actors: The entities performing activities, such as Person, Organization, or Application.

In ActivityPub, all objects (including actors and activities) have a unique global identifier (URI).

The following sections present Modelverse and ModelFed, our federated architecture for collaborative modeling, inspired by the decentralization principles of the Fediverse and ActivityPub.

3 Running Example

As a running example, imagine a collaboration scenario aimed at creating a common model to formalize the Digital Product Passport (DPP) initiative, part

[4] `https://www.w3.org/TR/activitystreams-core/`
[5] `https://www.w3.org/TR/json-ld11/`

of the Ecodesign for Sustainable Products Regulation (ESPR) by the European Commission [10]. The development of a DPP model involves concepts and knowledge from various areas, making it an ideal case for exploring collaborative modeling among stakeholders with diverse profiles.

A DPP aims to manage, collect, and share crucial information about products throughout their lifecycle to promote product sustainability and circularity. Its goals include accelerating the transition to a circular economy, improving material and energy efficiency, extending product lifetime, and optimizing design, manufacturing, usage, and end-of-life processes [29].

According to the ESPR agreement, between 2026 and 2033, the European Union is expected to adopt the DPP for several product categories, such as textiles, batteries, electronic devices, and others. However, the design of a structural or data model for the DPP requires foundational knowledge from several sectors and user profiles, depending on the product category. Our running example focuses on designing a general DPP data model that could serve as a foundation for more specialized areas. Specifically, we identify three key areas, each covered by a distinct user profile:

- Materials: Modeling the composition of products, including raw materials, recycled materials, hazardous substances, etc. This part of the model must be lead by a materials scientist.
- Manufacturing and supply chain: Representing manufacturing processes, carbon footprint, certifications, production standards, and supply chain traceability. This knowledge can be provided by an industrial engineer with experience in manufacturing and logistics.
- Knowledge and management of DPP: Review the overall structure of the DPP data model, ensure consistency, and manage collaboration between various industry experts. This includes knowledge of the DPP regulations, data governance, and interoperability requirements. An IT expert or DPP data manager can fulfill this role, acting as a DPP model administrator.

To help these profiles efficiently collaborate together, they require a collaborative space where each can use its own tool to describe the parts of the model relevant to them. Additionally, they should only see the parts of the model that they need to see to avoid information overload and prevent unauthorized changes.

In the next sections, we will see how a Modelverse provides the right environment to support this type of complex cross-platform distributed modeling scenarios.

4 Modelverse

We call Modelverse the federated ecosystem of distributed modeling platforms interconnected through ModelFed, a protocol specifically designed for collaborative modeling.

Unlike centralized solutions, Modelverse enables decentralized collaboration by allowing multiple users to concurrently edit shared models without relying on a central server. Changes propagate in real-time across platforms, ensuring runtime synchronization while maintaining platform autonomy as each modeling tool preserves its own internal representation of the model while adhering to the federation protocol.

To illustrate this, Fig. 1 depicts our running scenario from Section 3 as a potential Modelverse configuration, where three platforms joined the Modelverse and talk to each other via the ModelFed protocol. In this scenario, users collaboratively design the model at runtime, using different notation: the *Admin* and the *Industrial engineer* work graphically, while the *Materials scientist* interacts through a tabular view. This heterogeneity is naturally supported by Modelverse, where each platform manages its own concrete syntax and user interface for model representation. Additionally, platforms can store and maintain their local representation of the complete model or specific portions, depending on the user's needs. This local storage is not dictated by the ModelFed protocol and can follow the platform's own architectural choices (e.g., SQL/NoSQL databases, in-memory caching, or external storage solutions), as long as the platform complies with ModelFed's synchronization interface.

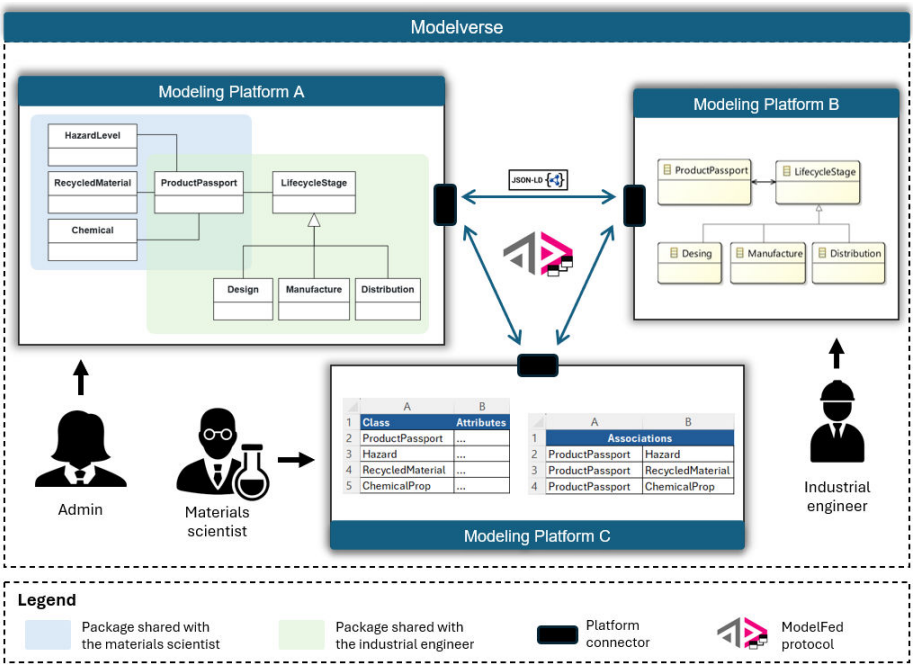


Fig. 1: Collaborative modeling in Modelverse.

The realization of a Modelverse scenario requires two key elements: (1) a standardized vocabulary to ensure semantic interoperability of modeling activities, and (2) the implementation of the ModelFed protocol based on this vocabulary. Section 5 details the vocabulary design and its modeling-specific concepts, while Section 6 describes the protocol's technical implementation.

5 ModelFed vocabulary

As illustrated in the Modelverse schema (Fig. 1), platform-to-platform communication is based on JSON-LD messages structured using the ModelFed vocabulary, an extension of the ActivityStreams vocabulary[6].

The ModelFed vocabulary enables semantic interoperability by supporting: (1) the representation of structural model elements (e.g., classes, attributes, relationships), (2) modeling-specific activities (e.g., Create, Update, Clone), and (3) access control through actor and grant definitions.

Without loss of generality (see Section 9, our vocabulary focuses on defining a JSON-LD syntax for structural conceptual models, drawing inspiration from the UML specification [21], as it is widely adopted in modeling practice.

Fig. 2 illustrates the core concepts and relationships of the ModelFed Vocabulary, emphasizing its extension points from the ActivityStreams foundation[7]. At its core, we find the *Object* concept, defined as an entity of any kind. Every *Object*, identified by a globally unique URI, falls into one of four main categories (*ModelElements*, *Activities*, *Actors*, and *Grants*) discussed below.

5.1 Model Elements

ModelElement which inherits from Object, is an abstract concept that serves as a general representation of any element within a model. The concepts inherited from *ModelElement* provide a pragmatic representation of the UML specification for structural models. These include fundamental constructs like Classes, Enumerations, Associations, and Generalizations, which represent the corresponding types of concepts (classes or entity types, associations or relationship types, taxonomies, etc.). For example, the Listing 1.1 specifies the JSON structure to define the class named *ProductPassport* in Fig 1, with a string attribute called *name*. Note that every object includes an *id* for global identification and a *type* for object classification. A similar JSON document would be used to describe other elements of our DPP example, following the vocabulary of Fig. 2

Listing 1.1: Example of a Class definition based on ModelFed

```
{
  "@context": [
    "https://www.w3.org/ns/activitystreams",
    "https://besser-pearl.github.io/ModelFed/ns/modelfed.jsonld"
  ],
```

[6] https://www.w3.org/TR/activitystreams-vocabulary/

[7] ModelFed vocabulary specification available at `https://modelfed.readthedocs.io`

```
  "type": "Class",
  "id": "https://platformA.com/classes/product_passport",
  "name": "ProductPassport",
  "attributes": [
    {
       "type": "Property",
       "id": "https://platformA.com/properties/name",
       "name": "name",
       "elementType": "str"
    }
  ]
}
```

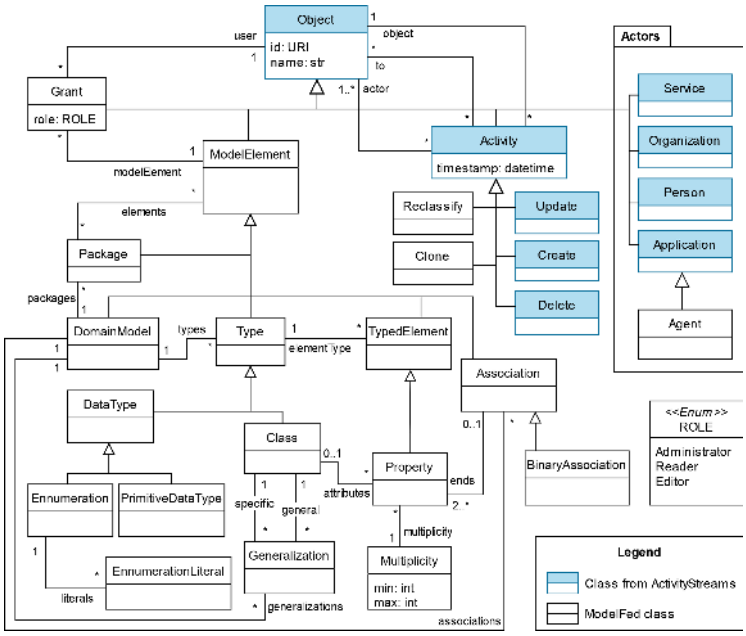


Fig. 2: The ModelFed vocabulary as an ActivityStreams extension.

5.2 Activities

Activities are among the most important concepts in Modelverse, as they allow the specification of actions performed on any *Object*. An *Activity*, serialized as a JSON-LD object, constitutes the primary payload exchanged during interactions between platforms.

ModelFed defines or reuses five activities: Create, Update, Delete, Reclassify (to change the classification of a *ModelElement*), and Clone (to create an identical copy of a *ModelElement*). Each *Activity* links one *Object* (on which the action is performed), one or more *Actors* who execute the action, and a list of *Actors* to federate the activity (relation named *to* in the diagram).

The JSON structure in Listing 1.2 represents an activity to create an attribute (*Property* object) named *UID* for the class *ProductPassport*, identified by the URI in the *owner* field. This action is performed by the *Admin* actor and is federated with the *Materials scientist* and the *Industrial engineer*. Finally, the *timestamp* field indicates the date and time of creation, which should be automatically calculated by the actor platform server.

Listing 1.2: Example of a *Create* activity based on ModelFed

```
{
  "@context": [
    "https://www.w3.org/ns/activitystreams",
    "https://besser-pearl.github.io/ModelFed/ns/modelfed.jsonld"
  ],
  "type": "Create",
  "id": "http://platformA.com/activities/a1c2t3",
  "actor": "https://platformA.com/admin/",
  "to": [
    "http://platformB.com/materialsscientist/",
    "http://platformC.com/industrialeng/"
  ],
  "object": {
    "type": "Property",
    "id": "https://platformA.com/properties/uid",
    "name": "UID",
    "elementType": "str",
    "owner": "https://platformA.com/classes/product_passport"
  },
  "timestamp": "2025-02-11T15:42:45Z"
}
```

5.3 Actors and Grants

Actors represent persons, organizations, services or applications with the capacity to perform *Activities* in a model. An *Agent* is a specialized type of *Application* that represents automated entities that can autonomously perform activities. Each modeling platform in Modelverse must manage and guarantee the minimal information required for actors by the ModelFed protocol: *id*, *type*, *name*, *inbox*, and *outbox*. In particular, the *inbox* and *outbox* are communication endpoints for the activity distribution. Federation is implemented through direct actor-to-actor interactions using these endpoints: *inbox* receives activities shared by other actors, while *outbox* publishes activities to be shared with others. For example, when the *Material scientist* creates the *Hazard* class, the activity is first sent to the *Material scientist's outbox*, and then delivered to the *Admin's inbox*.

The *Grant* concept governs access control and permissions. It specifies which actors are authorized to perform certain types of activity on specific model elements. As depicted in Fig. 2, a *Grant* specifies the *Actor* involved (*user* relationship), a *ModelElement*, and a role. Three role types are defined: *Administrator* with full control (including grant creation), *Editor* with read/write access (excluding deletion), and *Reader* with read-only access.

For instance, in our DPP running example, the *Admin* assigns editor permissions to the *Materials scientist* and *Industrial engineer* for distinct model packages, ensuring that each user interacts only with relevant components. Any

change in the *LifecycleStage* class will be federated only between the *Admin* and the *Materials scientist*, while changes in the *ProductPassport* class will be shared with everyone, as it belongs to both packages where all users have valid *Grants*.

6 Runtime implementation of ModelFed

The integration of modeling platforms within the Modelverse is mediated by a *Platform connector* (as shown in Fig. 1), which orchestrates bidirectional synchronization of modeling activities. Fig. 3 details the connector's architecture, specifying the core components required for implementation:

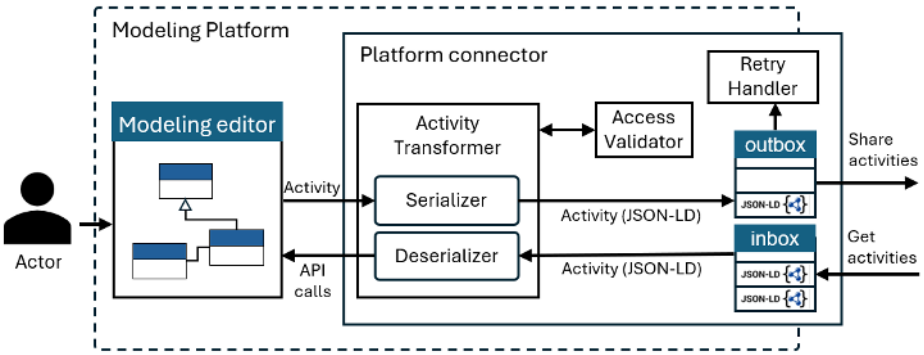


Fig. 3: Platform connector architecture

The **inbox** and **outbox** endpoints are RESTful services that handle the exchange of activities between platforms. The inbox accepts ModelFed activities in JSON-LD format, while the outbox uses the same format to queue outgoing activities for the federated nodes.

The **Activity Transformer** consists of two main components: the *Serializer*, which converts platform-specific events (e.g., "ClassCreated") into standardized ModelFed activities by mapping them to ModelFed vocabulary terms, adding metadata (such as actor and timestamp), and generating valid JSON-LD using the ModelFed @context; and the *Deserializer*, which performs the inverse operation by resolving JSON-LD contexts and translating the activities into platform-specific API calls to recreate the original actions.

The **Access Validator** performs mandatory authorization checks during both the serialization and deserialization phases. It checks the actor's *Grants* against the target model elements, applies role-based policies (e.g., editors can modify but not delete), and rejects any unauthorized activities.

Finally, to handle delivery failures, the Platform Connector can include a **Retry Handler**. This component ensures reliability by queuing failed activities and retrying delivery with an exponential backoff policy. If delivery still fails after

a set number of attempts, the system logs the error and notifies the user. This fault-tolerant approach follows established distributed system practices [24].

To illustrate the activity federation process and the orchestration of the key components involved, Fig. 4 presents a sequence diagram based on the DPP scenario from Section 3. In this scenario, the user *Admin* has already created a domain model and assigned editor permissions to the *Materials Scientist*.

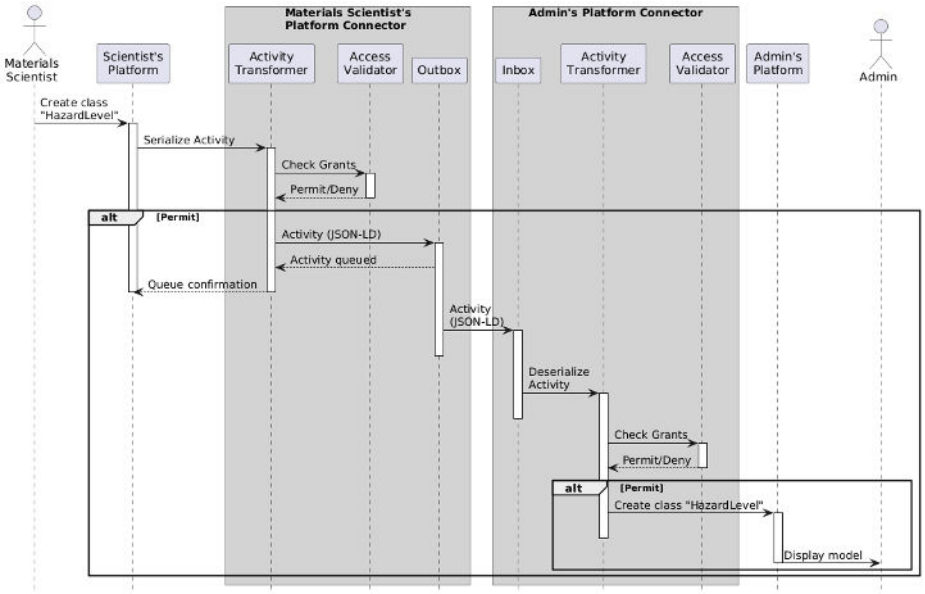


Fig. 4: Modelverse sequence diagram: *HazardLevel* class creation

The process begins when the *Materials Scientist* creates a *HazardLevel* class. This event triggers the platform to send an activity to the *Activity Transformer* for serialization. The transformer first delegates authorization to the *Access Validator*, which checks the actor's *Grants* against both the target model element (*HazardLevel* class or its owner domain model) and the requested operation (*Create*). Upon successful validation, the transformer serializes the activity in ModelFed's JSON-LD format and queues it in the *Outbox* for asynchronous distribution. This design ensures non-blocking user interaction by decoupling the *Materials Scientist* modeling tasks from the background federation subprocesses.

On the *Admin* side, once the message is received on the *Inbox*, it is sent to the *Activity Transformer* for deserialization. Before this, however, a second permission check is performed (this time on the recipient's side). This dual-authorization mechanism (enforced on both the actor and recipient servers) ensures that activities are executed only when the actor is authorized to perform them. If the activity passes this check, it is replicated on *Admin's Platform* via the corresponding API calls, allowing the Admin to view the updated model.

7 Tool Support

To facilitate the adoption of ModelFed, we provide a comprehensive set of open-source resources, including docs, vocabulary definitions, and a prototype.

The official documentation describes the protocol architecture, message formats, vocabulary structure, and implementation guidelines. It is available online[8] as a practical reference for developers and researchers.

Moreover, we have released a proof-of-concept prototype that supports and connects two modeling platforms via Python-based connectors: one for BESSER [1], a low-code modeling platform based on UML, and another for PyEcore [27], an MDE framework compatible with the EMF Java implementation. We configured web servers for each platform using FastAPI and implemented the endpoints required by the ModelFed protocol (e.g., outbox and inbox activities for users). These connectors handle inbox/outbox communication, interpret JSON-based modeling activities, reproduce them within each platform (e.g., creating or updating a class), and coordinate activity federation.

The prototype is available in a public repository[9]. The ModelFed vocabulary, which extends ActivityStreams with modeling-specific concepts, is defined using JSON-LD. The vocabulary context is publicly accessible[10] and can be extended as needed to support other modeling domains. Also, following the examples in the repo, it is easy to add support for additional platforms.

8 Validation

This section evaluates ModelFed via two experiments involving two modeling platforms. The goal is to assess the interoperability, consistency, and efficiency of the protocol. In particular, we want to address the following questions:

- RQ1 (interoperability): Does ModelFed enable an effective model federation across heterogeneous modeling platforms?
- RQ2 (consistency): Are the federated models correct and coherent across platforms after modeling collaboration?
- RQ3 (efficiency): What is the performance impact of using the federation protocol under different collaboration loads?

The evaluation uses the prototype presented in Section 7, and tests the DPP scenario discussed in Section 3. All experimental artifacts, including test scenarios, collected data, and experiment results are available[11]. For both experiments, we deployed separate web servers (BESSER or PyEcore) for each actor on a single machine with 32GB of RAM, an Intel(R) Core(TM) i7-1280P processor, and Windows 11 to simulate collaborative modeling.

[8] `https://modelfed.readthedocs.io`
[9] `https://github.com/BESSER-PEARL/ModelFed`
[10] `https://besser-pearl.github.io/ModelFed/ns/modelfed.jsonld`
[11] `https://github.com/BESSER-PEARL/ModelFed/tree/main/validation`

8.1 Experiment 1: Evaluating the consistency of ModelFed

Our first experiment consisted of recreating the DPP modeling scenario. To do so, we defined 50 modeling activities to build a simplified DPP domain model inspired by the ISO 59040 standard [14][12] and assign the activities to our three different actors: the *Admin*, who creates the domain model, packages, and grants access to other users (interacting through the BESSER platform); the *Materials Scientist*, who defines the aspects of the model related to material composition and hazardous substances (using the PyEcore platform); and the *Industrial Engineer*, who models the stages of product life (also using the PyEcore platform).

First, we used a Python script to simulate the actions of the three actors on their respective servers, with each activity performed every 100 ms.Fig. 5 presents the final federated model as viewed by the *Admin* user in the BESSER modeling platform. All 50 activities were properly processed by their corresponding servers, federated as intended, and handled by the platform connector. Specifically, 11 activities were performed by the *Admin* and federated with the other two users; 17 activities by the *Material Scientist* were federated only with the *Admin*; and 22 activities by the *Industrial Engineer* were federated with the *Admin*. We verified that the activities were federated or shared correctly by checking the outbox and inbox of each user.

After this first initial experiment, we permuted the order of the actions and repeated the test to confirm proper processing of the actions and consistency of the global model. In total, we tested eight different sequences, randomly reordering the activities while adhering to a set of basic rules to avoid logical errors. For instance, the first activity must always be the creation of the Domain Model, and an attribute cannot be created before its owning class exists. We chose eight sequences, as in our opinion, they are representative enough to cover a variety of execution orders, allowing us to discard potential consistency issues.

In these tests, we also mixed the activities of the three actors instead of keeping them grouped, simulating a more realistic scenario where users work concurrently. Finally, we analyzed the eight resulting models and confirmed that all were identical, containing the same model elements.

This experiment demonstrates that the ModelFed protocol enables interoperability (**RQ1**) and supports effective model federation (**RQ2**) between BESSER and PyEcore. Synchronization between both platforms was achieved by implementing technological connectors, and the federation of activities remained consistent after several attempts. However, as discussed in Section 6, this requires that each participant platform properly implements the protocol.

8.2 Experiment 2: Performance analysis

The objective of the second experiment was to evaluate the performance of a modeling platform under stress using the ModelFed protocol. We designed a

[12] Note the goal of the experiment is not to create an ideal DPP model, but to create one collaboratively; the model itself is just a possibility based on a standard description

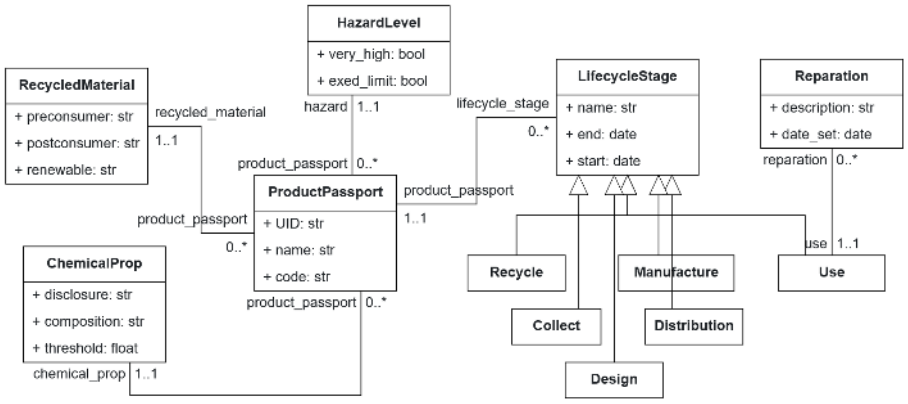


Fig. 5: Federated DPP Model in the BESSER Platform

scenario in which an admin user creates a model and executes *Create* activities for classes that are federated with multiple users. To simulate this, we ran a series of tests, gradually increasing the number of modeling activities and the number of federated users (from 5, 10, and to 20 federated users). For each user, we deployed a separate web server instance on the same machineWe used JMeterto configure the test plan and collected metrics such as processing time (the time taken to process and deliver activities), error percentage (the proportion of failed requests), and throughput (the number of successfully completed requests per second), among others.

The average processing times for different numbers of activities (ranging from 10 to 300) are presented in Fig. 6. As shown, the processing time increases with a higher number of activities and users to federate. The response time remains below 500 ms for up to 50 concurrent activities with 20 users, while for 10 and 5 users, the processing time is significantly lower. It is important to note that, in a real-world scenario, executing 50 concurrent activities may not be typical. In all three cases (5, 10, and 20 users), when the number of activities reaches 250, some errors occurred, and some activities were not processed due to server overload (the server was unable to handle the volume of requests). This explains why the response time decreases at this point (failed requests are handled more quickly). Again, unless the participants are AI Agents, this volume of modeling activities is not realistic, so we do not see this as an issue.

Based on these results, we argue that the response time remains acceptable, and therefore our approach is scalable, answering **RQ3**.

9 Extending ModelFed

For the sake of simplicity, until now we have focused on the use of ModelFed to implement collaborative modeling on structural conceptual models. However,

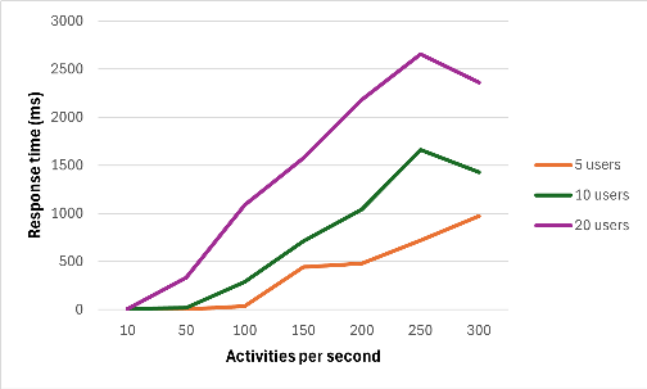


Fig. 6: Response time vs. number of activities varying federated user load

ModelFed can be used to collaborate on all types of models by just defining the corresponding vocabulary as described in Section 5.

As an example, we define a minimal extension to support the specification of object models [21], which can be used to model relevant scenarios of the system, expressed as instances of classes defined in the structural model. Fig. 7 models this extension by introducing two new concepts: *Instance*, which represents instances of a class (e.g., instances of the *ProductPassport* class), and *AttributeLink*, which represents instances of a class attributes (such as *UID* or *name*). Note that in this case, the extension is created on top of the vocabulary we created so far, but we could create a brand new vocabulary if needed, e.g. to represent state machines or any other type of dynamic model.

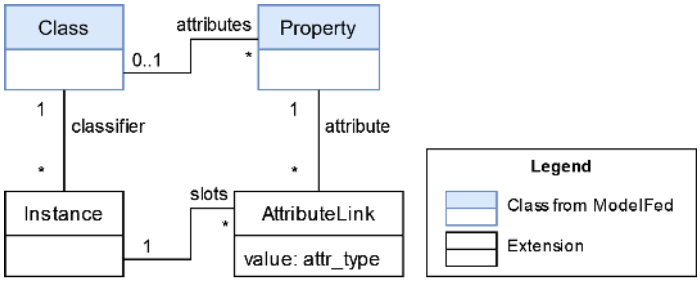


Fig. 7: ModelFed extension to enable object specification

Once this extension is modeled with JSON-LD, following the approach used in ActivityStreams (see Section *ModelElements*) we can automatically collaborate and exchange partial specifications of object models across the actors of our system. An example is shown in Listing 1.3, where the object model is an instance of our DPP running example. Note that the context in line 4 corresponds to the

new vocabulary extension, and it must be available and accessible to guarantee interoperability and automatic validation. This is key to understanding how to correctly map the concepts. The rest of the listing (lines 6–20) defines an *Instance* of the *ProductPassport* class with slots (attribute values) being *name* and *UID*, with values "Battery_AAA" and "A1B2C3", respectively.

Listing 1.3: Example of object definition extending ModelFed

```
{
    "@context": [
      "https://www.modelverse.com/ns/modelverse",
      "https://your-domain.org/ns/object_modeling.jsonld"
    ],
    "type": "Instance",
    "id": "http://platformA.com/productpassport/12345",
    "classifier": "https://platformA.com/classes/product_passport",
    "slots": [
      {
        "type": "AttributeLink",
        "attribute": "https://platformA.com/properties/name",
        "value": "Battery_AAA"
      },
      {
        "type": "AttributeLink",
        "attribute": "https://platformA.com/properties/uid",
        "value": "A1B2C3"
      }
    ]
}
```

It is worth highlighting that, since the JSON-LD format ensures that properties without a defined context are ignored, existing platforms in the Modelverse can continue to operate without errors, even if they do not support the new modeling extensions. This facilitates cross-platform collaboration when not all the tools connected in the Modelverse understand all types of models.

10 Related work

Many approaches have addressed collaborative modeling, though mostly focusing on a single preexisting framework or platform. Some are based on EMF (e.g., Sirius [9], MONDO [6], and Collaboro [15]), others rely on MPS (e.g., Modelix [20]), or other standalone solutions (e.g., WebGME [26], FlexiSketch [30], and MetaEdit [18]). Despite their benefits, they cannot be used as part of cross-platform interoperability scenarios without ad-hoc extensions.

A few approaches focus on the synchronization and federation of models across different platforms. Nevertheless, their main goal is data consistency, transformation and exchange among the platforms, more than the support for interactive collaboration features. Examples of tools in this category are NAOMI [8], which enables definition of model dependencies for change propagation, Reactive Links [23] , which synchronizes primitive properties through explicit linkages, or OpenFlexo [13] , which provides a dedicated language to define model mappings. Beyond the lack of real-time modeling, they lack support for essential collaborative features (such as fine-grained access control) and rely on centralized architectures for model integration.

Closer to ModelFed goals, Parsafix [28], DesignSpace [7], and BUMBLE-CE [2] also target cross-platform collaborative modeling. Parsafix relies on a central component that acts as an intermediary to synchronize model changes between two specific platforms. There is no support to add additional platforms. DesignSpace is also centralized, storing models as graphs that can be modified via tool adapters. Although DesignSpace offers a mechanism to control access by marking a model as public or private, it is not possible to share specific parts of a model with specific users. Similarly, BUMBLE-CEalso adopts a centralized architecture, where models are maintained in a central inventory, and modeling activities are coordinated through an inventory controller using platform-specific plugins. Also, similar to previous tools, BUMBLE-CE lacks support for fine-grained model sharing.

Finally, we remark that ModelFed has taken inspiration from several ActivityPub extensions developed to support federation in other domains. For example, BookWyrm [4] for social reading communities, Pixelfed [22] for photo sharing networks, and ForgeFed [11] to federate software project hosting activities across software forge platforms. These extensions demonstrate the protocol's extensibility, but they focus on domains outside of modeling.

To sum up, and to the best of our knowledge, our approach is the first to propose a federated architecture that enables collaborative cross-platform modeling without relying on a central server for model synchronization.

11 Conclusion

We presented ModelFed, a federated protocol that enables real-time collaboration across heterogeneous modeling platforms. To demonstrate its feasibility, we developed a prototype integrating two distinct modeling platforms and validated the whole approach in a collaborative modeling scenario, showing that ModelFed can synchronize activities both consistently and efficiently.

As further work, we plan to extend both the ModelFed protocol and its implementation. Regarding the protocol, our goal is to add more sophisticated traceability and conflict resolution mechanisms beyond the current timestamp-based ordering, e.g. via Conflict-free Replicated Data Type (CRDT). Existing work based on the use of Natural Language Processing (NLP) techniques to suggest semantically similar model concepts across different models [3,16] could also be useful here. Implementation efforts will focus on three directions: expanding support for other platforms (platform connectors), enhancing reliability through delivery failure handling and cryptographic signature verification, and developing an AI-assisted connector to automate JSON-LD serialization, validation, and consistency checking. This intelligent component will reduce the integration barrier for new modeling platforms joining Modelverse.

Acknowledgments. This work is supported by the Luxembourg National Research Fund (FNR) PEARL program, grant agreement 16544475.

References

1. Alfonso, I., Conrardy, A., Sulejmani, A., Nirumand, A., Ul Haq, F., Gomez-Vazquez, M., Sottet, J.S., Cabot, J.: Building besser: an open-source low-code platform. In: International Conference on Business Process Modeling, Development and Support. pp. 203–212. Springer (2024)
2. Aslam, K., Chen, Y., Butt, M., Malavolta, I.: Cross-platform real-time collaborative modeling: An architecture and a prototype implementation via emf. cloud. IEEE Access **11**, 49241–49260 (2023)
3. Aydemir, F.B., Dalpiaz, F.: Supporting collaborative modeling via natural language processing. In: Conceptual Modeling: 39th International Conference, ER 2020, Vienna, Austria, November 3–6, 2020, Proceedings 39. pp. 223–238. Springer (2020)
4. BookWyrm Project: Bookwyrm. `https://bookwyrm.social/` (2025), accessed on August 13, 2025
5. Busse, S., Kutsche, R.D., Leser, U., Weber, H.: Federated information systems: Concepts, terminology and architectures. Forschungsberichte des Fachbereichs Informatik **99**(9), 1–38 (1999)
6. Debreceni, C., Bergmann, G., Búr, M., Ráth, I., Varró, D.: The mondo collaboration framework: secure collaborative modeling over existing version control systems. In: Proceedings of the 2017 11th joint meeting on foundations of software engineering. pp. 984–988 (2017)
7. Demuth, A., Riedl-Ehrenleitner, M., Nöhrer, A., Hehenberger, P., Zeman, K., Egyed, A.: Designspace: an infrastructure for multi-user/multi-tool engineering. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing. pp. 1486–1491 (2015)
8. Denton, T., Jones, E., Srinivasan, S., Owens, K., Buskens, R.W.: Naomi–an experimental platform for multi–modeling. In: Model Driven Engineering Languages and Systems: 11th International Conference, MoDELS 2008, Toulouse, France, September 28-October 3, 2008. Proceedings 11. pp. 143–157. Springer (2008)
9. Eclipse Foundation: Eclipse sirius. `https://eclipse.dev/sirius` (2025), accessed on August 13, 2025
10. European Commission: Proposal for an ecodesign for sustainable products regulation. `https://environment.ec.europa.eu/publications/proposal-ecodesign-sustainable-products-regulation_en` (2022), accessed on April 03, 2025
11. ForgeFed Project: Forgefed. `https://forgefed.org/` (2025), accessed on August 13, 2025
12. Franzago, M., Di Ruscio, D., Malavolta, I., Muccini, H.: Collaborative model-driven software engineering: a classification framework and a research map. IEEE Transactions on Software Engineering **44**(12), 1146–1175 (2017)
13. Golra, F.R., Beugnard, A., Dagnat, F., Guerin, S., Guychard, C.: Addressing modularity for heterogeneous multi-model systems using model federation. In: Companion Proceedings of the 15th International Conference on Modularity. pp. 206–211 (2016)
14. International Organization for Standardization: ISO 59040:2025, Circular economy — Product circularity data sheet. `https://www.iso.org/standard/82339.html` (2025), iSO Standard
15. Izquierdo, J.L.C., Cabot, J.: Collaboro: a collaborative (meta) modeling tool. PeerJ Computer Science **2**, e84 (2016)

16. Köse, S.G., Aydemir, F.B.: Automated glossary extraction from collaborative requirements models. In: 2021 IEEE 29th International Requirements Engineering Conference Workshops (REW). pp. 11–15. IEEE (2021)
17. La Cava, L., Greco, S., Tagarelli, A.: Understanding the growth of the fediverse through the lens of mastodon. Applied network science **6**, 1–35 (2021)
18. MetaCase: Metaedit+. `https://www.metacase.com/products.html` (2025), accessed on August 13, 2025
19. Mistrík, I., Grundy, J., Van der Hoek, A., Whitehead, J.: Collaborative software engineering: challenges and prospects. Springer (2010)
20. Modelix Project: Modelix. `https://modelix.org` (2025), accessed on August 13, 2025
21. Object Management Group: OMG Unified Modeling Language (UML) — version 2.5.1. `https://www.omg.org/spec/UML/2.5.1` (2017), oMG Specification
22. PixelFed Project: Pixelfed. `https://pixelfed.org/` (2025), accessed on August 13, 2025
23. Raţiu, C.C., Assunção, W.K., Herac, E., Haas, R., Lauwerys, C., Egyed, A.: Using reactive links to propagate changes across engineering models. Software and Systems Modeling pp. 1–27 (2024)
24. Shahid, M.A., Islam, N., Alam, M.M., Mazliham, M., Musa, S.: Towards resilient method: An exhaustive survey of fault tolerance methods in the cloud computing environment. Computer Science Review **40**, 100398 (2021)
25. Sheth, A.P., Larson, J.A.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Computing Surveys (CSUR) **22**(3), 183–236 (1990)
26. Vanderbilt University: Webgme. `https://webgme.org` (2025), accessed on August 13, 2025
27. Vincent Aranega: PyEcore. `https://github.com/pyecore/pyecore` (2025), accessed on August 13, 2025
28. Voogd, S.N., Aslam, K., Van Gool, L., Theelen, B., Malavolta, I.: Real-time collaborative modeling across language workbenches–a case on jetbrains mps and eclipse spoofax. In: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). pp. 16–26. IEEE (2021)
29. Walden, J., Steinbrecher, A., Marinkovic, M.: Digital product passports as enabler of the circular economy. Chemie Ingenieur Technik **93**(11), 1717–1727 (2021)
30. Wüest, D., Seyff, N., Glinz, M.: Flexisketch team: Collaborative sketching and notation creation on the fly. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. vol. 2, pp. 685–688. IEEE (2015)