PAMoLa Models as First-Class Citizens of Vitruvius V-SUMs

Monalisha Ojha
University of Mannheim
Mannheim, Germany
monalisha.ojha@uni-mannheim.de

Mohammad Sadeghi
University of Mannheim
Mannheim, Germany
mohammad.sadeghi@uni-mannheim.de

Shilpi Gupta
University of Mannheim
Mannheim, Germany
shilpi.gupta@uni-mannheim.de

Colin Atkinson
University of Mannheim
Mannheim, Germany
colin.atkinson@uni-mannheim.de

Abstract

Vitruvius manages heterogeneous EMF models as participants in a Virtual Single Underlying Model (V-SUM) and preserves their consistency through change-driven Reactions. However, due to its EMF basis, it represents each participant using a strict two-level classification arrangement, places rules outside the managed content, and binds consistency specifications to domain-specific Ecore types. This paper investigates whether PAMoLa, a self-describing and level-agnostic modeling substrate, can mitigate these limitations without requiring changes to Vitruvius. The key is the PAMoLa Ecore Metamodel (PEM), a single, fixed Ecore metamodel whose instances represent PAMoLa domain classifiers, instances and values as ordinary model content. Once the PEM is registered as a Vitruvius domain, every PAMoLa model becomes an ordinary XMI participant, irrespective of its domain or ontological depth. Using a published brake-system case study, we examine five model-management benefits that follow from this arrangement and state their costs explicitly. The clearest benefit concerns the reuse of consistency logic. Inspection of the source reveals 24 nearly identical forward routines that mirror brake attributes into CAD parameters. We recast this family as one structural reaction pattern accompanied by 24 declarative feature bindings, and implement the pattern for integer-valued attributes. Finally, a cross-case artifact analysis and comparisons with Commonalities, EMF Views, and Openflexo federation delimit its applicability.

CCS Concepts

• **Software and its engineering → Model-driven software engineering**.

Keywords

Model management, View-based development, Consistency, Vitruvius, V-SUM, PAMoLa, Multi-level modeling

ACM Reference Format:
Monalisha Ojha, Mohammad Sadeghi, Shilpi Gupta, and Colin Atkinson. 2026. PAMoLa Models as First-Class Citizens of Vitruvius V-SUMs. In *ACM/IEEE 29th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion 2026), October 04–09, 2026, Málaga, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3837062.3841579

This work is licensed under a Creative Commons Attribution 4.0 International License.
MODELS Companion 2026, Málaga, Spain
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2903-4/26/10
https://doi.org/10.1145/3837062.3841579

1 Introduction

Model management is concerned not with a single model in isolation, but with the models and mappings through which a system is described [4]. In most engineering projects, such things as requirements, designs, code, and verification artifacts usually overlap, although each remains shaped by its own language and tool. Multi-view modeling must therefore not only provide suitable views, but also preserve the relationships between them as the underlying models evolve [8].

Vitruvius addresses this problem by treating the collection of participating artifacts as a Virtual Single Underlying Model (V-SUM) [19], where the elements in different participants are related through a correspondence model, and change-driven specifications written in a "Reactions language" propagate edits between them. Since the framework is based on the Eclipse Modeling Framework (EMF), a V-SUM also inherits a uniform reflective API, generic change recording, XMI serialization, and mature tooling [26]. These properties make it a useful setting in which to explore whether a different choice of participant metamodel can broaden the content managed by the framework beyond what a conventional domain-specific Ecore metamodel permits.

The question is important because the typical EMF/Vitruvius arrangement has five structural limitations that follow from the fact that every participant model is represented using a strict two-level classification arrangement.

(L1) One classification boundary. An Ecore metamodel and its instances are separate artifacts with different lifecycles. A V-SUM therefore has no natural participant in which domain metatypes, types, instances, and instances of instances coexist as editable content. Deeper classification must either be encoded as ordinary attributes or distributed across several metamodel/model pairs.

(L2) Virtual rather than intrinsic unity. The V-SUM is intentionally a federation of resources with correspondences connecting their elements. As a result, participants do not have one common domain-level identity and naming regime, so when two fragments are meant to form one typed model, the correspondence layer must artificially recreate a relation that is already intrinsic to that model.

(L3) Rules outside the managed content. Well-formedness and domain constraints normally reside in OCL files, validation plug-ins, tests, or transformation code. Although these artifacts can be versioned alongside a model, they are not themselves first-class model content governed by the same identities, packaging, and projection mechanisms.

(L4) A metamodel for every participant vocabulary. Every new kind of participant requires a new Ecore metamodel to be registered and associated tooling. Moreover, when its vocabulary evolves, generated code and serialized instances usually have to evolve with

it. Although the framework itself is generic, the schemas through which it is used remain domain-specific.

(L5) Reactions tied to those domain types. A reaction trigger names concrete EClasses and features. This provides excellent static guidance, but it also makes structurally similar consistency logic specific to each metamodel. For example, in the brake-system artifact from the case study, the forward relation contains 31 reaction declarations. Twenty-four of these implement the same parameter-mirroring pattern for different source features, while the remaining declarations either create structure or propagate the component identifier in a different way [28].

PAMoLa offers a different point at which to draw the fixed boundary. Its protected core provides a small linguistic vocabulary, whereas domain vocabularies, ontological classification, values, profiles, and rules remain ordinary model content [25]. The *PAMoLa Ecore Metamodel* (PEM) realizes this fixed core vocabulary (i.e., linguistic metamodel) as one Ecore metamodel. Consequently, any PAMoLa model can be persisted as an XMI resource conforming to the PEM and can be admitted to a V-SUM through the standard Vitruvius domain mechanism. The framework continues to see ordinary EMF objects and changes, but the domain meaning that a bespoke metamodel would encode in EClasses is carried by the PAMoLa objects themselves.

In this paper we ask *which model-management benefits arise when PAMoLa models become first-class V-SUM participants, and what costs accompany those benefits.* Our answer consists of five related results. A PAMoLa participant can contain multi-level content (L1), while clades and persistent identities permit logically connected fragments to remain modular (L2). Rules can be managed as model content and evaluated by PAMoLa's internal Datalog engine (L3), while the PEM remains unchanged when a domain vocabulary evolves (L4). Finally, reactions can be written against the shared structural vocabulary and parameterized by declarative domain bindings (L5).

The paper's contribution is deliberately bounded. We demonstrate the PEM integration and the structural reaction pattern on a published brake-system example that predates this work [17]. We also inspect additional Vitruvius case studies to determine whether comparable feature-level repetition recurs, although this source-level analysis should not be mistaken for a runtime scalability evaluation. The public prototype implements the generic integer-value path for selected brake-disk attributes. The complete 24-binding formulation extends the same mechanism to the remaining datatypes and component classes, but we do not claim that this extension has already been implemented.

The paper starts by characterizing the two technologies and then discusses the consequences of combining them. Sections 2–4 introduce Vitruvius, PAMoLa, the PEM, and the brake-system artifacts. Sections 5–9 then address the five limitations in turn. Section 10 broadens the evidence, while Sections 11 and 12 position the result and delimit its applicability.

2 Background

2.1 Model Management with Vitruvius

Vitruvius realizes view-based development around a V-SUM containing participant models, a correspondence model, and consistency preservation specifications [19]. Each kind of participant is first "registered" through its metamodel. View types then select and shape the content needed by stakeholders, while committed changes are recorded and propagated to the participants they affect.

The change model follows EMF's notification structure. A commit can therefore contain atomic events such as an EObject creation, insertion into a containment feature, replacement of an attribute value, or assignment of a reference. Reactions consume those events and take the appropriate steps required to maintain consistency. *Reactions* declare the triggers, while *routines* retrieve corresponding objects, create targets, and update values. A conventional trigger is deliberately domain-specific, for example, "after `minimumThicknessInMM` is replaced on a `BrakeDisk`." The generated accessors and metamodel types give the author strong static support, but they also bind the specification to that metamodel.

Vitruvius Commonalities provides a higher-level alternative for recurring bidirectional relations. A commonality declares a concept shared by several metamodels and generates Reactions from the declared participations [18]. Section 9.1 compares this mechanism with the PEM-based approach once it has been introduced.

2.2 PAMoLa (Poly-aligned Modeling Language)

PAMoLa is a self-describing modeling substrate whose first-class elements are called *ordia* (singular *ordium*) [25]. Figure 1 shows the fragment of the PAMoLa (linguistic) metamodel needed to read the examples in this paper. The brake model introduced below shows how classifiers (e.g., BrakeDisk) and instances (e.g., disk1) are both linguistically Entity ordia. The domain distinction between classifiers and instances is thus captured by ordinary model content rather than by instantiating a new domain-specific EClass, because disk1 is able to declare BrakeDisk as its ontological type within a model.

Every ordium, except the root ordium, has one *custodian*. The ordia directly custodied by it are its *wards*, and an ordium together with all transitive wards forms a *clade*. This custodian tree gives the otherwise general graph a locality for naming, loading, serialization, projection, execution, and transactions. A fully qualified name (FQN) is derived from the current custodian path, whereas a PAMoLa identifier (PID) remains stable when an ordium is renamed or moved. FQNs therefore provide unique, readable addresses in one model state, while PIDs provide unique lifetime identity.

The inheritance structure in Figure 1 shows how the core remains small without reducing every connection to an anonymous edge. Since Entity and Relationship specialize Clabject, they inherit the `hasDeclaredType` and `hasPotency` properties and can therefore participate in ontological classification hierarchies. Likewise, Mediation provides the `hasRelatum` reference through which its concrete subclasses connect clabjects in multi-participant relationships. These subclasses are Participation, representing a relationship end, and Intercalation, representing a participant in a categorisation (cf. a generalisation set). A Categorisation groups its intercalations

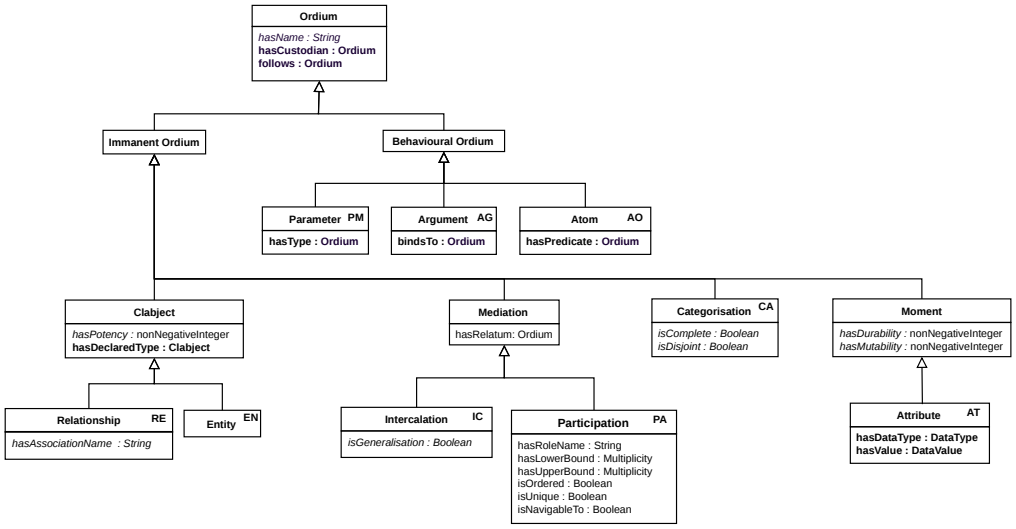

Figure 1: Selected fragment of the PAMoLa Metamodel. The generalization structure exposes how the categories used by the paper inherit their properties, while the labels in the upper-right corners give the HUTN badges of concrete categories.

and records the completeness and disjointness of the generalisation structure as a whole.

Beyond the fragment shown in Figure 1, the metamodel defines other feature and behavioral categories, not all of which are used in this paper. An Attribute carries a datatype and, where appropriate, a value, while a Marker attaches profile vocabulary without changing the linguistic type of the marked ordium. The behavioral branch contains Operation and Action, together with Atom, Parameter, and Argument. A rule can therefore be represented as a structured clade whose predicates, variables, and term occurrences have the same identities and custodianship as the domain content it governs.

PAMoLa's Human-Usable Textual Notation (HUTN) offers a family of textual projections over PAMoLa models [24]. The listings below use the FHAS form of HUTN, which means *Full, Hierarchical, Abridged, Symbolic* HUTN. A two-letter badge designates the linguistic type, nesting implies custodianship, strings are quoted, and unquoted symbols denote ordium references resolved in the current context. Properties that can be recovered from the badge, hierarchy, context, or profile defaults may be omitted. The structural examples use EN for Entity, AT for Attribute, RE for Relationship, PA for Participation, CA for Categorisation, IC for Intercalation, and MK for Marker. The reified rule example additionally uses AO, AG, and PM for Atom, Argument, and Parameter.

Listing 1 can therefore be read without learning a separate domain language. Line 1 starts an Entity named BrakeDomain. The nested Entities on lines 2 and 8 are its O0 and O1 wards. The unquoted reference `O0.ComponentKind` on line 19 identifies another ordium, whereas names such as "BrakeDisk" are quoted strings. In other words, the listing is a textual projection of objects that still conform to the fixed core structure shown in Figure 1.

Linguistic classification and domain classification remain orthogonal throughout this representation. The metamodel records that an ordium is an Entity, Attribute, or Atom. The Level-Agnostic Modeling Language (LML) profile gives `hasDeclaredType` and `hasPotency` their ontological interpretation. At the storage level, an absent potency fact and an explicitly stored zero are distinct states. Under the current LML profile, however, absence has an effective potency of zero. Potency then constrains the remaining depth of direct ontological instantiation, while specialization is represented separately. This distinction is what allows the same PEM objects to represent metatypes, types, and instances without turning their ontological levels into different technical schemas.

3 Encoding PAMoLa in Ecore

The integration is possible because PAMoLa and EMF make compatible structural choices at the point where ownership is represented. A PAMoLa graph contains a distinguished custodian tree together with arbitrary semantic references. An EMF resource likewise contains one containment tree together with non-containment references. PAMoLa custodianship can therefore be represented by Ecore containment, while declared types, relata, datatypes, markers, and other semantic links can be represented by ordinary EReferences or by kernel-resolved reference values. The two technologies do not thereby acquire identical semantics. The structural correspondence nevertheless allows PAMoLa content to inhabit the EMF object model without requiring a change to Vitruvius.

3.1 The PAMoLa Ecore Metamodel

The PEM contains one EClass for each core category and EStructuralFeatures for the corresponding core properties. It therefore fixes the *linguistic* vocabulary once, while every domain classifier remains an EObject in a model. Adding BrakeFluidReservoir, a new power type, or another classification level changes the participant content rather than `pem.ecore`. Since the PEM is valid Ecore, it can be registered as an ordinary Vitruvius domain and its instances can

be serialized as ordinary XMI resources. The `pem.ecore` used here is available in the artifact repository [23].

The mapping preserves distinctions that a simplistic graph encoding would lose. For example, optional scalar properties are represented so that stored absence remains distinguishable from an explicit value. The EKC can then apply profile semantics, such as effective potency zero, without forcing the storage layer to pretend that a fact was explicitly present. Similarly, relationships and their participations remain first-class EObjects rather than becoming anonymous EReferences, because their identity and end properties are part of the PAMoLa model.

The resulting arrangement shifts the schema boundary. Vitruvius sees one stable domain, namely the PEM. The reaction author sees the instances of Entity, Attribute, and the other structural categories, together with domain information carried by names, declared types, markers, and values. This is why a structural trigger can be reused across domains, but must perform data-level guards that a bespoke EClass would otherwise provide statically.

3.2 The Ecore Kernel Core

The PEM defines shape, not the whole PAMoLa behavior. The *Ecore Kernel Core* (EKC) implements custody, identity, naming, protected-Core loading, and model operations over EMF. PAMoLa currently has two independent Kernel cores of equal standing: the Java Kernel Core and the EKC. They use different host representations, but canonical shared HUTN artifacts must be emitted byte-identically. This cross-kernel gate provides evidence that the loaded core and notation rules, rather than one implementation's classes, define the common semantics [25].

The present stack also executes rules internally. A stratified, safety-checked, semi-naive Datalog evaluator reads the reified Atom/Parameter/Argument content through a normalized fact view and computes a fixpoint directly over the loaded clade. No export to an external NEMO process is required. The normalization boundary converts kernel-specific stored values into typed terms before the rules see them, which is particularly important because the Java and Ecore kernels use different value representations [25].

For the V-SUM integration, the PEM is the element that Vitruvius consumes because the framework registers Ecore metamodels and manages EMF resources. The EKC is nevertheless required whenever the participant must behave as PAMoLa rather than merely share its data structure. In particular, it resolves identities and references, applies the LML interpretation of classification, and evaluates the managed rules discussed in Section 7.

3.3 Two Viewpoints on One Participant

A PAMoLa participant therefore presents two compatible viewpoints. Vitruvius sees EObjects and recorded changes. Creating a domain entity produces an Entity object, an insertion into a containment feature, assignments of names and types, and possibly several Attribute objects. Correspondences to those objects remain ordinary Vitruvius correspondences.

The reaction author additionally interprets the model-level data. A conventional trigger can name the `minimumThicknessInMM` feature of `brakesystem::BrakeDisk`. A PEM-side trigger instead observes a value change on a PAMoLa Attribute. It then asks whether the changed attribute is identified by the relevant binding and whether its custodian is an instance of the required declared type. The first formulation distributes domain knowledge between the Ecore metamodel and the reaction. The second keeps the Ecore metamodel fixed and moves the domain-specific part into the participant and its binding model.

4 The Brake-System Case Study

We use the brake-system case study [17]. Its repository contains the Ecore models (Figure 2), XMI instances, Vitruvius Reactions, view types, and tests [28]. Because those artifacts were designed before and independently of this PAMoLa integration, the repetitive structure discussed later was not constructed to favor our approach.

The engineering model contains a Brakesystem and an abstract BrakeComponent specialized by ABSSensor, BrakeCaliper, BrakeHose, BrakeDisk, and BrakePad. The CAD model contains namespaces and string, numeric, or Boolean parameters. The forward consistency relation maps engineering properties such as a disk's minimum thickness, pin count, and ventilation flag to named CAD parameters with appropriate datatypes and units.

Figure 3 compares the two-level EMF representation with an HUTN (FHAS) rendering of corresponding PAMoLa content. The comparison establishes a deliberately narrow point: before adding a higher-order component kind, it shows that the classifier and instance fragments can both be represented with the same structural vocabulary.

The case also contains a useful hint of a deeper domain. Both Brakesystem and BrakeComponent carry `specificationType : EString`. Such a string can label a kind, but it cannot itself own attributes, participate in classification, or be checked as a classifier. Section 5 uses a higher-order ComponentKind to turn this latent classification dimension into explicit model content.

5 Multi-Level Content in One Participant

This section addresses L1. In Figure 3, BrakeDisk and disk1 are separated into two clades only to mirror the familiar Ecore/model split. PAMoLa does not require that split. The two clades can become wards of one BrakeDomain ordium, and the resulting model remains one PEM instance and one XMI resource. The declared-type link is then an ordinary internal reference rather than a metamodel-conformance relation outside the model.

The benefit becomes clearer when a third level is introduced. Listing 1 adds ComponentKind above the component classes. Lines 2–6 declare this O0 classifier and the properties by which component kinds can themselves be described. Lines 8–24 place BrakeComponent and BrakeDisk at O1. The declared-type reference on line 19 classifies BrakeDisk by ComponentKind, while lines 20–23 supply the kind-level values that distinguish this particular component category. Lines 26–33 then add the concrete disk at O2. O0, O1, and O2 make the ontological strata visible, but they are ordinary organizational wards rather than hard metamodel layers.

The example combines ontological classification and specialization without conflating them. ComponentKind has potency two because two direct classification steps may still follow. BrakeDisk consumes one of those steps when it declares ComponentKind as its type, and disk1 consumes the next when it declares BrakeDisk.

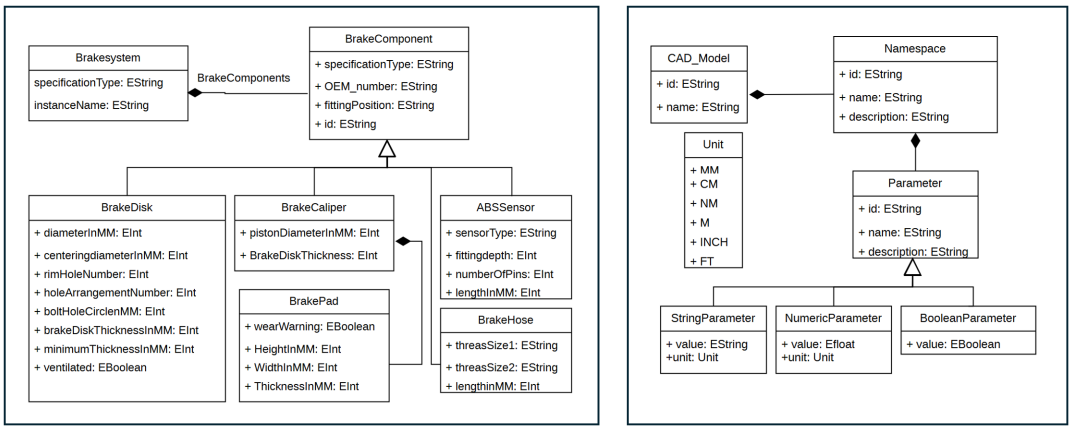

Figure 2: Abridged fragments of two case-study metamodels: `brakesystem.ecore` (left) and `cad.ecore` (right).

EMF artifacts	PAMoLa content in HUTN (FHAS)

`<!-- brakesystem.ecore -->`
`<eClassifiers xsi:type="ecore:EClass"`
`  name="BrakeComponent" abstract="true">`
`  <eStructuralFeatures xsi:type="ecore:EAttribute"`
`  name="OEM_number" eType="...EString"/>`
`</eClassifiers>`
`<eClassifiers xsi:type="ecore:EClass" name="BrakeDisk"`
`  eSuperTypes="#//BrakeComponent">`
`  <eStructuralFeatures xsi:type="ecore:EAttribute"`
`  name="minimumThicknessInMM" eType="...EInt"/>`
`</eClassifiers>`
	`EN {hasName "BrakeTypes",`
	`  EN {hasName "BrakeComponent", hasPotency 0,`
	`  AT {hasName "OEM_number", hasDataType String},`
	`  CA {hasName "BrakeComponentKinds", isComplete true, isDisjoint true,`
	`  IC {hasName "general", isGeneralisation true, hasRelatum BrakeComponent},`
	`  IC {hasName "disk", hasRelatum BrakeDisk}`
	`  }`
	`  },`
	`  EN {hasName "BrakeDisk", hasPotency 1,`
	`  AT {hasName "minimumThicknessInMM", hasDataType Integer}`
	`  }`
	`}`

`<!-- brakesystem.xml -->`
`<brakesystem:Brakesystem>`
`  <BrakeComponents xsi:type="brakesystem:BrakeDisk"`
`  OEM_number="VW123456" id="newId"`
`  diameterInMM="120"`
`  brakeDiskThicknessInMM="30"`
`  minimumThicknessInMM="25"`
`  ventilated="true"/>`
`</brakesystem:Brakesystem>`
	`EN {hasName "BrakeInstances",`
	`  EN {hasName "disk1", hasPotency 0, hasDeclaredType BrakeTypes.BrakeDisk,`
	`  AT {hasName "OEM_number", hasDataType String, hasValue "VW123456"},`
	`  AT {hasName "diameterInMM", hasDataType Integer, hasValue 120},`
	`  AT {hasName "brakeDiskThicknessInMM", hasDataType Integer, hasValue 30},`
	`  AT {hasName "minimumThicknessInMM", hasDataType Integer, hasValue 25}`
	`  }`
	`}`

Figure 3: The same classifier and instance information represented through an Ecore metamodel and XMI instance on the left, and as two PAMoLa clades conforming to the fixed PEM on the right.

The LML profile therefore checks the direct sequence 2–1–0 with one potency-step rule.

By contrast, the categorisation on lines 11–16 says that BrakeDisk specializes BrakeComponent. Specialization neither consumes potency nor requires the general and special classifiers to have the same potency. Thus BrakeComponent may remain closed to direct instances at potency zero, while its specialized BrakeDisk category may classify concrete disks. This separation is important because the higher-order ComponentKind does not replace the ordinary brake-component inheritance hierarchy. It adds a classification dimension over that hierarchy.

The same example also explains why power-type information is more than a string-valued tag. In the original Ecore model, `specificationType` can name a category, but the value cannot own the properties safetyRelevant and inspectionIntervalKm, classify component classes, or participate in rules as a classifier. Once ComponentKind is explicit model content, those operations use the same identities and structural predicates as the rest of the model.

This flexibility has an authoring cost. A Reactions trigger sees PEM classes, not the domain classifier BrakeDisk. Listing 2 shows the schematic form. The trigger in lines 1–4 is reusable because every new domain entity enters the same containment feature. The routine must then recover the domain role from declared-type data in lines 6–9. A complete library encapsulates that traversal, but it cannot make the additional test disappear.

Thus the PEM removes the architectural boundary, but not the need to express domain intent. The remainder of the paper asks whether expressing that intent once against a shared structure can repay the extra data-level navigation.

```
1  EN {hasName "BrakeDomain",
2    EN {hasName "O0",
3      EN {hasName "ComponentKind", hasPotency 2,
4        AT {hasName "safetyRelevant", hasDataType Boolean},
5        AT {hasName "inspectionIntervalKm", hasDataType Integer}
6      }
7    },
8    EN {hasName "O1",
9      EN {hasName "BrakeComponent", hasPotency 0,
10       AT {hasName "OEM_number", hasDataType String},
11       CA {hasName "BrakeComponentKinds", isComplete true,
12           isDisjoint true,
13         IC {hasName "general", isGeneralisation true,
14             hasRelatum BrakeComponent},
15         IC {hasName "disk", hasRelatum BrakeDisk}
16       }
17     },
18     EN {hasName "BrakeDisk", hasPotency 1,
19         hasDeclaredType O0.ComponentKind,
20       AT {hasName "safetyRelevant", hasDataType Boolean,
21           hasValue true},
22       AT {hasName "inspectionIntervalKm", hasDataType Integer,
23           hasValue 30000}
24     }
25   },
26   EN {hasName "O2",
27     EN {hasName "disk1", hasPotency 0,
28         hasDeclaredType O1.BrakeDisk,
29       AT {hasName "brakeDiskThicknessInMM",
30           hasDataType Integer, hasValue 30},
31       AT {hasName "minimumThicknessInMM",
32           hasDataType Integer, hasValue 25}
33     }
34   }
35 }
```

Listing 1: Three ontological levels represented as one HUTN (FHAS) clade and one PEM participant.

```
1 reaction EntityAdded {
2   after element inserted in pamola::Ordium[wards]
3   call handleEntity(affectedEObject)
4 }
5 routine handleEntity(pamola::Ordium x) {
6   if (!(x instanceof pamola::Entity)) return
7   if (!isInstanceOf(x, BrakeDomain.O1.BrakeDisk)) return
8   // perform the BrakeDisk-specific consistency action
9 }
```

Listing 2: Schematic structural reaction trigger with a domain-level declared-type guard.

6 Logical Unity by Construction

This section addresses L2. Keeping every level in one resource is possible, but it is not required and will often be undesirable. A company-wide classifier library may evolve slowly, while vehicle configurations are owned by different teams and updated daily. PAMoLa therefore treats a clade as a semantic and operational unit for loading, serialization, projection, transaction scoping, and rule execution. Splitting a model at a clade boundary changes how the content is stored and governed, but it does not introduce a new ontological boundary. The classifier and instance clades can consequently remain separate resources while participating in one typed model.

Listing 3 makes the relation explicit without relying on textual nesting. The flat rendering writes the relevant `hasCustodian` links directly. Lines 2–3 place the classifier and its attribute in the O1 clade, while lines 6–9 place the instance, its value-bearing attribute, and a relationship participation in O2. The `hasDeclaredType` reference on line 6 and the `hasRelatum` reference on line 9 cross the resource boundary through the same symbolic identity space. They are model facts rather than correspondences inferred by Vitruvius.

The distinction between FQNs and PIDs is important here. An FQN is a readable address derived from the current custodian path and therefore changes after a rename or re-custody. A PID remains stable. References can consequently be written through current names while durable traces and correspondences retain lifetime identity. The precise XMI representation of a cross-resource link, for example an EMF cross-resource EReference or a kernel-resolved symbolic value, is an implementation choice. Our semantic claim is narrower: PAMoLa already contains the type reference and identity regime before Vitruvius adds any relation between the resources.

This arrangement also gives rule execution a natural scope. A rule can be evaluated over one clade and the explicitly imported ordia on which that clade depends, rather than over every PAMoLa resource in a repository. Modularity therefore serves both organizational ownership and bounded processing. The paper does not claim that all lazy loading or distributed-clade mechanisms are already integrated with Vitruvius, but the clade boundary provides the unit around which such mechanisms can be added.

Vitruvius correspondences remain necessary when two independently authored representations must be related, such as a brake disk and its CAD namespace. They are not needed merely to restate that disk1 is classified by BrakeDisk. This distinction gives the V-SUM a more disciplined division of labor. PAMoLa references express relations internal to one logical PAMoLa model. Vitruvius correspondences then concentrate on relations between that participant and genuinely different views.

7 Rules as Managed Content

This section addresses L3. The shipped brake models propagate values but do not state whether those values form a valid brake specification. For example, a disk's minimum wear-limit thickness should be lower than its nominal thickness. In an ordinary EMF project, this condition would commonly reside in OCL, a validator, or a test. PAMoLa can instead store it in the model as rule content.

The distinction is stronger than attaching a Datalog string to an EObject. A reified rule has a place in the custodian tree, a PID and FQN, and explicit references to the predicates and model elements it uses. It can therefore be packaged with a domain clade, compared through ordinary model differencing, projected into textual syntax, and queried as data. Diagnostic results can likewise retain references to the ordia that caused them, which makes the rule and its witnesses part of the managed state rather than an external validation report.

Listing 4 shows two textual projections. Lines 2–8 report a disk for which the wear limit is not strictly below the nominal thickness. The use of >= is deliberate: equality already violates the required strict margin. Lines 9–13 report a disk with no valued nominal-thickness slot. Both rules return an explicit witness rather than a detached Boolean.

The textual clauses are projections of reified model content. In the stored representation, an Atom identifies a predicate, Argument wards bind positions to constants or shared Parameter ordia, and

```
// O1 classifier fragment (resource A)
EN {hasName "BrakeDisk", hasCustodian BrakeDomain.O1, hasPotency 1}
AT {hasName "minimumThicknessInMM", hasCustodian BrakeDomain.O1.BrakeDisk, hasDurability 0, hasMutability 0, hasDataType Ordium.Core.
    DataType.Integer}

// O2 instance fragment (resource B)
EN {hasName "disk1", hasCustodian BrakeDomain.O2, hasPotency 0, hasDeclaredType BrakeDomain.O1.BrakeDisk}
AT {hasName "minimumThicknessInMM", hasCustodian BrakeDomain.O2.disk1, hasDurability 0, hasMutability 0, hasDataType Ordium.Core.DataType.
    Integer, hasValue 25}
RE {hasName "bc1", hasCustodian BrakeDomain.O2}
PA {hasName "component", hasCustodian BrakeDomain.O2.bc1, hasRelatum BrakeDomain.O2.disk1, hasLowerBound 0, hasUpperBound unlimited,
    isOrdered false, isUnique true, isNavigableTo true}
```

Listing 3: Flat HUTN (FLUS) excerpt for separately loadable classifier and instance clades, with custody, typing, values, and relata as explicit symbolic references.

```
% wear limit must be strictly below nominal thickness
violation(?D, thicknessMargin) :-
    hasDeclaredType(?D, BrakeDisk),
    hasCustodian(?A, ?D), hasName(?A, "minimumThicknessInMM"),
    hasValue(?A, ?Min),
    hasCustodian(?B, ?D), hasName(?B, "brakeDiskThicknessInMM"),
    hasValue(?B, ?Nom), ?Min >= ?Nom.

violation(?D, missingNominalThickness) :-
    hasDeclaredType(?D, BrakeDisk),
    not hasSlot(?D, "brakeDiskThicknessInMM").
hasSlot(?X, ?N) :- hasCustodian(?A, ?X),
                   hasName(?A, ?N), hasValue(?A, _).
```

Listing 4: A domain rule and a mandatory-slot rule in the internal Datalog syntax.

```
EN {hasName "rule",
  AO {hasName "thicknessMargin",
      hasPredicate violation,
    AG {hasName "headObject", bindsTo D},
    AG {hasName "headKind", bindsTo thicknessMargin},
    AO {hasName "body1", hasPredicate hasDeclaredType,
      AG {hasName "object", bindsTo D},
      AG {hasName "type", bindsTo BrakeDomain.O1.BrakeDisk}
    },
    PM {hasName "D", hasType Ordium}
  }
}
```

Listing 5: Abridged HUTN (FHAS) projection of the rule head and first body atom.

the custodian and `follows` relations recover clause structure and order. Listing 5 shows only the head and first body atom of the margin rule. Its purpose is not to define the complete rule metamodel, but to show that the predicate and variable occurrences have ordium identities.

The current internal engine evaluates this content directly. It checks rule safety, normalizes stored values into typed terms, and performs stratified semi-naive fixpoint evaluation [25]. The broader behavioral branch also contains Operation and Action, which represent callable behavior and execution requests in the same space. Consequently, the rule can be named, diffed, projected, and reasoned about as model content, while its diagnostic results retain references to the ordia that caused them.

Rules do not replace Reactions. Reactions propagate a committed change between participants, whereas rules check or derive consequences over the resulting model state. A practical integration can therefore evaluate selected rule clades after propagation settles

```
% correspondsTo is projected from Vitruvius correspondences
violation(?I, missingCounterpart) :-
    hasDeclaredType(?I, PCM.OperationInterface),
    not correspondsTo(?I, _).
```

Listing 6: PCM/UML counterpart invariant as managed PAMoLa rule content.

and on demand during review. Since the rules range over structural predicates and declared-type data, the same rule language reaches O0, O1, and O2 without separate validators for each technical level.

This placement is not peculiar to the brake system. The published Vitruvius case-study suite contains cross-view invariants in its tests, including the requirement that an operation interface have a counterpart in the other view [29]. Listing 6 expresses that requirement schematically as a rule that can be stored with the participant it governs.

The predicate `correspondsTo` is not a PAMoLa core predicate. It denotes a fact view that an integration ware can project from the Vitruvius correspondence model. Written in this form, the invariant is named and versioned with the model, ranges over a declared domain role rather than a generated metaclass, and can produce a witness during editing or review. The example therefore connects the cross-case observation to the central model-management claim. The issue is not that the projects lack invariants, but that those invariants otherwise remain beside the code that tests them rather than in the managed content itself.

8 Schema-Stable Participation

This section addresses L4. A bespoke participant metamodel hardens the domain vocabulary into an artifact that the runtime, generated code, editors, and serialized instances depend on. The PEM hardens only the stable structural vocabulary. Domain evolution therefore becomes model evolution unless the core itself changes.

Table 1 does not claim that migration disappears. Existing content may no longer satisfy an evolved domain vocabulary, and rules must report the affected elements. The difference is that the parser and storage schema continue to accept the model, so migration becomes visible model-management work rather than a failure to load an obsolete XMI shape.

The same arrangement provides editor continuity and reduces the number of domain-specific schemas that must be maintained. An editor generated once from the PEM can inspect any PAMoLa participant. A traceability model, supplier catalog, or requirements graph represented in PAMoLa can therefore enter the same V-SUM

Table 1: Representative evolution operations and where their cost is paid.

Operation	Bespoke Ecore participant	PEM participant
Add BrakeFluidReservoir	edit Ecore; regenerate and re-register; migrate instances; review rules	add an O1 Entity
Add `wearIndicator`	edit Ecore; regenerate; migrate; add consistency logic	add an Attribute and, where needed, a binding
Add O0 component kinds	introduce a separate metamodel or encode kinds as data	add an O0 classifier clade
Add the thickness rule	add OCL, validation code, or a test release	add a rule clade

without another metamodel registration. This benefit becomes more important as the number of participant vocabularies and their rate of change increase. In the brake example, adding a mapped feature no longer requires a new EStructuralFeature merely so that Vitruvius can observe it. The new feature is an Attribute occurrence, and the domain-specific mapping is one binding.

This schema stability is obtained by moving some checks out of generated infrastructure. A bespoke metamodel makes required features and slot types available to EMF's static and generated code. Under the PEM, those conditions are explicit profile or domain rules. Runtime validation therefore replaces part of the compile-time discipline, and reaction authors receive less domain-specific completion and refactoring support. Whether the trade is favorable depends on the volatility, classification depth, and governance of the participant vocabulary rather than on a general preference for one representation.

9 Generic, Reusable Reaction Libraries

This section addresses L5. The brake-system case study reaction file contains 31 forward reactions [28]. Six create the CAD structure. Twenty-five respond to attribute replacements, but one of those, `IdChanged`, assigns the namespace identifier directly. The remaining 24 implement one recurring parameter-mirroring algorithm: retrieve the corresponding namespace, find or create a CAD parameter, assign its display name and unit, and copy the source value. Listing 7 shows one member of that family.

In Listing 7, lines 1–5 bind the trigger to one EClass feature, and lines 7–17 implement a routine that differs from its siblings mainly in the source feature, CAD display name, parameter class, and unit. The mapping information still has to exist in any generic solution. The relevant reduction is therefore not from 24 facts to one fact, but from 24 imperative copies of an algorithm to one algorithm plus 24 declarative feature bindings.

A binding must identify a classifier-qualified feature rather than a local name, because names such as `lengthInMM` occur in more than one component class. The parameter class can usually be derived from the PAMoLa datatype. Display names and units remain

```
1  reaction MinimumThicknessChanged {
2    after attribute replaced at
3          brakesystem::BrakeDisk[minimumThicknessInMM]
4    call updateMinimumThickness(affectedEObject)
5  }
6  routine updateMinimumThickness(brakesystem::BrakeDisk disk) {
7    match { val ns = retrieve cad::Namespace corresponding to disk
         }
8    update {
9      var p = ns.parameters.findFirst[
10         q | q.name == "Minimum Thickness"] as NumericParameter
11     if (p === null) {
12       p = new cad::NumericParameter
13       p.name = "Minimum Thickness"; p.unit = Unit.MM
14       ns.parameters.add(p)
15     }
16     p.setValue(disk.minimumThicknessInMM)
17   }
18 }
```

Listing 7: One parameter-mirroring abridged reaction from the case study.

Table 2: Examples from the 24 feature-FQN bindings recovered from the forward relation.

Source feature	CAD name / class	Unit
`OEM_number`	OEM Number / String	–
`specificationType`	Specification Type / String	–
`diameterInMM`	Diameter / Numeric	MM
`minimumThicknessInMM`	Minimum Thickness / Numeric	MM
`numberOfPins`	Number of Pins / Numeric	COUNT
`ventilated`	Ventilated / Boolean	–
24 classifier-qualified feature bindings in total		

genuine domain knowledge, as the acronym in "OEM Number" and the unit COUNT illustrate.

Listing 8 gives the binding-driven generalization. Lines 1–4 provide the one structural trigger. Lines 6–13 recover the custodian and corresponding namespace, check the domain role, and resolve a binding by feature FQN. Lines 14–24 apply the common find-or-create-and-copy algorithm. The public prototype implements this architecture for integer-valued attribute occurrences and tests selected BrakeDisk features. Supporting all 24 bindings additionally requires datatype dispatch for string and Boolean values and the complete binding model shown here.

This reformulation has three consequences. First, the 24 mappings become inspectable model content that can itself be validated for missing or duplicate bindings. Second, the algorithm can be reused for any PEM participant that follows the same parameter-mirroring policy. Third, adding `wearIndicator` requires a content edit and one binding rather than another copy of the routine. The cost is that a routine over Attribute communicates less immediate domain intent and receives less static IDE support than one over BrakeDisk. Declared-type guards, generated binding accessors, and naming discipline may mitigate that loss, but the degree to which they restore comprehension remains an empirical question.

The comparison does not depend on the claim that ordinary Vitruvius is incapable of reflective generic code. A reaction could in principle drop to the EMF reflective API, inspect an EObject's features, or introduce a manually designed common supertype and metadata table. Such a solution would recreate the abstraction

```
reaction AttributeValueChanged {
  after attribute replaced at pamola::Attribute[value]
  call mirrorAttribute(affectedEObject)
}
routine mirrorAttribute(pamola::Attribute a) {
  match {
    val owner = a.eContainer as pamola::Entity
    val ns = retrieve cad::Namespace corresponding to owner
  }
  update {
    if (!isComponentInstance(owner)) return
    val featureId = featureFqn(owner, a)
    val b = findBinding(featureId)
    if (b === null) return
    var p = ns.parameters.findFirst[q | q.name == b.cadName]
    if (p === null) {
      p = newParameter(a.dataType)
      p.name = b.cadName
      setUnit(p, b.cadUnit)
      ns.parameters.add(p)
    }
    setTypedValue(p, a.value)
  }
}
```

Listing 8: Binding-driven generalization of the implemented structural reaction pattern.

needed by the generic routine. The PEM contribution is that this abstraction is already the standard participant structure, so the same Attribute, datatype, declared-type, and identity vocabulary is available across every PAMoLa domain. The 24-to-one result should therefore be read as a reduction achieved by this shared structural vocabulary, not as an impossibility theorem about the Reactions language.

9.1 Relation to Commonalities

Commonalities already addresses repeated hand-written Reactions. It declares a shared concept and its participations in known metamodels, then generates bidirectional consistency logic [18]. For a fixed set of established Ecore metamodels, it is therefore often more convenient than the one-way routine in Listing 8, and it retains the static metamodel vocabulary in the specification.

The PEM changes a different layer. A commonality still names domain metaclasses, whereas a PEM participant presents one structural metamodel and keeps domain roles as declared-type data. Adding another bespoke domain therefore requires another participation in the commonality, while adding another PAMoLa domain can reuse the same structural trigger and supply new bindings as content. Conversely, Commonalities is preferable when preserving strongly typed autonomous metamodels is the primary objective. The two approaches are compatible: a Commonalities specification can treat the PEM as one ordinary Vitruvius domain and generate bidirectional relations around it.

10 Cross-Case Artifact Analysis

The brake case demonstrates feasibility, but one small case study cannot show whether the underlying repetition is unusual. We therefore inspected the large consistency specifications and tests in the official Vitruvius case-study suite as available in the official repository [29]. For each relation, we counted reaction families whose triggers and routines have the same control structure and differ primarily in the named feature or role. We then counted the consistency conditions each project states about corresponding elements, such as that a UML package and the PCM repository it corresponds to must carry the same name, or that every interface on one side must have a counterpart on the other. These conditions are held neither in the models nor in the reactions: each project collects them in a dedicated validation class in its test suite, written as JUnit assertions. Table 3 reports both counts against the size of each relation.

Table 3: Analysis of the published Vitruvius case-study artifacts [29].

Consistency relation	Reactions	Repetitive family	Test assertions
Brake system → CAD	31	24	0
PCM ↔ UML	126	24	21
UML ↔ Java	112	7	75
PCM ↔ Java	59	15	0

The result supports a limited generalization. Repetitive families recur, and the largest observed family again consists of 24 variants. Each of them satisfies the precondition the reformulation of Section 9 depends on, a stable control algorithm varying only in the feature or role named, and each is therefore a candidate for the same reduction to one structural reaction whose per-feature knowledge is held as bindings. Only the brake family has been prototyped, so for the remaining relations this is a projection from the fixed structure of the PEM rather than a measured outcome. Not every consistency rule collapses in this way. Creating one PCM repository component may require a UML package, class, and constructor. A binding can describe that structure, but it cannot turn three semantically different target creations into one value copy. The reusable opportunity is therefore strongest where the control algorithm is stable and only the feature role varies.

The test assertions reveal a second pattern. The PCM/UML and UML/Java projects state many cross-view conditions, but the conditions live in JUnit rather than in either model. Listing 6 works through one representative PCM/UML condition, while Table 3 shows that the placement issue recurs. The listing is schematic rather than a port of the case study, but it demonstrates how a ware-projected correspondence fact can be consumed by a rule that is itself managed as PAMoLa content.

This section is not a performance study. The counts characterize source artifacts, not propagation time, memory use, or fixpoint cost. Runtime scaling with participant count, binding count, model size, and rule volume remains future work. The analysis instead answers the narrower threat that the brake-system repetition may have been an isolated or deliberately selected anomaly.

11 Related Work

The closest work differs principally in where it locates shared meaning. Generic model management seeks reusable operators over models and mappings [4], while multi-view research studies how heterogeneous views are selected, synchronized, and governed [5, 8]. Classifications of Single Underlying Model approaches and formal accounts of multi-model commonalities distinguish explicit underlying models, correspondences among autonomous models, and virtual constructions over them [21, 27]. Vitruvius occupies

the correspondence-based part of this space [19]. Our question is what changes when one family of participants adopts a shared, level-uniform structural metamodel within that architecture.

EMF Views and Openflexo are particularly relevant because they preserve autonomous source technologies. EMF Views constructs non-materialized views over heterogeneous models and has been extended across storage technologies and toward security-compliant virtual views [6, 7, 20]. Openflexo/FML coordinates autonomous models through virtual models, concepts, and links, with applications to modularity, continuous requirements engineering, and traceability between informal and formal specifications [3, 12–14]. Amrani et al. place these products within the wider family of federative approaches for model management in MBSE [1]. These approaches are attractive when existing technological spaces must remain authoritative and loosely coupled. The PEM addresses the complementary case in which several vocabularies can accept one protected structural substrate. PAMoLa models may still be split into clades and resources, but their classifiers, rules, and identity-bearing references already share one internal vocabulary before federation with non-PAMoLa views.

Openflexo/FML has also been applied to the multi-level process challenge [16]. It assembles flexible levels over federated models through concepts and type-object structures, whereas PAMoLa makes clabjects and potency intrinsic to the participant. Both expose more than two domain levels, but Openflexo preserves the source technologies through a federative interpretation layer, while the PEM allows domains to share one linguistic layer.

Vitruvius Commonalities are a third point. They declares shared concepts over known metamodels and generates bidirectional Reactions [18], thereby retaining their static vocabulary and tooling. Each new metamodel must, however, participate explicitly in the commonality. The PEM supplies the common structural vocabulary before a particular relation is written, so further PAMoLa domains can reuse the same trigger and contribute bindings. Section 9.1 therefore treats the mechanisms as complementary.

PAMoLa's level-uniform classification follows the orthogonal-classification tradition [2]. Multi-level modeling research has developed its rationale, patterns, and application boundaries [9–11], while the power-type pattern used in Section 5 predates those systems [15, 22]. Our contribution is to allow such content to reside in a two-level V-SUM through one fixed Ecore metamodel.

12 Discussion and Limitations

The five benefits follow from one inversion. A bespoke participant places domain vocabulary in the Ecore schema and obtains static tooling, whereas the PEM places it in model content and obtains schema stability and level uniformity. Adoption therefore depends on four questions: how often the vocabulary changes, whether the organization controls its migration schedule, whether several ontological levels must remain editable, and whether rules and mappings should travel with the participant. The case strengthens as more answers are positive. Conversely, a stable standard metamodel with strong generated tooling and no deeper classification requirement is a poor candidate for conversion merely to gain uniformity.

The same inversion explains the costs and the present evidence boundary. Requiredness, type conformance, and some navigation checks move from generated Ecore code into profile or domain rules. Structural Reactions require declared-type and binding resolution, while generic routines may obscure intent and introduce binding content that must itself be governed. The public prototype implements this architecture for integer-valued attribute occurrences and tests selected BrakeDisk features; its PEM domain registration, PAMoLa models, reactions, and tests are available in the artifact repository [23]. The proof of concept registers the PEM as a Vitruvius domain and exercises a structural integer-value reaction on the brake example. The full 24-binding library, complete datatype dispatch, bidirectional co-editing, and PEM ports of the other relations remain to be implemented. These costs must therefore be compared with the actual regeneration and migration burden of the bespoke alternative rather than assumed away on either side.

Performance and tooling also remain open. Relevant measurements include declared-type and binding resolution, internal Datalog fixpoint cost, propagation latency as participants grow, and memory consumption for carrier ordia. Generated views and accessors may recover some static support lost when reactions operate over Attribute rather than BrakeDisk, but comprehension and maintenance require user evaluation.

Finally, PIDs and FQNs give PAMoLa fragments intrinsic identity, but independent non-PAMoLa views still require correspondences and synchronization policies. The approach therefore does not eliminate federation. It gives one participant family stronger internal unity so that Vitruvius can concentrate on genuinely cross-representational relations.

13 Conclusion

This paper has shown that PAMoLa and Vitruvius can be combined without changing either technology. The PEM realizes PAMoLa's fixed linguistic vocabulary as one Ecore metamodel, so every PAMoLa model is an ordinary EMF resource and can become a first-class V-SUM participant. The EKC supplies the identity, classification, and rule semantics that the metamodel alone cannot provide. This changes where the cost of model-management effort is paid. Multi-level classifiers and instances can coexist as modular clades, rules can be stored and executed as model content, and domain vocabulary can evolve without changing the registered metamodel. Consistency logic can also be written against one structural vocabulary and parameterized by declarative bindings. In the brake-system case, 24 parameter-mirroring routines become one algorithm plus 24 feature mappings.

The result is complementary to virtual views, federation, and Commonalities rather than a replacement for them. Where autonomous technological spaces must remain authoritative, those mechanisms remain essential. Where a shared soft substrate is acceptable, PAMoLa can move volatile classifiers, rules, and mappings out of hard schemas and into the managed model. The next step is therefore to implement the complete binding-driven library bidirectionally, measure runtime and authoring costs against bespoke Reactions and Commonalities, and test several independently evolving PAMoLa participants in one V-SUM. Such an evaluation can determine whether schema stability, shared rule libraries, and intrinsic identity repay the structural indirection they introduce.

Acknowledgments

This work was funded by the German Research Foundation (DFG) – CRC 1608 – 501798263.

References

[1] Moussa Amrani, Rakshit Mittal, Miguel Goulão, Vasco Amaral, Sylvain Guérin, Salvador Martínez, Dominique Blouin, Anish Bhobe, and Yara Hallak. 2024. A Survey of Federative Approaches for Model Management in MBSE. In *Companion Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. ACM, 990–999. doi:10.1145/3652620.3688221

[2] Colin Atkinson and Thomas Kühne. 2001. The Essence of Multilevel Metamodeling. In *Proceedings of the 4th International Conference on the Unified Modeling Language (UML) (Lecture Notes in Computer Science, Vol. 2185)*. Springer, 19–33. doi:10.1007/3-540-45441-1_3

[3] Jean-Christophe Bach, Antoine Beugnard, Joël Champeau, Fabien Dagnat, Sylvain Guérin, and Salvador Martínez. 2024. 10 Years of Model Federation with Openflexo: Challenges and Lessons Learned. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. ACM, 25–36. doi:10.1145/3640310.3674084

[4] Philip A. Bernstein. 2003. Applying Model Management to Classical Meta Data Problems. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR)*. 209–220.

[5] Hugo Brunelière, Erik Burger, Jordi Cabot, and Manuel Wimmer. 2019. A Feature-Based Survey of Model View Approaches. *Software and Systems Modeling* 18, 3 (2019), 1931–1952. doi:10.1007/s10270-017-0622-9

[6] Hugo Brunelière, Jokin Garcia Perez, Manuel Wimmer, and Jordi Cabot. 2015. EMF Views: A View Mechanism for Integrating Heterogeneous Models. In *Conceptual Modeling (ER 2015) (Lecture Notes in Computer Science, Vol. 9381)*. Springer, 317–325. doi:10.1007/978-3-319-25264-3_23

[7] Hugo Brunelière, Florent Marchand de Kerchove, Gwendal Daniel, Sina Madani, Dimitris S. Kolovos, and Jordi Cabot. 2020. Scalable Model Views over Heterogeneous Modeling Technologies and Resources. *Software and Systems Modeling* 19 (2020), 827–851. doi:10.1007/s10270-020-00794-6

[8] Antonio Cicchetti, Federico Ciccozzi, and Alfonso Pierantonio. 2019. Multi-View Approaches for Software and System Modelling: A Systematic Literature Review. *Software and Systems Modeling* 18, 6 (2019), 3207–3233. doi:10.1007/s10270-018-00713-w

[9] Juan de Lara, Esther Guerra, and Jesús Sánchez Cuadrado. 2014. When and How to Use Multilevel Modelling. *ACM Transactions on Software Engineering and Methodology* 24, 2 (2014), 12:1–12:46. doi:10.1145/2685615

[10] Claudenir M. Fonseca, João Paulo A. Almeida, Giancarlo Guizzardi, and Victorio A. Carvalho. 2021. Multi-Level Conceptual Modeling: Theory, Language and Application. *Data & Knowledge Engineering* 134 (2021), 101894. doi:10.1016/j.datak.2021.101894

[11] Ulrich Frank. 2022. Multi-Level Modeling: Cornerstones of a Rationale. *Software and Systems Modeling* 21 (2022), 451–480. doi:10.1007/s10270-021-00958-8

[12] Fahad R. Golra, Antoine Beugnard, Fabien Dagnat, Sylvain Guérin, and Christophe Guychard. 2016. Addressing Modularity for Heterogeneous Multi-Model Systems Using Model Federation. In *Companion Proceedings of the 15th International Conference on Modularity*. ACM, 206–211. doi:10.1145/2892664.2892701

[13] Fahad R. Golra, Antoine Beugnard, Fabien Dagnat, Sylvain Guérin, and Christophe Guychard. 2016. Continuous Requirements Engineering Using Model Federation. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*. IEEE, 347–352. doi:10.1109/RE.2016.42

[14] Fahad R. Golra, Fabien Dagnat, Jeanine Souquières, Imen Sayar, and Sylvain Guérin. 2018. Bridging the Gap Between Informal Requirements and Formal Specifications Using Model Federation. In *Software Engineering and Formal Methods (SEFM 2018) (Lecture Notes in Computer Science, Vol. 10886)*. Springer, 54–69. doi:10.1007/978-3-319-92970-5_4

[15] César González-Pérez and Brian Henderson-Sellers. 2006. A Powertype-Based Metamodelling Framework. *Software and Systems Modeling* 5, 1 (2006), 72–90. doi:10.1007/s10270-005-0099-9

[16] Sylvain Guérin, Joël Champeau, Jean-Christophe Bach, Antoine Beugnard, Fabien Dagnat, and Salvador Martínez. 2022. Multi-Level Modeling with Openflexo/FML: A Contribution to the Multi-Level Process Challenge. *Enterprise Modelling and Information Systems Architectures* 17, 9 (2022), 1–21. doi:10.18417/emisa.17.9

[17] Nathan Hagel, Johannes Makelburg, Claus Hammann, Thomas Weber, Thomas Alexander Volk, Francesco P. Urbano, Patrick Grycz, Katharina Bause, Minakshi Kaushik, Vincenzo Scotti, Akhila Bairy, Maike Schwammberger, Maribel Acosta, Albert Albers, Anne Koziolek, and Tobias Düser. 2025. Explainability in Automated Cross-Domain Model-Driven Brake System Development. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. IEEE, 204–211. doi:10.1109/ASEW67777.2025.00047

[18] Heiko Klare and Joshua Gleitze. 2019. Commonalities for Preserving Consistency of Multiple Models. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 371–378. doi:10.1109/MODELS-C.2019.00058

[19] Heiko Klare, Max E. Kramer, Michael Langhammer, Dominik Werle, Erik Burger, and Ralf H. Reussner. 2021. Enabling Consistency in View-Based System Development—The Vitruvius Approach. *Journal of Systems and Software* 171 (2021), 110815. doi:10.1016/j.jss.2020.110815

[20] Salvador Martínez, Alexis Fouche, Sébastien Gérard, and Jordi Cabot. 2018. Automatic Generation of Security Compliant (Virtual) Model Views. In *Conceptual Modeling (ER 2018) (Lecture Notes in Computer Science, Vol. 11157)*. Springer, 109–117. doi:10.1007/978-3-030-00847-5_10

[21] Johannes Meier, Christopher Werner, Heiko Klare, Christian Tunjic, Uwe Aßmann, Colin Atkinson, Erik Burger, Ralf H. Reussner, and Andreas Winter. 2020. Classifying Approaches for Constructing Single Underlying Models. In *Model-Driven Engineering and Software Development*. Communications in Computer and Information Science, Vol. 1161. Springer, 350–375. doi:10.1007/978-3-030-37873-8_15

[22] James J. Odell. 1994. Power Types. *Journal of Object-Oriented Programming* 7, 2 (1994), 8–12.

[23] PAMoLa. 2026. PAMoLa-Vitruvius: Reference Implementation of PAMoLa Models as V-SUM Participants. GitHub repository. https://github.com/monalisha31/PAMoLa-Vitruvius

[24] PAMoLa Project. 2026. HUTN Guide. Project documentation. https://pamola.net/docs/HUTN_Guide

[25] PAMoLa Project. 2026. PAMoLa: Technical Overview. Project documentation. https://pamola.net/docs/info

[26] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. 2008. *EMF: Eclipse Modeling Framework* (2 ed.). Addison-Wesley.

[27] Patrick Stünkel, Harald König, Yngve Lamo, and Adrian Rutle. 2021. Comprehensive Systems: A Formal Foundation for Multi-Model Consistency Management. *Formal Aspects of Computing* 33 (2021), 1067–1114. doi:10.1007/s00165-021-00555-2

[28] Vitruv Tools. 2025. Brake System V-SUM Case Study: Models, Instances, Reactions, View Types, and Tests. GitHub repository. https://github.com/vitruv-tools/Brake-System-V-SUM-Case-Study

[29] Vitruv Tools. 2026. Case Studies for the Vitruvius Framework. GitHub repository. https://github.com/vitruv-tools/Vitruv-CaseStudies