JOT
JOURNAL OF
OBJECT TECHNOLOGY

Journal of Object Technology | **RESEARCH ARTICLE**

Supporting OWL-Style Reasoning over UML-Style Models via Endogenous OWL Overlays

Shilpi Gupta, Mohammad Sadeghi, Monalisha Ojha, and Colin Atkinson
University of Mannheim, Germany

ABSTRACT Unified Modeling Language (UML)-style class diagrams are widely used for conceptual and system modelling because they provide compact and familiar visual representations of domain structure. However, the built-in reasoning support provided by UML tools is generally modest. The Web Ontology Language (OWL), in contrast, is optimised for supporting well-defined logical inference but is less well suited to authoring compact, engineering-oriented model representations comparable to UML-style diagrams. To address these respective weaknesses, we propose endogenous OWL overlays as a lightweight bridge from an authoritative UML-style model to standards-based OWL reasoning services. An overlay is a rule-derived OWL view computed over a triple/fact-based representation of the authoritative model snapshot used by the overlay rules. The host concrete syntax is LML, a UML-style multi-level modelling notation, and the overlay targets selected OWL-style services rather than full UML coverage. Because the overlay is recomputed on demand and consumed by standard OWL tools, the source model remains the source of truth and the derived view stays directly traceable to the modelling declarations that produced it. The approach handles both the model's declared structure and selected modelling idioms that arise naturally in UML-style modelling, using a small set of local OWL axioms only where needed. Using four representative scenarios, we demonstrate subproperty closure, role-membership inference, derived associations, and the detection of logically unsatisfiable classes without maintaining a separately synchronised OWL artefact.

KEYWORDS Endogenous OWL overlays, UML-style models, LML, OWL/RDFS reasoning, Powertype-aware reasoning, Role inference.

1. Introduction

UML-style class models remain the default authoring notation for conceptual modelling in software and systems engineering because they are compact, familiar, and well supported by mainstream tooling for editing, documentation, and downstream development tasks (OMG 2017). However, the semantic services available to modellers are usually modest: beyond syntax and basic well-formedness checks, many tools do not systematically derive implied facts or expose semantic contradictions. When richer correctness questions arise, engineers may fall back to manual inspection, ad hoc scripts, or constraints expressed in languages such as the Object Constraint Language (OCL) (OMG 2014). This matters because many everyday modelling idioms carry intended semantics that modellers expect to be exploitable automatically. Examples include subsetted role ends (where a specialised association end such as *hasFather* should imply the more general *hasParent* relationship), role modelling with role classes (where role membership should be derivable from how instances participate in links), derived relations such as *hasGrandparent* (which require property-chain semantics), and powertype/partition-style classifications (which often intend disjointness and can make certain combinations of specialisation logically impossible). In standard UML-centric workflows, such implications are usually not derived as a service for the modeller; instead they must be asserted explicitly or encoded as separate constraints.

Constraint-based approaches such as OCL are useful for refining the semantics of UML models and defining precise

JOT reference format:
Shilpi Gupta, Mohammad Sadeghi, Monalisha Ojha, and Colin Atkinson. *Supporting OWL-Style Reasoning over UML-Style Models via Endogenous OWL Overlays*. Journal of Object Technology. Vol. 25, No. 3, 2026.

http://dx.doi.org/10.5381/jot.2026.25.3.a6

satisfiability criteria. Their typical use, however, is constraint checking over a given model snapshot. The services considered in this paper are different in character: they are closer to ontology-style closure, classification, and logical diagnostics, such as deriving a general parent link from a father link, inferring role membership from participation in a role-specific association, inferring composed relations from a declared property chain, or detecting that a class becomes logically unsatisfiable once explicit disjointness intent is analysed.

OWL, by contrast, was designed precisely to support such semantics-driven reasoning over classes and properties. Standard OWL reasoners provide services such as classification (subclass inference), instance checking, property hierarchy reasoning, and consistency or unsatisfiability detection under a precisely defined model-theoretic semantics (W3C OWL Working Group 2012a). However, OWL was not designed as a notation for engineering-oriented diagrams like UML, and the visualisation options available for OWL models remain modest.

Prior work has therefore explored multiple strategies for bringing formal analysis or ontology-style reasoning to UML models. One major line of work translates UML class models (often with OCL) into other analysers such as SAT-/SMT-based model finders, constraint solvers, or theorem provers, enabling satisfiability checks and counterexample generation (Richters & Gogolla 2000; Anastasakis et al. 2007; Cabot et al. 2008; Brucker & Wolff 2008). Another line of work maps UML constructs into description logics/OWL so that DL reasoners can be applied (Berardi et al. 2005). UML–OWL integration is also supported by standards and tooling ranging from OMG's ODM to transformation- and profile-based approaches (OMG 2009; Gašević et al. 2006; Parreiras & Staab 2010; Grünwald & Moser 2012; Zedlitz & Luttenberger 2012; No Magic, Inc. 2021).

Despite this progress, three recurring practical obstacles remain. First, many approaches introduce a second maintained artefact—an ontology, a transformed model, or an analysis representation—with its own lifecycle. Keeping this consistent with the evolving UML model often becomes an engineering problem in its own right. Second, mappings often focus on encoding UML meta-elements, but do not systematically capture the semantics of modelling patterns that engineers actually rely on in practice, such as subset relations between role ends, role-class idioms, and powertype/partition relationships. Third, many approaches focus on a single classification level, so domain content at other levels is not naturally incorporated into the reasoning workflow.

In this paper we present *endogenous OWL overlays*, which address these problems by keeping model authoring and reasoning distinct while preserving a single authoritative source. The approach is based on two main ideas. First, we expose the current UML-style model snapshot as a base graph: a triple/fact representation used by the overlay rules. Second, we provide a small, domain-independent overlay rule library that derives an OWL/RDFS-aligned view whenever reasoning support is desired. The overlay is *endogenous* because it augments the UML-style model rather than replacing it, and because it is regenerated on demand.

We use the term "UML-style model" rather than "UML model" because the concrete host notation is LML, a multi-level language from the multi-level modelling community (Lange & Atkinson 2018; Gerbig 2011). LML keeps the look and feel of UML class-diagram notation while extending its abstract foundation with level-agnostic constructs such as potency (de Lara & Guerra 2010). The paper therefore does not claim full UML metamodel coverage; it targets selected UML-style idioms for which OWL/RDFS reasoning gives clear, actionable feedback.

We do not claim that direct semantic extension of UML tooling, migration to richer conceptual-modelling languages, or dedicated constraint technologies are inferior. Rather, our contribution is a lightweight reuse strategy that preserves the existing authoritative UML-style model, exposes it through a base-graph representation, and provides selected OWL reasoning services on demand through a derived overlay that remains traceably connected to the source model. Overall, this paper makes the following four contributions:

1. We define the notion of an *endogenous OWL overlay* for UML-style class models: a rule-derived OWL/RDFS-aligned view computed over a base graph representing the current authoritative model snapshot.

2. We present a small, domain-independent overlay compilation kernel: reusable rules over the base-graph vocabulary that align basic LML structures to OWL/RDFS and capture common modelling idioms as derived OWL facts.

3. We demonstrate, through four reasoning scenarios, how overlays enable practical OWL-style services over familiar idioms: subproperty-driven link completion, instance-based role inference, derived relations via property chains, and detection of logically unsatisfiable classes in the presence of explicit partitioning/disjointness.

4. We provide a tool-agnostic execution workflow and characterise the reasoner portfolio enabled by overlays, including a modeller-controlled feedback loop in which selected consequences can be accepted as explicit model updates without compromising model authority or traceability.

The rest of the paper is structured as follows: Section 2 recaps the modelling and reasoning background needed for the remainder of the paper. Section 3 introduces the running example. Section 4 presents the overlay approach and rule library. Section 5 walks through four reasoning scenarios. Sections 6 and 7 discuss validation, limitations, and implications. Section 8 reviews related work, and Section 9 concludes.

2. Background

This section summarises the modelling and reasoning background underpinning the paper. We first recall what UML-style class models (and common constraint add-ons such as OCL) typically provide in practice. We then summarise the OWL reasoning services we want to access, the semantic mismatches that matter in UML-centric workflows, and the specific services targeted in this paper.

2.1. UML class models and constraint checking

UML class diagrams are primarily used to communicate and refine domain structure, and to provide a backbone for later design and implementation tasks (OMG 2017). While the UML standard defines a rich set of modelling constructs, including classes, associations, association ends, and generalisation relationships, many of their intended semantics are not available as *derived* information in everyday tools: modellers can draw a diagram, but they rarely get feedback in the form of logically implied facts.

OCL is the standard companion language for expressing additional integrity conditions over UML models (OMG 2014). In practice, OCL support tends to focus on constraint checking rather than semantic closure: a tool evaluates invariants, pre-/postconditions, and queries against a given snapshot of model instances, but it does not usually derive implied links, infer additional types, or diagnose semantic contradictions that become visible only after multiple modelling declarations are combined. This distinction matters because the services considered in this paper are closer to OWL-style entailment and classification than to traditional constraint evaluation. In short, our goal is to make OWL-style reasoning services more directly available to UML modellers, while also allowing OWL modellers to benefit from UML-style visualisations.

2.2. OWL reasoning services

OWL 2 provides a model-theoretic foundation for representing classes, properties, and logical axioms, together with a standard entailment notion that underpins automated reasoning (W3C OWL Working Group 2012a). In an OWL setting, it is routine to ask for subclass closure, instance classification, property hierarchy closure, and logical consistency or unsatisfiability diagnostics. Because these services are central to the paper, we use the following terms in their ordinary OWL sense: *subclass closure* means deriving implied subclass relations, *instance classification* means deriving additional type memberships, *property hierarchy closure* means deriving implied subproperty relations and their consequences, and *unsatisfiability diagnostics* means detecting classes that cannot possibly have instances once the stated axioms are combined.

These are the standard kinds of questions asked of ontology reasoners in ontology-engineering workflows, and they are precisely the style of feedback we want to make directly available over UML-style models. In practical terms, these services answer modelling questions such as: Which specialisations are implied but not drawn explicitly? Which additional types follow from the declared class structure and links? Which role or navigation hierarchies collapse into more general ones? And do the stated declarations jointly make some classes impossible to instantiate? This is exactly the kind of semantic feedback ontology engineers routinely use to inspect and debug ontologies, and is largely missing from UML-style authoring workflows.

OWL reasoning is typically applied over RDF graphs, that is, directed labelled graphs represented as subject/predicate/object triples. Turtle is simply a compact textual syntax for writing such RDF triples. OWL and RDFS add standard vocabulary and axiom forms on top of this graph representation, such as `rdf:type`, `rdfs:subClassOf`, and `owl:inverseOf`. For UML-centric workflows, this matters because a model snapshot can be exposed as a graph and enriched with OWL/RDFS-aligned overlay facts before being handed to an OWL reasoner.

A key practical aspect is that OWL 2 is accompanied by profiles that trade expressivity for more predictable reasoning behaviour (W3C OWL Working Group 2012b). In particular, OWL 2 RL (Rule Language) is designed to support many useful entailments by forward-chaining rule evaluation. For UML-centric workflows, this is attractive because it supports repeatable "compute the closure and inspect derived facts" feedback loops for a substantial fragment of OWL reasoning, and it aligns well with rule-engine implementations.

2.3. Semantic differences relevant to UML-centric use

Two semantic differences are especially relevant when OWL reasoning is applied in a UML-oriented setting.

Open world vs. closed world. For the positive, monotonic consequences targeted here, open-world semantics is not a problem: overlay-generated or OWL-entailed facts can be reported back as genuine consequences of the current model snapshot. The distinction matters when absence is treated as false. Under OWL semantics, absence means "unknown", so overlays provide entailment, classification, and satisfiability diagnostics, not closed-world validation or a replacement for OCL-style integrity checking.

Two-level architecture vs. multi-level intent. OWL reasoners and most description-logic tooling distinguish classes from individuals. UML-style modelling, especially in multi-level modelling languages like the LML, also contains higher-order intent such as powertypes and role markers. Our goal is not to turn OWL into a full multi-level logic but rather to allow multi-level idioms to be represented in the base graph and compiled into OWL facts when this supports a useful reasoning service.

2.4. Reasoning services targeted in this paper

The paper focuses on four representative OWL-style reasoning services that align closely with common UML modelling idioms and provide actionable feedback to modellers: property-hierarchy closure from subsetted role ends, role-membership inference via domain/range typing, derived relations via property chains, and logical diagnostics under explicit partition/disjointness declarations. These services were chosen because each corresponds to a familiar UML-style or multi-level modelling idiom and yields modeller-visible results. They are demonstrations of a useful service portfolio rather than the full limit of the approach. As discussed later, the broader architecture can be understood through three routes by which semantics enter the overlay: core UML-style notation, marker-based extensions, and small local OWL carry-overs.

3. Motivating Example

To explain and evaluate the approach we use a fragment of the well-known *Family History Knowledge Base* (FHKB) (Stevens & Stevens 2008) as a running example (Figure 1). We selected a

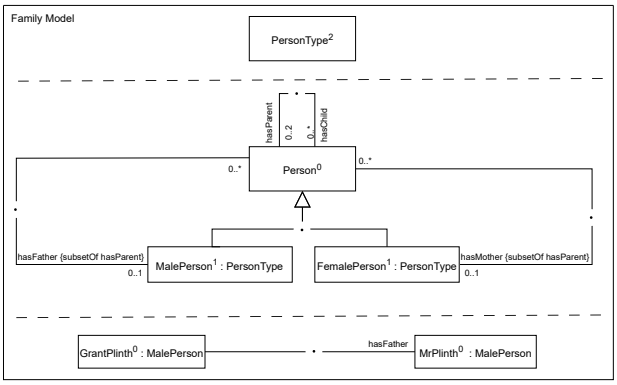


Figure 1 UML-style LML view of the FHKB fragment.

compact fragment that fits naturally into a UML-style diagram and can be extended to cover three ontological classification levels in a way familiar from multi-level modelling (Lara et al. 2014). The example is intentionally small, but it is semantically dense: it exercises four modelling idioms that are common in practice and that raise reasoning questions beyond what typical UML tools provide directly. These are (i) subsetted role ends, (ii) role classes, (iii) derived relations that require property-chain semantics, and (iv) partitioning intent that introduces disjointness constraints and can reveal hidden logical contradictions. These idioms are revisited directly in Scenarios 1–4 (Section 5).

3.1. Multi-level structure of the model

Figure 1 is organised into three horizontal levels that represent different classification levels. The diagram combines role modelling with a powertype-style classification (`PersonType`) that is later compiled into OWL constraints and queried by an OWL reasoner (Scenario 4). We use the LML notation to make multi-level intent explicit. In the multi-level modelling literature, modelling elements that can play both class-like and object-like roles are often called *clabjects*. We use that term here: a clabject may be classified by another clabject, may classify other clabjects, or both. Each clabject name carries a superscript indicating its potency, declarations of the form "x : T" indicate the declared type of clabjects at any level, and the dot notation indicates the presence of an explicit model element where standard UML would normally use lines, for example for relationships and categorisations.

At the top level, the model introduces a powertype classifier `PersonType` used to classify the subtypes of `Person`. This is an example of the powertype pattern which requires subclasses of a base class to be instances of its powertype. In this case, the pattern is a partitioning-powertype which additionally requires those subclasses to be complete and disjoint (Carvalho et al. 2016). At the next level, `Person` is the base class representing people in the domain, with `MalePerson` and `FemalePerson`, both instances of `PersonType`, appearing as its specializations. At the bottom level, individual people (e.g., `GrantPlinth`) are typed by one of these specialised classifiers. This combination of instances, classes, and *types of classes* in a single diagram is natural for engineers, but challenging for reasoning techniques that assume a strict separation between classes and individuals. Scenario 4 exploits exactly this point.

3.2. Role modelling idioms used in the scenarios

The scenarios use family relations to illustrate two role-modelling styles that UML modellers commonly employ. Figure 1 itself shows the association-end style used in Scenario 1. Scenario 2 then introduces a self-contained role-class variant to show that the overlay also supports role intent represented through a role class. The point is not to enforce one idiom, but to show that the overlay can support both.

Association-centric roles (role ends). One style represents roles through association ends on an association such as `hasParent`. Specialised role ends such as `hasFather` and `hasMother` are marked as *subsets* of the more general `hasParent` role. For a modeller, the intended reading is clear: every `hasFather` link is also a `hasParent` link. For a standard UML tool, however, these are typically treated as distinct edges unless additional analysis machinery is introduced. This idiom underpins Scenario 1.

Role-class-centric roles (role classes). A second style represents a role (e.g., Father or Mother) as a role class that participates in associations capturing role-specific relations (e.g., a child relationship). In the current host notation, this intent is represented through explicit markers in the base graph rather than through UML profile machinery. This idiom is introduced in the Scenario 2 storyboard rather than in Figure 1 itself.

3.3. Intended semantics and reasoning questions

Even in a small fragment of the FHKB such as Figure 1, key semantics are obvious to humans but not operationally accessible in typical modelling workflows. The remainder of the paper focuses on making these semantics explicit in a derived overlay so they can drive automated reasoning services:

- **Implicit relationships:** can implied `hasParent` links be derived from explicit `hasFather`/`hasMother` links?
- **Role membership:** can instances be classified into role classes based on participation in role-specific links, without requiring explicit role typing?
- **Derived associations:** can relations such as `hasGrandparent` be inferred from existing links when the intended semantics is a composition of relations?
- **Hidden contradictions:** if the modeller intends a powertype classification to be a disjoint partition, can a reasoner detect when the class structure violates this intent (e.g., by making a class logically unsatisfiable)?

To answer these questions, we (i) expose the authoritative LML model as a base graph, i.e. a triple/fact representation used by the overlay rules, (ii) derive an OWL/RDFS-aligned overlay over that base graph, (iii) analyse the resulting facts with an OWL reasoner, and (iv) present the results back to the modeller as traceable analysis feedback, with selected consequences becoming explicit model updates at the modeller's discretion. Figure 2 previews this fact view.

4. Overlay Approach

We represent the current authoritative model snapshot as a base graph, written G_{base}, over which the overlay rules are executed. In the executable artefacts accompanying the paper, G_{base} is realised as a PAMoLa (Poly-Aligned Modelling Language) base graph (*PAMoLa* 2026). PAMoLa is a self-describing modelling substrate in which a finite model snapshot is represented as a typed directed graph disciplined by a rooted custodian arborescence. Its fixed core vocabulary supplies the small set of categories and properties needed by the overlay rules, while the model content itself can grow and evolve. PAMoLa calls each first-class model element an *ordium* (plural *ordia*), a term used here to avoid the overloaded word "node" and to emphasise that these elements are named, typed, custodied, referenceable model items. Figures 2 and 3 show the part of this representation needed for the examples: Figure 2 gives HUTN (Human-Usable Textual Notation) and Turtle views of a G_{base} fragment, and Figure 3 shows the corresponding PAMoLa Core slice.

4.1. Pipeline: from LML to an OWL-consumable overlay

The end-to-end workflow has four conceptual stages. These are instantiated in the storyboard scenarios of Section 5.

1. **Authoring (authoritative model).** Engineers create and evolve a UML-style class model in LML, the UML-style concrete syntax for deep (multi-level) modelling based on potency and strictness (Lange & Atkinson 2018). In the implementation used for the examples, this authored model is supported by (i.e., representable in) PAMoLa. This model remains the single source of truth throughout.

2. **Base graph extraction.** The LML snapshot is exposed as a PAMoLa base graph: a typed triple/fact representation suitable for rule evaluation and traceability. A tool-specific front-end could also construct equivalent G_{base} facts from XML/XMI or other repository formats, but discussing such alternative adapters is outside the scope of this paper.

3. **Overlay derivation.** A small, domain-independent overlay rule library is executed over G_{base} to derive additional facts that align the model content with OWL/RDFS vocabulary and the semantics of the modelling idioms being used. The result is an OWL/RDFS-aligned overlay graph G_{owl}.

4. **Reasoning, feedback, and optional model update.** An OWL reasoner is invoked over G_{owl} to obtain entailments and diagnostics. Consequences are then presented back to modellers using the vocabulary of the authoritative model, distinguishing overlay-generated facts from reasoner-inferred information. In the current workflow such results are simply analysis results and will only become ordinary model facts if the modeller adds them.

Crucially, there is no disruptive translation step: we do not replace the LML model, and we do not maintain a parallel OWL artefact. Instead, G_{owl} is recomputed when needed from the current model snapshot. The derived overlay is aligned to standard RDFS/OWL vocabulary so it can be consumed by standard OWL reasoners and ontology tooling that consume RDF/OWL serializations. Where possible we target forward-chaining-friendly entailments in the OWL 2 RL profile (W3C 2014; W3C OWL Working Group 2012a,b).

4.2. What the G_{base} fact view enables

The G_{base} fact view provides a uniform, tool-agnostic interface to the model's abstract syntax and LML-specific extensions. It records selected PAMoLa facts, including identity, custodian location, declared type, clabject potency, relationship and participation structure, role names, relata, categorisation/intercalation structure, markers, and relevant pattern declarations.

Two points are important for the overlay approach:

- **PAMoLa base graph.** The base graph is not itself an OWL ontology. It is a faithful, query-friendly PAMoLa fact view of the LML model, intended to support rule execution and traceability. Since it is based on triples, G_{base} can be represented in RDF, as shown in the right-hand column of Figure 2. Other UML-style tools could supply equivalent G_{base} facts through an adapter.
- **Traceability by construction.** Because overlay facts are derived over the same identifiers that represent model elements in G_{base}—and, where available, the same stable persistent identifiers—derived OWL axioms and inferred consequences can be traced back to the modelling constructs and pattern instances that gave rise to them.

4.3. Endogenous OWL overlays

An *endogenous OWL overlay* is the derived graph G_{owl} produced by applying a rule set R_k to G_{base}, optionally taking into account small inline OWL annotations attached to the model. The overlay contains OWL/RDFS-aligned axioms and assertions such as:

- schema alignments (e.g., generalisation interpreted as `rdfs:subClassOf`);
- typing alignments (e.g., declared instantiation interpreted as `rdf:type`);
- pattern semantics (e.g., subsetted ends interpreted as `rdfs:subPropertyOf`);
- additional constraints when LML structure alone is not enough (e.g., property chains) or where marker-based intent should be compiled (e.g., partition disjointness).

Overlay derivation is realised by a small, domain-independent rule library, the *overlay compilation kernel*, whose rule bodies refer only to the base-graph vocabulary—entities, relationships, participations, categorisations, intercalations, markers, role names, declared types, and relatum links—rather than to domain classes such as `Person`, `Father`, or `PersonType`. Domain-specific names appear only as model data, while optional modules add support for particular idiom families.

We write R_k for the particular fragment of this library selected for a given analysis task (kernel plus any optional modules). For presentation purposes, we *group* the rules into a stable kernel R_1 and three small add-on modules ΔR_2–ΔR_4, corresponding to the additional modelling idioms illustrated in

G_{base} HUTN

```
EN1 {hasName "FamilyModel", hasCustodian Undefined, hasPotency 0, hasDeclaredType Undefined,
    EN2 {hasName "PersonType", hasPotency 2, hasDeclaredType Undefined},
    EN3 {hasName "Person", hasPotency 0, hasDeclaredType EN2},
    EN4 {hasName "MalePerson", hasPotency 1, hasDeclaredType EN2},
    EN5 {hasName "FemalePerson", hasPotency 1, hasDeclaredType EN2},
    CA1 {hasName "Male-Female", isComplete True, isDisjoint True,
        IC1{hasName "Person", isGeneralisation True, hasRelatum EN3},
        IC2 {hasName "Male", isGeneralisation False, hasRelatum EN4},
        IC3 {hasName "Female", isGeneralisation False, hasRelatum EN5},
    },
    RE1 {hasName "Parent-Child", hasAssociationName Undefined, hasPotency 1, hasDeclaredType Undefined,
        PA1 {hasName "PA_1", hasRoleName "hasParent", hasLowerBound 0, hasUpperBound 2, hasRelatum EN3},
        PA2 {hasName "PA_2", hasRoleName "hasChild", hasLowerBound 0, hasUpperBound *, hasRelatum EN3},
    },
    RE2 {hasName "Father-Child", hasAssociationName Undefined, hasPotency 1, hasDeclaredType Undefined,
        PA3 {hasName "PA_3", hasRoleName "hasFather", hasLowerBound 0, hasUpperBound 1, hasRelatum EN4,
            MK {hasName "subsetOf", hasTag subsetOf, refersTo PA1}},
        PA4 {hasName "PA_4", hasRoleName Undefined, hasLowerBound 0, hasUpperBound *, hasRelatum EN3},
    },
    ...
}
```

G_{base} RDF Turtle

```
:EN2 :hasName "PersonType" ;
     :hasCustodian :EN1 ;
     :hasPotency "2"^^xsd:integer ;
     :hasDeclaredType :Undefined .

:CA1 :hasName "Male-Female" ;
     :hasCustodian :EN1 ;
     :isComplete true ;
     :isDisjoint true .

:IC1 :hasName "Person";
     :hasCustodian :CA1 ;
     :isGeneralisation true;
     :hasRelatum :EN3 .

:RE1 :hasName "Parent-Child";
     :hasCustodian :EN1 ;
     :hasAssociationName :Undefined;
     :hasPotency "1"^^xsd:integer ;
     :hasDeclaredType :Undefined .
```

Figure 2 Illustrative excerpts of a base graph G_{base} view of Figure 1. Left: Illustrative HUTN serialisation (custodian relationships are shown by indentation). Right: Illustrative selection of corresponding RDF/Turtle facts.

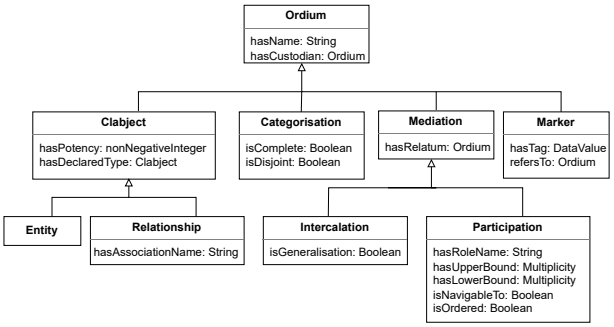


Figure 3 Fragment of the PAMoLa metamodel used to type the base graph G_{base} (*PAMoLa* 2026).

Scenarios 2–4. This organisation keeps the storyboard figures readable; it is not a requirement of the approach.

- R_1 **(kernel).** Aligns generalisation and declared typing to OWL/RDFS vocabulary and compiles subsetted association ends into `rdfs:subPropertyOf` axioms.
- ΔR_2 **(role classes).** Interprets role classes identified by explicit role markers, together with their incident associations, by reusing matched type-level role-class participations as OWL object properties with basic schema constraints (notably `rdfs:domain`/`rdfs:range`); matching instance participations then yields property assertions from which OWL reasoning can infer role membership.
- ΔR_3 **(property chains).** Carries locally attached `owl:propertyChainAxiom` annotations into the overlay so that derived relations can be inferred by OWL tooling.
- ΔR_4 **(powertype/partition intent).** Compiles explicitly declared partition intent into OWL constraints (notably disjointness, and optionally completeness when declared), enabling satisfiability-style diagnostics such as unsatisfiable class detection.

When convenient, we use the shorthand $R_2 = R_1 \cup \Delta R_2$, $R_3 = R_2 \cup \Delta R_3$, and $R_4 = R_3 \cup \Delta R_4$ to indicate which fragments are exercised in a given scenario.

4.4. Inline OWL axioms

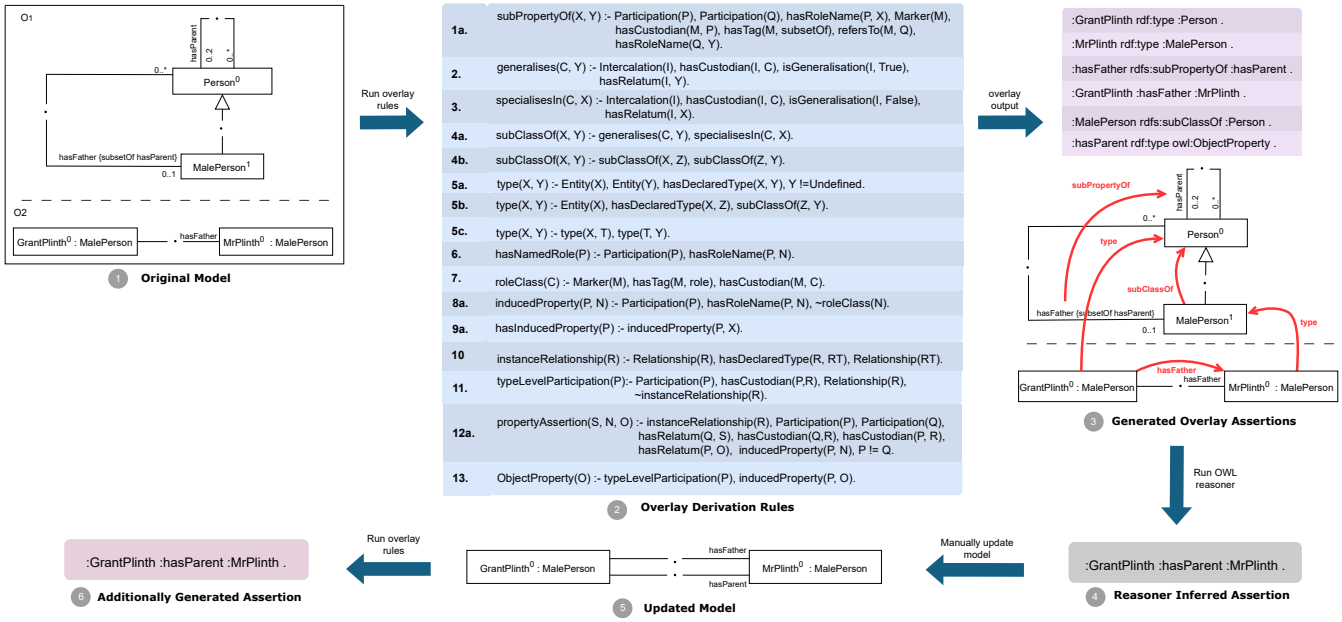
We allow inline OWL/Turtle annotations only when the intended semantics cannot be expressed cleanly in the source-model structure or in the marker vocabulary already supported by the base graph. The purpose is not to maintain a parallel ontology, but to attach small, local axioms that remain traceable to the source declaration.

The property chains in Scenario 3, where OWL provides a standard axiom form (`owl:propertyChainAxiom`) and LML does not provide an equivalent construct, illustrate this mechanism. In contrast, Scenario 4 does not rely on inline OWL annotations: partition/disjointness intent is represented in the base graph through explicit markers and flags and compiled into OWL constraints by the overlay rules. Inline axioms are therefore an exceptional carry-over mechanism, not the default route by which semantics enter the overlay.

4.5. Tool-agnostic execution workflow

A minimal execution workflow, independent of any specific modelling environment or reasoner, is as follows:

1. Export the original model as G_{base} (in the present example, a PAMoLa fact view serialised as HUTN, Datalog-style facts, or RDF/Turtle).
2. Select the overlay rule fragment(s) R_k required for the intended reasoning service (kernel plus optional modules).
3. Execute R_k to compute the overlay graph G_{owl}.
4. Invoke an OWL reasoner over G_{owl} to obtain entailments and diagnostics.



Scenario 1 (Rule set R_1): subsetted role ends induce subproperty axioms in the endogenous OWL overlay, enabling inferred association instances. The modeller can choose to add the inferred link to the authoritative original model.

5. Present consequences back to modellers as overlay-generated and reasoner-inferred results so that they have the option of extending the model accordingly.

Section 5 demonstrates this workflow in four scenarios and summarises the resulting observations without reproducing the rule listings and triple snippets that are already contained in the storyboard figures.

5. Reasoning Scenarios and Observations

This section makes the overlay approach concrete by working through four reasoning scenarios over the LML family model (Figure 1). Each scenario follows the same pattern: we start from a self-contained LML fragment, expose it as the PAMoLa base graph G_{base}, derive an OWL/RDFS-aligned overlay, apply an OWL reasoner for entailments and diagnostics, and optionally present selected consequences to the modeller for acceptance as explicit model updates.

Although all four scenarios are based on the same running example, each storyboard is intended as a self-contained fragment for the reasoning service it demonstrates. For readability, we omit irrelevant elements and show only selected overlay assertions and rule fragments. The figures should therefore be read as explanatory excerpts rather than complete closures. The companion repository contains the corresponding PAMoLa HUTN/fact files, overlay rules, Turtle exports, expected outputs, reasoner outputs, and a concise description of the PAMoLa vocabulary used in the prototype.

5.1. Scenario 1: Role ends and subproperties

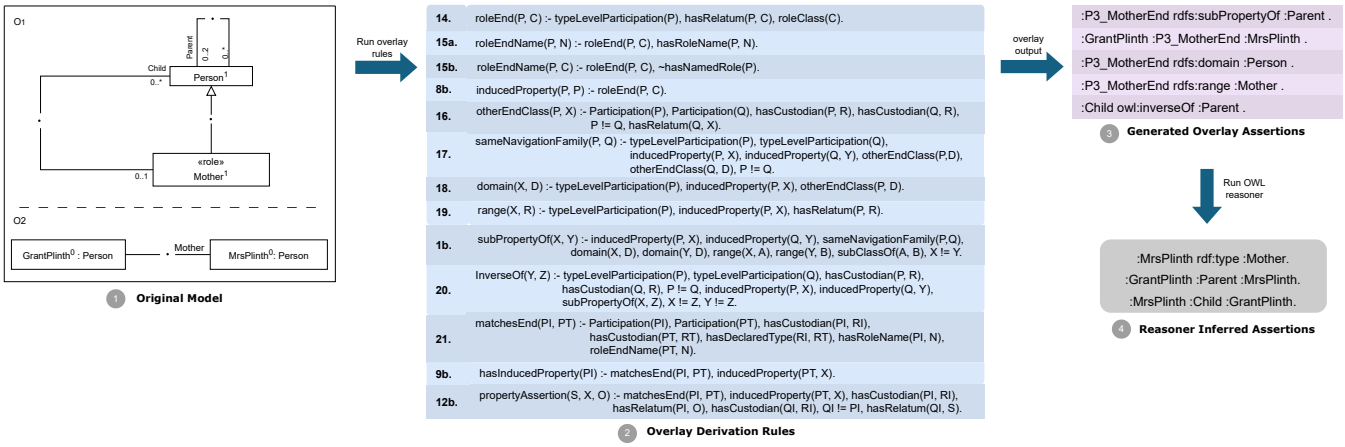
Scenario 1 starts from a conventional LML pattern: a self-association (e.g., `hasParent`) and a specialised role end (e.g., `hasFather`) marked as a *subset* of the more general end. In LML this is a compact way to state an inclusion constraint between role ends, but it is not something a standard UML tool typically exploits to derive additional instance links.

Rule set R_1 compiles this pattern into the endogenous OWL overlay by generating an `rdfs:subPropertyOf` axiom between the specialised and general role-end properties. For readability, rules are numbered; when multiple rule bodies share the same head we use letters (a, b, ...). Rules with the symbol "~" at the beginning denote stratified negation-as-failure, as supported by the rule engine. Once the overlay is available, standard subproperty reasoning yields the implied `hasParent` links from explicit `hasFather` links. The inferred `hasParent` link is first an overlay/reasoner consequence, analogous to a read-only analysis result. The storyboard also illustrates the closed-loop option: a modeller may choose to accept selected derived links as explicit model updates, and on the next overlay run they are treated as ordinary asserted facts.

Outcome. The Scenario 1 storyboard shows how subsetted role ends are compiled into `rdfs:subPropertyOf` axioms in the overlay, enabling standard subproperty reasoning to derive implied `hasParent` links from explicit `hasFather` links.

5.2. Scenario 2: Role classes and domain-based typing

Scenario 2 demonstrates that overlays do not force a single role-modelling idiom. Instead of representing a role as an association end, the LML model uses a role class identified by an explicit role marker (e.g., `Mother`) connected by an association that



Scenario 2 (Rule set $R_2 = R_1 \cup \Delta R_2$): role-class participations are reused as OWL object properties, while instance-level role names identify the occupied role end. Domain/range constraints then support role membership inference from links.

captures a role-specific relation (e.g., `Child`). This style is common when roles should participate in further structure or constraints as first-class modelling elements. In the PAMoLa host notation, stereotype-like intent is achieved through markers rather than through a literal UML profile mechanism.

Scenario 2 uses the role name carried by an instance-level participation as the explicit source-model signal that an endpoint is playing a marked role. In the running example, `MrsPlinth` and `GrantPlinth` initially start only as ordinary `Person` instances. Since the participation that connects `MrsPlinth` to the role-specific relationship carries the role name `Mother`, the overlay rules match that participation to the corresponding type-level role-class participation whose relatum is the marked role class `Mother`. That matched type-level participation is then reused as the OWL object property in the overlay. The resulting inference path is operationally straightforward. The overlay derives a property assertion from the instance link, together with schema-level facts such as `rdfs:domain` and `rdfs:range` for the source-anchored role-class property. Because the property has range `Mother`, standard OWL domain/range reasoning can infer that `MrsPlinth` is an instance of `Mother`, even though that type was not asserted in the original model. Thus the role membership is inferred from participation in the role-specific link, not assumed in advance.

The kernel rules from Scenario 1 are reused unchanged. ΔR_2 adds a domain-independent compilation for the role-class idiom: role-class participations and their incident associations are interpreted into OWL object properties with derived domain/range constraints, while instance-level participations generate the corresponding property assertions. Type-level guards separate schema-level property declarations from ABox property assertions, while the `~roleClass(N)` guard routes role-class name tokens such as `Mother` to R2 rather than ordinary navigation-property export. As in Scenario 1, the inferred role membership is treated as a derived analysis result by default. It is shown using source-model vocabulary so that the modeller can choose whether to include it explicitly in the model.

Outcome. The Scenario 2 storyboard shows how a role name on an instance participation is matched to a marked role-class participation, how the corresponding source-anchored OWL object property receives domain/range constraints, and how role membership is inferred from the resulting property assertion.

5.3. Scenario 3: Property chaining

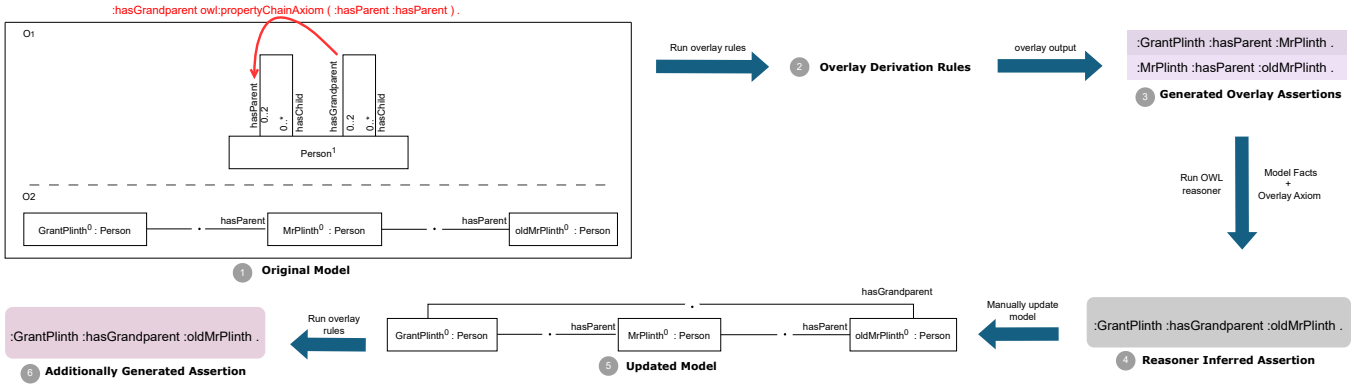
Scenario 3 addresses a kind of inference that is not directly expressible using LML structure alone: property chains. The model uses a base relation (e.g., `hasParent`), and the intended derived relation (e.g., `hasGrandparent`) should hold whenever two parent steps compose.

Here the authoritative LML model remains the source of truth, but the missing semantics is supplied by a small inline OWL axiom attached to the model `:hasGrandparent owl:propertyChainAxiom (:hasParent :hasParent)` . ΔR_3 ensures that this axiom is carried into the overlay in OWL-consumable form. Scenario 3 is deliberately treated as local OWL carry-over rather than as a generic UML-structure compilation rule: the modeller explicitly supplies the property-chain intent, and the overlay carries that axiom into the reasoner input while preserving traceability to the source declaration. Given the chain axiom and the base links, standard OWL reasoning can infer the derived association instances; for this Horn-style pattern, the same entailment can also be obtained by forward-chaining rule evaluation.

Outcome. The Scenario 3 storyboard illustrates how the locally attached chain axiom, carried into G_{owl}, enables derived association instances to be inferred by standard OWL tooling.

5.4. Scenario 4: Partitioning Powertypes

Scenario 4 focuses on multi-level intent expressed through the *powertype* pattern. A powertype relates a base class (e.g., `Person`) to a higher-order classifier (e.g., `PersonType`) whose instances correspond to subtypes of the base. In this paper we adopt the terminology for powertypes used in MLT (a well-founded multi-level theory) (Carvalho et al. 2016). In MLT, a



Scenario 3 (Rule set $R_3 = R_2 \cup \Delta R_3$): an inline OWL property chain axiom, carried by the endogenous OWL overlay, enables derived association instances (e.g., hasGrandparent) to be inferred from composed base links.

regular powertype relationship between a higher-order type and the base type it classifies is referred to as a *characterization* relationship, whereas a powertype relationship that additionally requires the categorisation to be both *disjoint* and *complete* is referred to as a *partition*. Intuitively, when a partition relationship applies, every instance of the base class must instantiate *exactly one* of the subtypes induced by the powertype (no gaps, and no overlaps) (Carvalho et al. 2016).

This is the intent captured in our model via an explicit partition marker and, in the Core slice of Figure 3, the `isDisjoint` and `isComplete` flags on the `Categorisation` element, with `Intercalation` members using `isGeneralisation` and `hasRelatum`. In the scenario rules, the helper predicate `partitionedBy` recognises the full partition declaration: the marked categorisation identifies the powertype, the general intercalation identifies the base type, and the categorisation is declared both disjoint and complete. The OWL diagnostic shown here, however, uses only the disjointness component of that declaration: the overlay exports `owl:disjointWith` axioms between the classified subtypes. We do not export a covering axiom for completeness in this scenario; adding such an axiom would be a natural extension, but it is not needed to expose the unsatisfiable class shown here.

ΔR_4 compiles such partition intent into OWL constraints in the overlay. When the modeller declares that a powertype classification represents a disjoint partition, the overlay derives the corresponding `owl:disjointWith` axioms between the classified subtypes. A DL reasoner can then detect modelling errors that remain invisible at the LML level—for example, a class that inherits from disjoint subtypes becomes *unsatisfiable*.

Outcome. The Scenario 4 storyboard shows the compiled disjointness constraints and the resulting unsatisfiable-class diagnostic obtained with a DL-capable reasoner.

5.5. Observations

The scenarios lead to several practical observations about the scope and behaviour of endogenous OWL overlays.

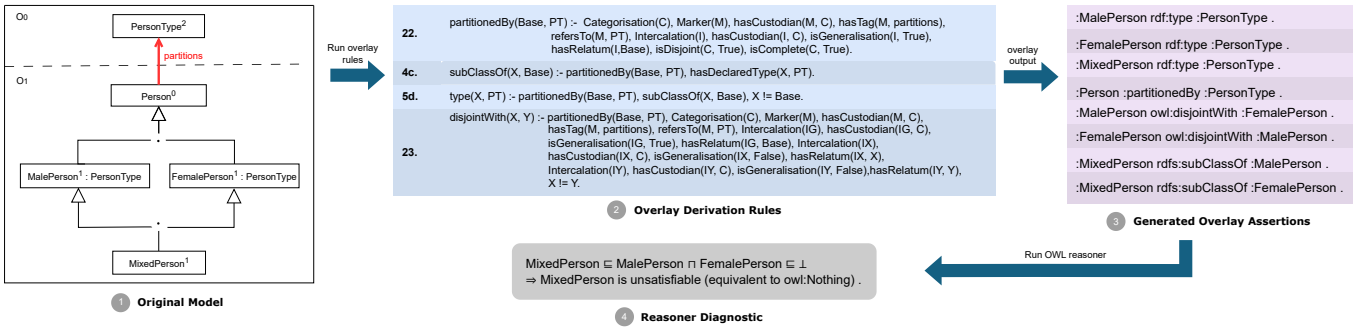
Endogeneity preserves modelling authority and traceability. In all scenarios, the LML diagram remains the authoritative artefact. The overlay is computed from the base graph G_{base} and can be re-derived on demand; there is no separately maintained OWL model that can drift out of sync. Because overlay facts are derived over the same ordium identifiers that represent model elements in the base graph, entailments and diagnostics can be traced back to the modelling declarations and pattern instances that caused them.

Rules are domain-independent and modular. The scenario progression is engineered so that each new capability is introduced by exercising an additional, focused module on top of the kernel rule library. Later scenarios therefore extend (rather than replace) earlier ones.

Inline OWL is used only when source-model structure and the supported marker vocabulary are not enough. Scenario 3 illustrates the intended policy: OWL/Turtle annotations are introduced only when the modeller's intended semantics cannot be expressed cleanly using LML structure or the current marker vocabulary. These annotations are carried by the overlay alongside rule-derived axioms, preserving traceability. This capability therefore essentially allows OWL expression to be applied to UML-like models.

Two reasoning regimes are supported. Most services demonstrated here (Scenarios 1–3) fit a forward-chaining Horn regime consistent with OWL 2 RL (Rule Language). When satisfiability-style feedback is required (Scenario 4), the same overlay can be exported to a DL-capable reasoner.

Model updates are optional and should be selective. Accepting inferred facts into the LML model is useful when derived information should become part of the design (e.g., to make implicit structure explicit for downstream stakeholders). However, not every entailment is necessarily a modelling element worth recording explicitly; the storyboards treat such updates as a modeller-controlled choice. It is possible to automate the updating of the LML model with inferred facts, but this is beyond the scope of this paper.



Scenario 4 (Rule set $R_4 = R_3 \cup \Delta R_4$): powertype/partition semantics are compiled into disjointness constraints in the endogenous OWL overlay, enabling unsatisfiable class detection when the LML model violates the intended partition.

6. Evaluation

The contribution of this paper is primarily architectural and semantic, but the evaluation aims to make two things explicit: first, it shows that the overlay rules compile the intended modelling semantics into the expected OWL/RDFS-aligned facts, and second, that this can be done without maintaining a separately synchronised OWL artefact.

6.1. Prototype workflow and validation target

The current prototype pipeline has four stages: (1) expose the current LML snapshot as the PAMoLa base graph G_{base}, (2) execute the selected overlay rules over the corresponding PAMoLa fact view, (3) export the resulting overlay facts to RDF/Turtle together with any inline OWL axioms, and (4) analyse the exported overlay with OWL tooling. In the current prototype, stage (2) uses Nemo (Ivliev et al. 2024) and stage (4) uses HermiT (Glimm et al. 2014). The storyboard figures in Section 5 instantiate this pipeline. In this paper, the validation corpus consists of the four self-contained scenario models derived from the FHKB running example in Section 3; each was executed through the same export–overlay–reasoning pipeline and checked against an expected reasoning outcome. We also provide a companion repository containing the scenario artefacts, PAMoLa HUTN files, exported overlays, Turtle files, expected outputs, reasoner outputs, and a concise description of the PAMoLa vocabulary used to realise G_{base} in the examples.[1]

The current prototype supports modeller-controlled acceptance of selected consequences rather than a fully integrated round-trip modelling environment: inferred consequences can be copied back into the authoritative model when they are worth turning into explicit design commitments, but we have not yet implemented a push-button plugin that updates the LML model automatically. In an integrated environment, such derived consequences would naturally be exposed first as analysis results, becoming ordinary model content when the modeller explicitly accepts them. Likewise, this paper evaluates semantic correctness and traceability rather than benchmarking alternative rule engines or OWL reasoners; Nemo and HermiT are cited here simply to document the current prototype implementation.

6.2. Scenario-based semantic validation

Section 5 serves as a semantic validation harness. For each scenario, we check that (i) overlay derivation produces the intended OWL/RDFS-aligned facts from the current authoritative model snapshot, and that (ii) standard OWL reasoning over the resulting overlay yields the expected consequences or diagnostics. Table 1 summarises these checks.

Concretely, the validation checks were as follows. In Scenario 1 we verified that the exported overlay contains the expected `rdfs:subPropertyOf` alignment and that the reasoner derives the implied `hasParent` link from the asserted `hasFather` link. In Scenario 2 we verified that the role-name signal on the instance participation is matched to the corresponding type-level role-class participation, that the overlay exports the expected object-property, domain/range, subproperty, inverse-property, and property-assertion facts, and that the reasoner infers the previously unasserted `Mother` membership. In Scenario 3 we verified that the property-chain axiom is carried into the overlay and that the expected `hasGrandparent` link is derived. In Scenario 4 we verified that the exported disjointness constraints match the declared partition intent and that the reasoner reports the expected unsatisfiable class. The companion repository contains the corresponding Nemo rules, Turtle exports, and HermiT outputs for inspection.

These checks support the central claim that the overlay library acts as a systematic compilation from common LML idioms to OWL axioms and assertions, so that off-the-shelf reasoners can provide the intended entailments and diagnostics over familiar UML-style models. The purpose of these checks is to validate the correctness of the compilation route on representative idioms, not to claim broad empirical coverage of typical UML model corpora or workflow usability.

6.3. Effort, traceability, and current limits

A key design goal of endogenous overlays is to minimise authoring and maintenance effort relative to approaches that create and maintain a parallel OWL model. In our workflow, the LML diagram remains the single authoritative artefact; the overlay is recomputed on demand from the current base graph. Addi-

[1] https://github.com/shilpi-gupta-mzn/Endogenous-OWL-Overlays

Scen.	Source idiom	Exported overlay	Reasoner consequence	Validation target
1	subsetted role ends	`rdfs:subPropertyOf`, `rdf:type`, direct property assertions	implied `hasParent` link from explicit `hasFather` link	subproperty-driven link completion
2	role class + role-named instance participation	source-anchored object property, `rdfs:domain`, `rdfs:range`, `rdfs:subPropertyOf`, `owl:inverseOf`, property assertion	inferred `Mother` membership from link participation	role inference via domain/range typing
3	local property-chain axiom	`owl:propertyChainAxiom` plus base links	inferred `hasGrandparent` link	derived associations via composed relations
4	powertype/partition intent	`owl:disjointWith` plus subclass structure	unsatisfiable class under multiple inheritance	satisfiability-style diagnostic

Table 1 Scenario-based semantic validation of the overlay compilation and downstream reasoning.

tional authoring is required only in the relatively small set of cases where source-model structure and the supported marker vocabulary are not enough, and where the modeller therefore attaches a small local OWL axiom (e.g., a property chain). Because derived overlay facts use the same identifiers as the base graph, entailments and diagnostics can be related back directly to the model elements and pattern declarations that triggered them. This supports explanation-oriented feedback such as "this inferred parent link follows from that subset declaration" or "this class becomes unsatisfiable because these partition-induced disjointness axioms clash with multiple inheritance."

Most overlay generation targets a forward-chaining, Horn-style reasoning regime consistent with OWL 2 RL tooling (W3C OWL Working Group 2012b). This makes the approach a good fit for repeatable "compute the closure and inspect the derived facts" loops for many modelling tasks. When deeper reasoning is required, most notably satisfiability-style feedback such as unsatisfiable class detection, the same derived overlay can be delegated to a DL-capable reasoner.

The present evaluation does *not* provide a full-scale usability or performance study over large model corpora, nor a benchmark comparison between alternative rule engines or OWL reasoners. Those remain important future work once the architectural and semantic core is stable.

7. Discussion

The paper treats OWL as a *reasoning service layer*, not as a replacement authoring notation. The central design decision is *endogeneity*: the LML model remains authoritative, while an OWL/RDFS-aligned overlay is recomputed on demand over the current base graph G_{base}. This section discusses why that separation matters, what broader scope the approach supports, how it relates to multi-level modelling and partition semantics, and where the semantic and engineering limits lie.

7.1. Why an endogenous overlay?

A direct UML-to-OWL transformation typically produces a second artefact with its own lifecycle. Once an OWL ontology becomes a maintained deliverable, the engineering problem shifts to synchronisation: keeping the ontology consistent with the evolving UML model and preserving traceability. Endogenous overlays avoid that potential source of inconsistencies by making the OWL view recomputable on demand from the current authoritative snapshot.

This has three practical consequences. First, model authority is preserved: modelling decisions remain expressed in LML wherever possible, while OWL axioms appear primarily as derived overlay facts and only a small number of local OWL axioms are attached when the intended semantics cannot be stated cleanly in LML or using PAMoLa markers. Second, traceability is direct by construction: overlay facts are derived over the same identifiers as G_{base}, so entailments and diagnostics can be related back to the concrete model elements and pattern declarations that caused them. Third, reasoners become a task-level choice: because the derived view is aligned to standard OWL/RDFS vocabulary, RL-style rule evaluation can be used for closure tasks and DL-capable reasoning for satisfiability-style diagnostics (W3C OWL Working Group 2012a,b).

In the current prototype, derived information is treated primarily as feedback to the modeller. We do not claim an automatic reverse transformation as part of the current prototype. In an integrated environment, derived consequences would more naturally be exposed first as read-only analysis results and only be turned into ordinary model content when the modeller explicitly accepts them as design commitments.

7.2. Broader scope of the approach

The scope of the approach is best understood in terms of how semantics enter the overlay, rather than by asking whether the paper covers the entire UML metamodel or enumerates every possible OWL construct. In the current architecture there are three main routes. First, semantics can come from the core UML-style notation itself: classes, associations, role ends, generalisation/categorisation structures, and instance links. Second, semantics can come from explicit marker-based extensions that record additional modelling intent. Third, semantics can be added through local OWL axioms when the intended meaning cannot be stated cleanly in the source model structure alone.

The first route therefore covers a broad structural kernel of UML-style modelling. At this level, the overlay can compile typing, subclassing, subproperties, domains, ranges, inverse properties, property assertions, and disjointness-related consequences directly from the model structure. Scenario 1 is the clearest example of this route: it uses ordinary UML-style notation only and shows how subsetted role ends are compiled into subproperty reasoning.

The second route is marker-based extension. Here the modeller uses explicit markers to state additional intent that, in standard UML, would often be signalled through stereotypes, tagged values, profiles, or other extension mechanisms. In the current paper, role classes and partition/disjointness intent are handled in this way (Scenarios 2 and 4). More generally, the architecture can support extensions whose intended meaning can be captured by overlay rules over the base graph. This is

why the paper speaks of a domain-independent overlay kernel rather than a closed list of hard-wired scenarios.

The third route is local OWL carry-over. For some semantics, the source-model structure alone is not the most natural vehicle. In such cases, a small local OWL axiom can be attached and carried into the overlay, as in the property-chain example of Scenario 3. This does not mean that the approach covers all of OWL without qualification. Its cleanest and most defensible boundaries are still those of the underlying reasoning profile and implementation strategy: clean OWL 2 RL support excludes constructs such as `DisjointUnion` and `ReflexiveObjectProperty`; richer datatype constructs, top/bottom properties, and heavier cardinality patterns sit outside the current clean claim; and list-based or annotation-heavy constructs require either preprocessing, direct carry-over, or DL-capable tooling. The four scenarios are therefore demonstrations of a broader but still disciplined architecture, not ad hoc exceptions to it.

7.3. Powertypes, partitions, and multiple levels

Multi-level modelling makes "types of types" explicit (e.g., via potency and deep instantiation) (Atkinson & Kühne 2001; Odell 1994). OWL reasoners and most description-logic tooling, however, are fundamentally organised around a two-level separation of classes and individuals. UML contains devices such as powertypes and stereotypes that hint at higher-order classification, but mainstream UML semantics and tooling do not provide a robust general foundation for multi-level modelling. This is why the paper uses LML as the concrete host notation: it supports individuals, types, and higher-order classifiers more uniformly.

The approach is therefore pragmatic rather than doctrinal. It does not claim to turn OWL into a full multi-level logic. Instead, it treats multi-level constructs as modelling patterns and compiles only the consequences needed for a reasoning service. Scenario 4 illustrates this for powertype/partition intent. In particular, when we speak of a *partition*, we adopt the notion formalised in MLT: a higher-order type partitions a base type when it both completely and disjointly characterises it (Carvalho et al. 2016). In the overlay, the disjointness component is compiled into explicit `owl:disjointWith` constraints between the classified subtypes, enabling satisfiability-style diagnostics such as unsatisfiable class detection. If completeness is also declared, it can be preserved as an overlay fact and exported later as a stronger covering constraint for DL-capable tooling, but that is not required for the inconsistency exhibited in Scenario 4.

7.4. Limitations and adoption considerations

Endogenous overlays are a bridge to OWL reasoning services; they do not change OWL's underlying semantic assumptions. In particular, OWL reasoners remain open-world: absence of information means "unknown", not "false" (W3C OWL Working Group 2012a). Overlays therefore provide access to OWL-style entailment and classification, but do not by themselves supply closed-world validation or a general replacement for OCL-style integrity checking. SHACL-style or query-based closed-world validation could be integrated into the overlay pipeline when a workflow needs it, but this is orthogonal to the positive entailment services demonstrated here.

The current rule library also targets only a selected set of modelling idioms. We do not currently compile aggregation/-composition semantics, association classes, n-ary associations, richer datatype/property constraints, or general arithmetic/aggregate OCL conditions. These remain possible future overlay modules, but some would move the derived view beyond forward-chaining-friendly fragments and would require DL-capable reasoning and stronger explanation support.

From a tooling perspective, the adoption path is conceptually straightforward: export the current LML snapshot as the PAMoLa base graph G_{base}, compute the selected overlay, invoke the chosen OWL reasoner, and present consequences in modelling terms. The main engineering work lies in integrating this pipeline into modelling environments, providing explanation-oriented feedback, and deciding when derived consequences should be accepted as explicit design commitments. Broader evaluation over typical UML model corpora and performance/scalability benchmarking remains future work.

8. Related Work

The problem of combining the pragmatic advantages of UML-style class modelling with the formal reasoning services of ontologies and logics has been addressed from several angles. This section positions endogenous OWL overlays relative to (i) UML analysis via model finding/constraint solving, (ii) UML–OWL integration and transformation approaches, (iii) hybrid reasoning and integrity validation on RDF/OWL graphs, and (iv) multi-level modelling and powertype/partition semantics.

8.1. Analysis via model finding and constraint solving

A long-standing strategy for rigorous feedback on UML class models is to translate them (often together with OCL constraints) into an analyser with a precise formal semantics. The USE tool family, for example, supports checking OCL constraints over object diagrams and exploring consistency via snapshot search (Richters & Gogolla 2000). Other work maps UML/OCL to Alloy to enable bounded model checking and counterexample generation (Anastasakis et al. 2007). Translations to constraint satisfaction problems (CSP) have also been used for checking class-diagram properties and OCL constraints (Cabot et al. 2008), and theorem-proving frameworks such as HOL-OCL provide a reliable way of reasoning about OCL specifications, at the cost of heavier interaction (Brucker & Wolff 2008).

These lines of work are valuable, but their goal is typically *validation* (find a counterexample or show satisfiable) rather than OWL-style *entailment services* (derive implied facts, compute hierarchies, compute consequences). They also typically require an explicit translation step into a solver-specific representation. In contrast, endogenous overlays provide standards-based OWL reasoning services as a derived OWL/RDFS-aligned view, recomputed on demand from the LML model snapshot.

8.2. UML to OWL and ODM-style integration

A second major strand maps UML constructs into description logics/OWL so that DL reasoners can be applied for classi-

fication and consistency feedback (Berardi et al. 2005). At the standards level, the OMG Ontology Definition Metamodel (ODM) provides profiles to connect UML-based modelling with ontology representations (OMG 2009).

Transformation-oriented approaches similarly map UML models into OWL/RDF serialisations (Gašević et al. 2006; Zedlitz & Luttenberger 2012; Grünwald & Moser 2012), and UML tool integrations and profile-based solutions support practical workflows (e.g., TwoUse and Cameo-based tooling) (Parreiras & Staab 2010). Related ecosystems such as Ecore/EMF also support derived features, but typically as tool-level properties inside the modelling environment rather than as a recomputable standards-based OWL overlay over an authoritative model snapshot (Eclipse Foundation 2014). In our workflow, derived facts are similarly analysis results by default, but are represented as traceable overlay consequences that can become explicit authoritative-model facts if the modeller so chooses.

Compared to these approaches, overlays differ in two pragmatic respects. First, overlays are engineered to avoid a separately maintained ontology artefact: the OWL view is a *recomputable* overlay derived from the authoritative model snapshot, which reduces synchronisation drift risks. Second, the emphasis is not only on encoding UML meta-elements, but on compiling the semantics of modelling patterns that practitioners rely on (subsetted role ends, role-class idioms, and powertype/partition intent), so that the resulting OWL view supports the reasoning questions modellers actually ask.

8.3. Hybrid reasoning and integrity validation

Within the Semantic Web community, many techniques combine OWL reasoning with rule engines, query answering, and integrity-style validation. OWL 2 RL is commonly realised through forward-chaining rule evaluation (W3C OWL Working Group 2012b), and SHACL provides a way to validate RDF graphs against shapes (with validation, rather than entailment, semantics) (W3C RDF Data Shapes Working Group 2017).

Our approach is compatible with this broader tool ecosystem, but keeps the modelling centre of gravity on UML-style diagrams: overlay rules act as a compilation step from modelling patterns to OWL/RDFS axioms; standard OWL tooling can then be reused for entailment and (when needed) satisfiability-style diagnostics, while integrity-style checks can be added orthogonally where a workflow requires closed-world validation.

8.4. Multi-level modelling, powertypes, and partitions

Multi-level modelling makes classification levels explicit and supports constructs such as potency and deep instantiation (Atkinson & Kühne 2001). Powertypes and related "types of types" patterns have long been used in modelling practice (Odell 1994), and systematic tool support for multi-level modelling has been explored in several environments (e.g., deep metamodeling and m-object/m-relationship approaches) (de Lara & Guerra 2010; Neumayr et al. 2009). In contrast, OWL reasoners are fundamentally organised around a two-level separation (TBox/ABox), whereas engineers often want reasoning support over multi-level classification schemes. Endogenous OWL overlays do not attempt to turn OWL into a full multi-level logic. Instead, we treat selected multi-level idioms as patterns whose *relevant consequences* can be compiled into an OWL-consumable form when a reasoning service benefits from it.

In particular, our use of *partition* intent in Scenario 4 is grounded in MLT, which formalises variants of the powertype pattern and defines *partitions* as (completely and disjointly) characterising a base type (Carvalho et al. 2016). We compile the consequences needed for the targeted reasoning service (disjointness, and optionally completeness when declared) into explicit OWL constraints in the overlay, enabling standard reasoners to detect contradictions such as unsatisfiable classes.

9. Conclusion

This paper addresses a recurring tension in modelling practice: UML-style class models, here written in LML, are optimised for engineering communication and design, but provide only modest and often disconnected reasoning services, whereas OWL is optimised for well-defined inference but is less well suited to compact, engineering-oriented diagrammatic representation. Our answer is to separate authoring from reasoning: the LML model remains the authoritative artefact, while OWL-style reasoning is enabled through an endogenous OWL overlay computed on demand.

The approach hinges on two ingredients. First, the current LML model snapshot is exposed as the PAMoLa base graph Gbase, a triple/fact representation suitable for rule execution and traceability. Second, a small, domain-independent overlay rule library derives an OWL/RDFS-aligned overlay graph Gowl that can be consumed by off-the-shelf OWL tooling without maintaining a separately synchronised OWL artefact. Where a modeller's intent cannot be expressed in LML or as a marker vocabulary, local OWL axioms, such as property chains, can be attached and carried into the same overlay view. Other extensions, such as partition/disjointness intent in Scenario 4, are handled through marker-based compilation into OWL constraints.

Across four scenarios, we showed how this enables practical OWL-style services directly over familiar modelling idioms: subproperty-driven link completion for subsetted role ends, instance-based role inference for role classes via source-anchored domain/range constraints, derived associations via property chains, and detection of logically unsatisfiable classes when powertype/partition intent is compiled into explicit OWL constraints. Crucially, the workflow remains LML-centric: overlay-generated and reasoner-inferred consequences can be inspected as analysis results and selectively accepted by the modeller as explicit design decisions in the authoritative model.

There are numerous ways in which this approach can be evolved. A broader catalogue of LML idioms, including additional multi-level modelling constructs, can be covered by further domain-independent overlay rules; explanation support can make inferences easier to justify in modelling terms; and incremental overlay recomputation can improve responsiveness in interactive settings. Finally, systematic evaluation on a wider range of models would be needed to characterise scalability and to better understand when OWL-style open-world reasoning aligns with, or diverges from, typical modelling expectations.

Acknowledgements

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

References

Anastasakis, K., Bordbar, B., Georg, G., & Ray, I. (2007). UML2Alloy: A challenging model transformation. In *MODELS 2007 Conference.*

Atkinson, C., & Kühne, T. (2001). The essence of multilevel metamodeling. In *UML 2001* (Vol. 2185). Springer.

Berardi, D., Calvanese, D., & Giacomo, G. D. (2005). Reasoning on UML class diagrams. *Artificial Intelligence*, *168*.

Brucker, A. D., & Wolff, B. (2008). HOL-OCL: A formal proof environment for UML/OCL. In *Fundamental approaches to software engineering (fase 2008)* (Vol. 4961). Springer.

Cabot, J., Clarisó, R., & Riera, D. (2008). Verification of UML/OCL class diagrams using constraint programming. In *Proceedings of (icstw).* IEEE.

Carvalho, V. A., Almeida, J. P. A., & Guizzardi, G. (2016). Using a Well-Founded Multi-level Theory to Support the Analysis and Representation of the Powertype Pattern in Conceptual Modeling. In *CAiSE 2016* (Vol. 9694). Springer.

de Lara, J., & Guerra, E. (2010). Deep metamodeling with metadepth. In *Model driven engineering languages and systems* (Vol. 6141). Springer.

Eclipse Foundation. (2014). *Estructuralfeature (emf documentation).* Eclipse EMF Javadoc.

Gašević, D., Djurić, D., & Devedžić, V. (2006). *Model driven architecture and ontology development.* Springer-Verlag Berlin Heidelberg.

Gerbig, R. (2011). *The level-agnostic modeling language: Language specification and tool implementation.* (ub-madoc.bib.uni-mannheim.de/37153)

Glimm, B., Horrocks, I., Motik, B., Stoilos, G., & Wang, Z. (2014). HermiT: An OWL 2 Reasoner. *Journal of Automated Reasoning*, *53*(3).

Grünwald, A., & Moser, T. (2012). umltuowl - a both generic and vendor-specific approach for uml to owl transformation. In *Software engineering and knowledge engineering (2012).*

Ivliev, A., Gerlach, L., Meusel, S., Steinberg, J., & Krötzsch, M. (2024). Nemo: Your friendly and versatile rule reasoning toolkit. In P. Marquis, M. Ortiz, & M. Pagnucco (Eds.), *Proc kr 2024.* IJCAI Organization.

Lange, A., & Atkinson, C. (2018). Multi-level modeling with melanee. In *MULTI 2018 Workshop.* Springer.

Lara, J. D., Guerra, E., & Cuadrado, J. S. (2014). When and how to use multilevel modelling. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *24*(2).

Neumayr, B., Grün, K., & Schrefl, M. (2009). Multi-level domain modeling with m-objects and m-relationships. In *Proceedings of the sixth asia-pacific conference on conceptual modeling-volume 96.*

No Magic, Inc. (2021). Cameo concept modeler plugin documentation [Computer software manual]. Retrieved from docs.nomagic.com/display/CCMP2021x (UML to Equivalent OWL in OWL Functional Syntax)

Odell, J. J. (1994). Power types. *Journal of Object-Oriented Programming*, *7*(2).

OMG. (2009). *Ontology Definition Metamodel (ODM), version 1.0* (Tech. Rep. No. formal/2009-05-01). OMG Specification. (www.omg.org/spec/ODM/1.0)

OMG. (2014). *Object constraint language (OCL) 2.4.* OMG Specification. (www.omg.org/spec/OCL/2.4)

OMG. (2017). *Unified Modeling Language (UML), version 2.5.1.* OMG Specification. (www.omg.org/spec/UML/2.5.1)

PAMoLa. (2026). Retrieved 2026-06-13, from https://pamola.net

Parreiras, F. S., & Staab, S. (2010). Using ontologies with UML class-based modeling: The twouse approach. *Data & Knowledge Engineering*, *69*(11).

Richters, M., & Gogolla, M. (2000). Validating UML models and OCL constraints. In *International conference on the unified modeling language.*

Stevens, R., & Stevens, M. (2008). A family history knowledge base using OWL 2. In *OWLED* (Vol. 432). CEUR-WS.org.

W3C. (2014). *RDF schema 1.1.* W3C Recommendation. (www.w3.org/TR/rdf-schema/)

W3C OWL Working Group. (2012a). *OWL 2 web ontology language: Direct semantics.* W3C Recommendation. (www.w3.org/TR/owl2-direct-semantics/)

W3C OWL Working Group. (2012b, December). *OWL 2 Web Ontology Language: Profiles* (W3C Recommendation). W3C. (www.w3.org/TR/owl2-profiles/)

W3C RDF Data Shapes Working Group. (2017). *SHACL: Shapes constraint language.* W3C Recommendation. (www.w3.org/TR/shacl/)

Zedlitz, J., & Luttenberger, N. (2012). Transforming between uml conceptual models and owl 2 ontologies. In *Proceedings of the 2012 terra cognita workshop.*

About the authors

Shilpi Gupta is a doctoral researcher at the University of Mannheim, Germany. She holds a Bachelor of Technology degree in Information Technology and a Master of Technology degree in Computer Science Engineering from India. You can contact the author at shilpi.gupta@uni-mannheim.de.

Mohammad Sadeghi is a doctoral researcher at the University of Mannheim, Germany. He holds a Bachelor degree in Computer Engineering and a Master degree in Software Engineering from Iran. You can contact the author at mohammad.sadeghi@uni-mannheim.de.

Monalisha Ojha is a doctoral researcher at the University of Mannheim, Germany. She holds an Integrated Master degree in Mathematics and Computing from India. You can contact the author at monalisha.ojha@uni-mannheim.de.

Colin Atkinson leads the Software Engineering Group at the University of Mannheim, Germany. He holds a Ph.D. in Computer Science from Imperial College, London. You can contact the author at colin.atkinson@uni-mannheim.de.