Systemic Framework for Enterprise Architecture & Transformation

Multilevel Modeling

Introduction

- This document is an integral component of the SysFEAT architectural framework. It provides foundations to address the challenges posed by Enterprise Architecture in the 21st century, which include :
  - Increasing complexity in system structures and behaviors.
  - Growing intricacy in architecture, management and governance of these systems.
  - The mission of the framework is to demystify these complexities, ensuring they are comprehensible to a broad audience, thereby facilitating the design and management of complex-systems across all scales, from micro-systems to enterprise level systems.

- Enterprise Modeling refers to the overarching language and conceptual framework used to describe, understand, and communicate the complex structures and dynamics of an enterprise and its sub-systems.

- The following slides present the **theoretical foundations** used by SysFEAT to establish **extensible multilevel modeling** ontologies.

Introduction – a formalized theory

- All SysFEAT formalizations are expressed in Agda, a dependently typed functional programming language and proof assistant that serves as a formalization language, where types are first-class citizens and programs are synonymous with mathematical proofs.
- Agda provides a highly expressive foundation for SysFEAT's ontological requirements by leveraging its advanced dependent type theory:
  - Enforces the construction of local predicates through ∑-types and dependent records, which intrinsically index properties to specific contexts and prevent the trap of uncontextualized global definitions.
  - Enables multi-level classification, thanks to its polymorphic universe hierarchy (e.g., Set u or Type u), naturally modeling powertypes where concepts at one level mathematically instantiate those at a higher universe level without the need for cumbersome wrapper structures.
  - Handles the complex reflexivity required when powertypes act as subtypes of their own powerinstances.

The world is higher-ordered

The major problems in the world are the result of the difference between how nature works and the way people think."

— Gregory Bateson,

Why multilevel Modeling?

- Most modeling frameworks (UML, OWL, ER) impose a rigid two-level architecture: classes and instances.
- This forces all domain knowledge into a single flat layer of types.
- Real domains have entities that are simultaneously types and instances: Lion is an instance of Species and a classifier of individual lions.
- Aristotle already distinguished primary substances (individuals) from secondary substances (species, genera) — a proto-multi-level scheme.
- Russell's Type Theory (1908) formalized the insight: a class is of a higher logical type than its members; confusing levels produces paradoxes.
- Modern higher-order metaphysics argues that properties *of* properties require genuine higher-order quantification, not first-order workarounds.

Existing multi-modeling initiatives

- Aristotle already distinguished primary substances (individuals) from secondary substances (species, genera) — a proto-multi-level scheme
- Russell's Type Theory (1908) formalized the insight: a class is of a higher logical type than its members; confusing levels produces paradoxes
- Modern higher-order metaphysics argues that properties *of* properties require genuine higher-order quantification, not first-order workarounds
  - Example: Table -> Red -> Color

SysFEAT — Multi-Level by Construction

- SysFEAT uses Agda's universe polymorphism as the native backbone for multi-level classification
- Element u = Set u → ClassOfElement u = Set (lsuc u) → PowerType u = Set (lsuc (lsuc u)) — the level hierarchy is the type system itself
- Linkage (proof-relevant fibered predication) works uniformly across levels — no need for ad-hoc cross-level relation mechanisms
- Verification is built in: Agda's type checker enforces structural soundness that ML2 must validate externally

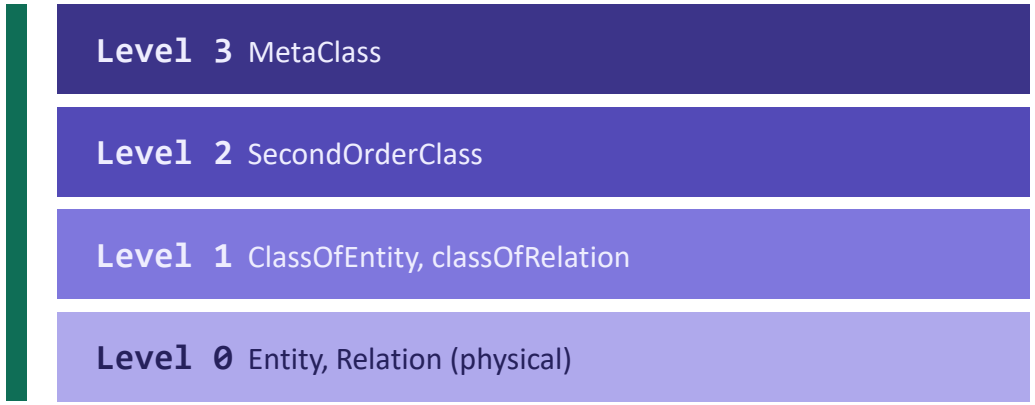
Beyond Strict Stratification — Why Reflexivity Matters?

- Russell's type theory (1908) imposed rigid stratification: no type can classify itself — a safeguard against paradoxes
- MLT* partially loosens this with "orderless types" but still treats self-classification as exceptional
- Real knowledge structures are inherently reflexive: "Type" is itself a type; "ClassOfElement" classifies elements including itself
- Non-well-founded set theory (Aczel, Barwise) shows that controlled circularity is mathematically coherent and necessary for modeling circular phenomena
- SysFEAT achieves reflexive powertypes within a consistent framework: universe polymorphism provides the safety net that Russell's ban was designed to provide, without the rigidity

Syntactic anchoring via Agda universes

Universe polymorphism for cross-layer patterns, before any ontological classification

Universe level stack




```
Linkage {u v w} (S : Element) (T : Element)
-- agnostic to ontological nature of S, T
```

Linkage operates on raw Set u across any level. Ordered, Mixed-Order, or abstract — it does not matter.

OrderedClass, MixedOrderClass and multilevel

Introduced after the spatial topology is already defined. They inherit Linkage and Nesting automatically.

Core mechanisms

1. **Universe polymorphism**
   `Linkage {u v} (S : Set u) (T : Set v)` makes the fiber bundle entirely agnostic to the ontological nature of S and T.
2. **Pre-ontological foundation**
   Spatial rules (Nesting injectivity) are universal syntactic laws. They apply regardless of whether entities are abstract, physical, or hybrid.
3. **Universal pattern application**
   The same Nesting pattern automatically applies to Ordered entities (hardware hierarchies) and Mixed-Order entities (architecture blending physical and abstract roles).

Architectural takeaway

Decouple *how things connect* from *what things are*.

By pushing the definition of Linkage and Nesting down to Agda's syntactic level (Set u), topological proofs are written once. The compiler then guarantees its absolute consistency across every ontological layer of the framework.

The benefit of locality for multi-level modeling

- Multi-level models involve levels authored by different people, evolving at different rates (domain level vs. meta-level vs. foundational level)
- In Cartesian-product semantics, adding a relation type potentially affects the entire product space — no structural isolation
- In fibered semantics, a new Linkage between two modules only affects the fibers over connected elements — local change, local impact
- Reflexive structures (meta-level classifying itself) are only safe if contained — a change in a domain-level type must not break meta-classification
- Extensibility (adding a new classification level — e.g., 3rdOT) should be local: adding a new order doesn't require re-validating the entire structure