Systemic Framework for
Enterprise Architecture & Transformation

Modular Ontologies:
The principle of Locality

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 modular ontologies: the predicate substract.

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 power-instances.

Modularity begins at syntax

Complexity is good; It is confusion that is bad

Don Norman : The DESIGN of EVERYDAY THINGS

The need for proper syntax: scientific foundations

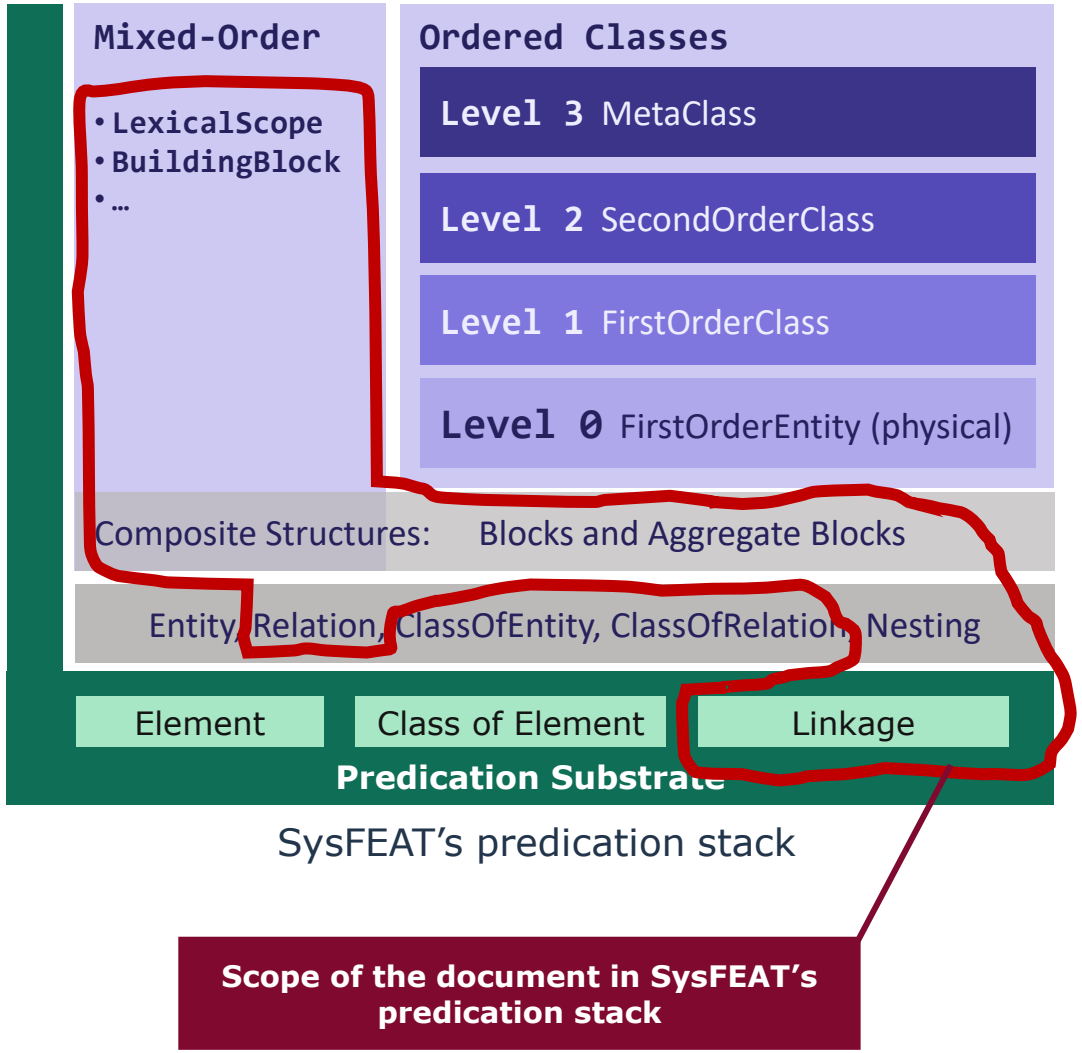
- Holland, John H.. Signals and Boundaries: Building Blocks for Complex Adaptive Systems (The MIT Press) (p. 36).
  - *Typically, the rules of deduction are drawn from symbolic logic, in which the rules manipulate symbols without reference to the interpretation or the meaning of the symbols. That is, the manipulations are syntactic, depending only on the arrangement of the symbols. <..>*
  - *This syntactic approach comes close to being a sine qua non for theoretical science. Matters of speculation and interpretation are moved from the argument back to the premise.*

- The need for modularity is not only an ontology concerns, but rather an architectural necessity:
  - In business and technology, this is reflected in Modularity Theory, which explains product adoption and market performance, as championed by the Christensen Institute.
  - https://www.christenseninstitute.org/theory/modularity/

Scope and structure.

- SysFEAT's predication architecture spans multiple layers, from syntactic primitives up to multi-level classification.
- Multi-level classification — instanceOf, subTypeOf, powerTypeOf — is addressed separately in SysFEAT Theoretical Foundations: SysFEAT-TheoraticalFoundations-MultiLevelModeling

- This document focuses on how the principle of locality is applied, progressively, at each layer of the predication stack:
  - **Predication Substrate** — the Linkage as a local, fibered predicate.
  - **Relations in Entities** — relations are local to the entities they connect, not global assertions over the entire graph.
  - **Lexical scope of Entities** — entities define a local namespace; what is visible is determined by structure, not by convention.
  - **Building blocks and Composite structures** — locality scales up: composites internalize their parts and connections, preserving local reasoning at every level of aggregation.

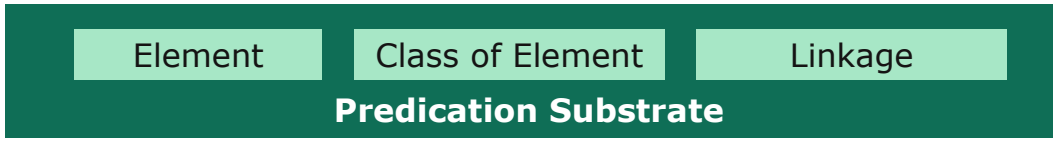


SysFEAT's predication stack

The predication substrate

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over."

Christopher Alexander, *A Pattern Language* (1977):



Global versus Local predicates

- **The Traditional View is Global:** In standard models, a relation is a global bridge existing independently between two elements.
- **The Fibrational Shift:** A "natural fibration" turns this inside out. Instead of the link existing in a global space, the link is mathematically embedded directly *inside* the source element.
- **A Structural Primitive:** In this architecture, Linkage serves as the structural primitive of predication. It is a local, proof-relevant predicate rather than a disconnected arrow.
- **Resolving Polysemy:** This approach gives the link a triple nature—it acts simultaneously as a graph edge, a verifiable predicate, and a thing in its own right.

The Structural Flaws of Global Predicates

- Absence of locality:
  - Because relations are modeled as global bridges existing independently between two elements, proving that element A is linked to element B requires querying a global registry rather than inspecting the element itself.
- Polysemy between things-that-are and things-that-connect:
  - The same syntactic form — $P(x)$, $R(x, y)$, $S(x, y, z, \ldots)$ — is used indiscriminately for elementary predication (classifying an element, linking two elements), for complex predication (composite structures, connectors, n-ary dependencies), and for the entities themselves (e.g., tables in relational databases).
- Lack of lexical scope: any portion of the ontology may depend on other portions arbitrarily far away
  - When links are external and global, introducing a new relation implicitly mutates the structural footprint of an element across the entire system. It is akin to a dictionary where the structural definition of a word arbitrarily shifts depending on the page you are reading, destroying any reliable local encapsulation.
- Conclusion
  - The notion of scope — understood as existential, semantic, and contextual locality— cannot be expressed intrinsically and must be reconstructed through auxiliary mechanisms, such as Templates in ISO-15926.
  - The scope problem is mathematically and structurally insoluble within the pure global-predicate paradigm.

Predication Substrate: the foundation for scoped statements

- SysFEAT moves away from representing relationships as subsets of a Cartesian product ($R \subseteq S \times T$).

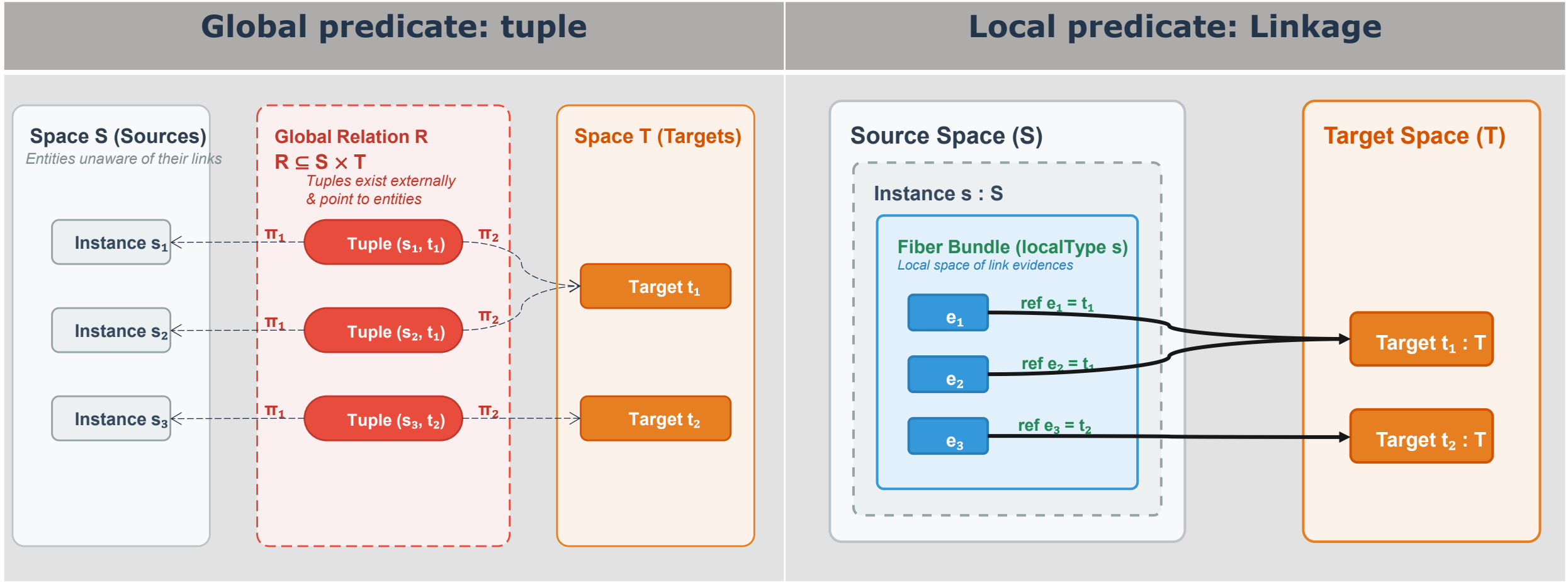
- The Predication Substrate studies *how* predication works structurally :
  - What a predicate is made of,
  - How predicates compose,
  - How predicates classify and are classified.

- The Element/Linkage layer is a **theory of relation** as such — a formal account of what it means to be a link, prior to and independent of what the link connects. It provides the internal structure of relating (the fibered decomposition into local evidence and target projection via Linkage).

The Predication Substrate layer: definition

- The Predication Substrate defines the compositional system of Element, Class of Element, using a principle of local predicates – Linkage - from which SysFEAT's ontological structures are grown.
- Higher-level domains—like Entity and Relation, Blocks and Aggregates —simply inherit these principles. This ensures that the entire ontology shares a single, mathematically verified predication engine.
- Linkage is the structural primitive of predication. It is a local, proof-relevant predicate rather than a disconnected arrow..
- The triple nature of Linkage — edge, predicate, element — is what resolves the polysemy that traditional frameworks impose between things-that-are and things-that-connect.

Moving from Global Edges to Embedded Local Links



Moving from Global Edges to Embedded Local Links

- Imagine a traditional Entity-Relationship (ER) model.
  - If you have “Orders" (Source) and “Products" (Target), the relationship “ordered products" is usually a third, separate table (or a set of free-floating edges in a bipartite graph). The entities themselves are "blind" to their connections; the database has to look at the global “ordered products" table to figure out who connects to whom.
- A **natural fibration** fundamentally changes this topology.
  - Instead of creating a third, global “ordered products" table, you give every single Order a mathematical "backpack" (this is the *fiber*).
  - If an Order is about a specific Product, the contract (the *evidence*) is placed directly into their backpack. To find out the ordered product, you don't query a global graph;
  - You simply open the Order's backpack, pull out the contract, and read the destination address (the *projection*). The edge is no longer a separate line on a whiteboard; it is a physical item embedded strictly within the source's local space.

How localType and ref capture topology

- **The Local Space (localType):** A Linkage assigns a private, local type of evidence to every single source element.
- **The Projection (ref):** Once an evidence exists within the source's local space, a projection determines the specific target from that evidence.
- **Proof Relevance:** To prove that element A is linked to element B, you don't look at a global registry. You look inside A for a specific piece of evidence (a witness), combined with a proof that this evidence projects to B.
- **True Composition:** Because links are embedded entities, they can be mathematically composed; the internal evidence of one link can seamlessly chain into the evidence of another.

```
record Linkage {u v w} (S : Element) (T : Element) :
               Set (lsuc (u ⊔ v ⊔ w)) where
  field
    localType : S → Set u
    ref       : {s : S} → localType s → T
```

Linkage: mathematical origins

- The term **Fibration** originates from **Algebraic Topology** and **Category Theory** (specifically Fiber Bundles and Grothendieck Fibrations).
- In topology, a *fiber bundle* consists of a total space E, a base space B, and a continuous surjective map $\pi : E \to B$.
  - For every point x in the base space B, the inverse image $\pi^{-1}(x)$ is called the "*fiber*" over x.
  - It intuitively means that above every point in a space, there is another localized space "attached" to it.
- In SysFEAT, the source set S acts as the "base space". Instead of drawing flat arrows between sets, you construct a bundle of fibers over *S*. Each element s in S has a fiber hovering right above it, containing all the outgoing connections belonging exclusively to s.

Linkage: summary

A linkage is a natural fibration for embedding links into their source

Agda type definition

```
Element : (u : Level) → Set (lsuc u)
Element u = Set u

record Linkage {u v w} (S : Element) (T : Element) :
               Set (lsuc (u ⊔ v ⊔ w)) where
  field
    localType : S → Set u
    ref       : {s : S} → localType s → T
```

The three fields

`label` Human-readable name identifying the relation.

`localType` For each source s, defines the local fiber: the set of possible links originating at s.

`ref` Projects each fiber element to a target in T. This is the actual connection.

Core principle

- A link is not a global arrow from S to T.
- It is a Fiber Bundle: each source element s carries its own local space of possible links.
- The edge only exists within the fiber above s.

Mathematical reading

Total space $E = \Sigma(s:S).\text{localType } s$

Projection $\pi : E \to S$ (via fst)

Target map $\text{ref} : E \to T$ (via snd)

The fiber above s is $\pi^{-1}(s)$ = localType s.
Each fiber element maps to one target via ref.

PS: the "u" and "lsuc u" expressions are explained in the multi-level modeling document

Locality of Relations in Entities

Relations are local to their source entity

Relation in SysFEAT - overview

- Relation is a Linkage: edge data lives in the source entity's local fiber (localType s), not in a global relation set — every relation is bound to its source by construction.
- Relations on relations use the same Linkage mechanism one universe level up: a Linkage whose source and target are themselves Linkages, which are already Elements at a higher level.
- This avoids the UML / KerML / ISO 15926 confusion, where Relationship is declared a subtype of Entity — collapsing things that are and things that connect into a single flat hierarchy.
- In SysFEAT, a Relation is not an Entity pretending to be an edge. It is a Linkage that is an Element, at a higher universe level — classifiable and specializable without losing its connecting nature.

Relations are built on Linkage

From fiber bundles to typed, classified connections between entities

Definitional equivalences

```
-- M0 (physical instances)
Relation s t = Linkage s t

-- Mx (class level / metamodel)
classOfRelation cs ct = Linkage cs ct
```

Two-level architecture

- Linkage is the single universal mechanism.
- At M1, it connects classes of entities (classOfRelation cs ct).
- At M0, it connects physical entities (Relation s t).
- Same fiber structure, two abstraction levels.

Promotion to sets

```
RelationSet rel = Σ s (λ x → localType rel x)
```

The total space of the fiber bundle becomes a first-class Set, enabling classification of M0 relations against M1 schemas.

M0 → M1 classification

```
instanceOfRel rel crel =
  instanceOf (RelationSet rel)
```

A physical relation (M0) is classified against its class (M1) by comparing their promoted sets. This bridges the two Linkage levels.

Structural laws on fibers

isFunctionalLink At most one target per source (deterministic).

isTotalLeftLink Every source element has at least one link (totality).

Relation of relation: RelLinkage

Linkage applied recursively — relating relations without confusing levels

Step 1 — Promote relations to sets

```
ClassOfRelationSet rel = Σ cs (λ x → localType rel x)
```

```
RelationSet rel = Σ s (λ x → localType rel x)
```

A relation (itself a Linkage) is promoted to a first-class Set by collecting its total space. This Set lives in the entity domain, enabling Linkage to act on it.

Step 2 — Apply Linkage on promoted sets

```
subTypeOfRel subRel superRel =
  subTypeOf (ClassOfRelationSet subRel)
            (ClassOfRelationSet superRel)
```

```
baseRel withRelAspect aspect = subTypeOfRel baseRel aspect
```

Both operators are themselves Linkages (via subTypeOf), applied on the entity-domain projections of relations.

No confusion principle

Entity domain S, T, cs, ct are elements/classes.

Relation domain Linkage structures between them.

To relate two relations, we never mix domains. Instead, we promote the relation to a Set (via Σ), then apply Linkage on those promoted Sets.

The recursive pattern

1. Two relations R_1, R_2 exist as Linkages between entity classes.
2. Each is promoted to a Set via Σ (ClassOfRelationSet).
3. A new Linkage (subTypeOf) connects those Sets, creating a typed relation-of-relation.

Linkage → Set → Linkage: the same mechanism at every level, domains never crossed.

Locality at the Entity Level

Lexical scope for nested entities

Nesting of Entities : overview

- If standard Linkage is about visibility, Nesting is about containment. By enforcing topological injectivity rather than relying on heavy data structures (like ∑-types), SysFEAT maintains an elegant, metaphysical universe while rigorously enforcing complex containment trees.
- Core Mechanisms:
  - Lexical Scope: aspect of entities that acts as an absolute container for its contents.
  - Spatial Exclusivity (Strict Injectivity): The reference mapping (ref) becomes mathematically injective. A target entity T can only be referenced by one unique piece of evidence within one specific fiber of one specific source.
  - Captive Targets: Because of this strict injectivity, the target entity is physically "nested" within the source's local space. It cannot be shared with other sources, and its lifecycle is bound to its container.
  - From Model to Physical (M0): At the instance level (M0), the nestingLink operationalizes this rule, ensuring that components like ‘parts' are strictly captive within their parent ‘aggregates’.

Nesting of Entities - details

Entities embedding entities through LexicalScope-guarded Linkage

The gatekeeper: LexicalScope

```
postulate
  LexicalScope : ClassOfMixedOrderEntity u
  LexicalScope ⊏ₑ ClassOfMixedOrderEntity
```

An abstract class marking any entity that can act as a spatial container. Only subtypes of LexicalScope are allowed as nesting sources.

Nesting constructors (M1 and M0)

```
-- M1 (class level)
classOfNestingRelation (h : cs ⊏ₑ LexicalScope)
  = classOfMixedOrderRelation cs ct
```

```
-- M0 (physical instances)
nestingRelation (h : s ::ₑ LexicalScope)
  = Relation s t
```

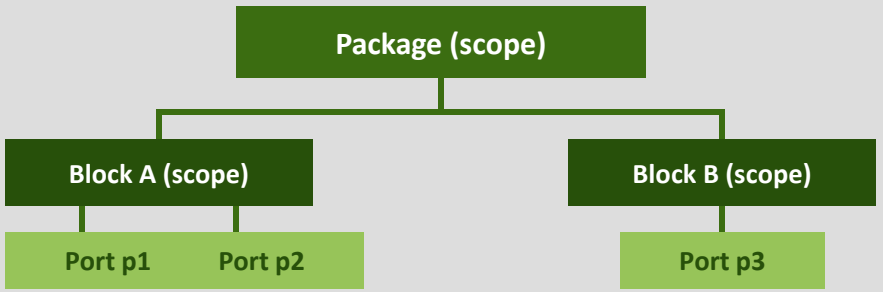
Both are ordinary Linkages. The guard is the proof argument, not a different structure.

Two structural guards

1. **Source guard**
   $cs \sqsubset_e \text{LexicalScope}$
   Only a scope class may own nested children.
2. **Injectivity axiom**
   $\text{ref}(x_1) \equiv \text{ref}(x_2) \to s_1 \equiv s_2 \wedge x_1 \equiv x_2$
   Each nested entity belongs to one container only.

Result: containment trees

The two guards together guarantee a strict tree:



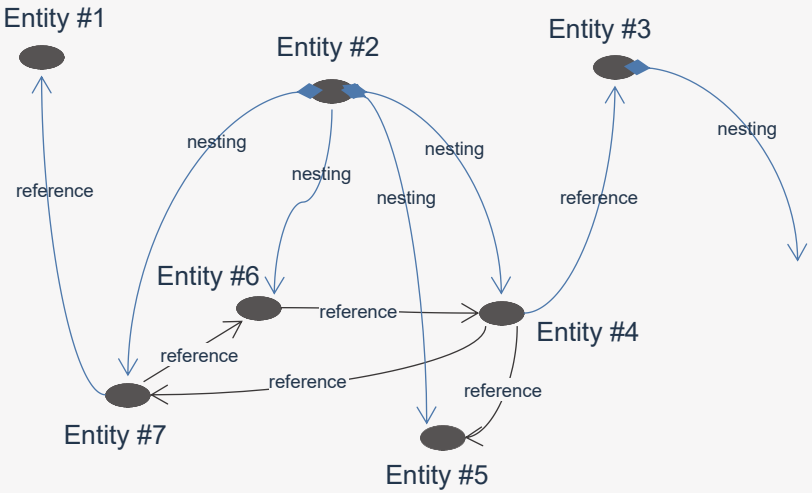
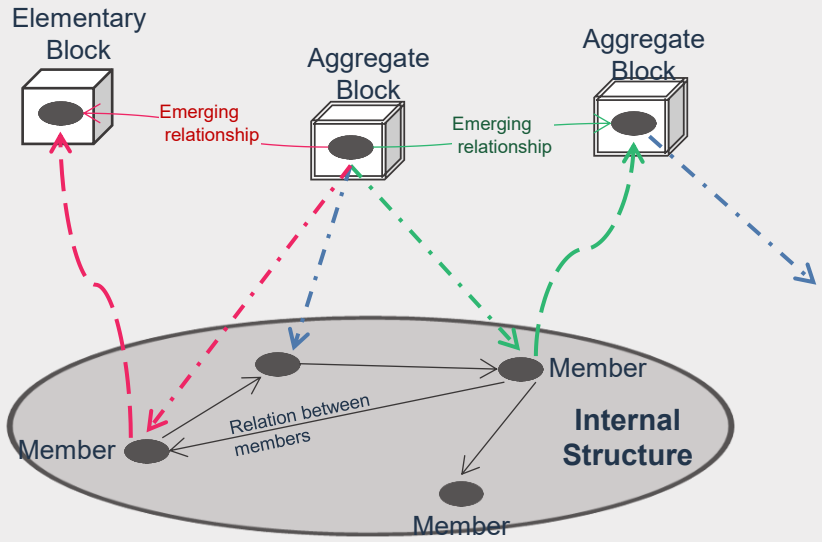
p1 cannot migrate to Block B (injectivity forbids it).

Compositionality - Aggregates

SysFEAT layered approach of relationships

The modularity principles of SysFEAT aims at providing modular connectable structures, using a layered approach of locality.

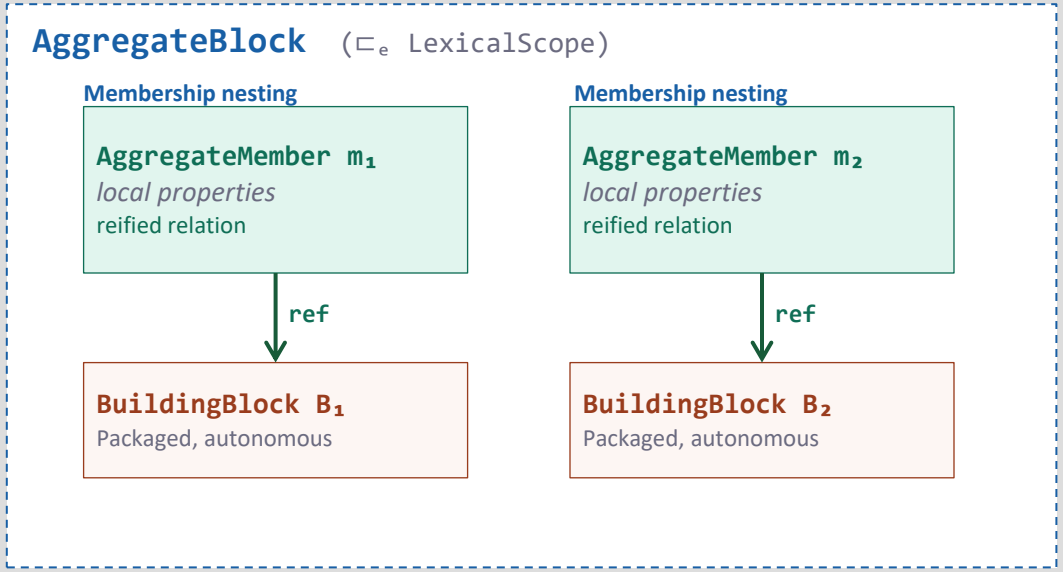
1. Locality of relationships is supported by Dependent Pair Types (∑-types), which embeds and source–target asymmetry as Grothendieck Fibration:
2. Lexical scoping provides the ability to nest entities *(namespace in KerML)*.
3. Compositionality provides dynamic locality for entities *(aggregates in DDD, that and connectors in KerML).*

E/R Knowledge Graph	
Aggregate Graph	

Aggregate block compositionality

Whitebox structure vs. blackbox composition — two views of the same entity

Whitebox — internal architecture



The internal structure is visible: AggregateMembers are nested entities (reified relations) each pointing to an external BuildingBlock via a reference.

```
membershipOfAggregateMember : classOfNestingRelation AggBlock
AggrMember
```

Blackbox — derived relationship

From outside, only the composed relation is visible:

```
aggregateMember =
  membership ∘ aggregation
-- AggregateBlock → BuildingBlock
```

The internal members are erased. Mathematical composition (∘) collapses nesting + reference into a single direct Linkage.

Why the indirection?

The AggregateMember is a reified relation — a real entity, not an invisible arrow. It carries its own local properties and enables emergent structure (attributes, process steps, specializations are all AggregateMember instances).

Same entity, two views: Both are derived from the same fiber structure
Whitebox reveals the nesting + reference architecture;
Blackbox exposes only the composed Linkage.

Aggregate block – dynamic locality

The that operator (1) — recovering the aggregate from within the fiber

The that operator

```
that : ∀ {u v u'} {S : ClassOfEntity u}
                  {T : ClassOfEntity v}
                  {L : classOfRelation S T}
     → (isMember : L ⊏ₐᵣ membershipOfAggregateMember)
     → {s : S}
     → (localEdge : Linkage.localType L s)
     → S
that isMember {s} localEdge = s
```

Given a local edge inside a fiber, that returns the source container s. The implementation is trivial (just return s), but the type signature carries all the structural weight.

The subtype locking chain

```
membershipOfAgentMember     ⊏ᵣ membershipOfAggregateMember
membershipOfBehaviorMember  ⊏ᵣ membershipOfAggregateMember

→ any specialized membership inherits access to that
```

The locking works on the root relation. Every specialized membership (AgentMember, BehaviorMember) must prove it is a subtype of the root, gaining access to **that** automatically.

What the type signature enforces

1. **Membership proof**
   L ⊏ᵣ membershipOfAggregateMember — only relations that are subtypes of the root membership can use that.
2. **Fiber locality**
   localEdge : Linkage.localType L s — the caller must provide a concrete edge living in the fiber above s.
3. **Aggregate recovery**
   The return type is S. You get back the AggregateBlock that owns this member, type-safe.

Architectural insight

Dynamic Locality gives every aggregated block intrinsic, type-safe awareness of its Aggregate.

This makes SysFEAT's aggregation natively suited to complex adaptive systems, where emergent behavior arises from each constituent sensing its position in the containment hierarchy.

(1) The name of the "that" operator comes from KerML but is formalized differently in SysFEAT.

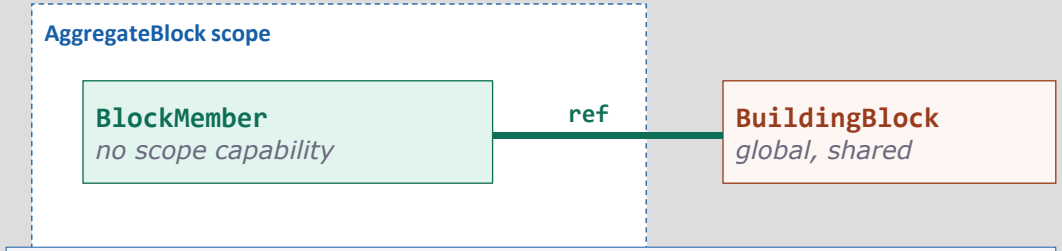
Late binding of Building Block lexical scope

The reified relation (Aggregate Member) decides the scope, not the BuildingBlock

Key architectural insight

The BuildingBlock does not decide whether it is local or global. It is the reified relation (the Member) that, depending on its nature, deploys or not a lexical scope around the targeted block. The mechanism lies in granting **`LexicalScope`** capability to the HierarchicalMember, transforming its aggregation into a true Nesting.

BlockMember — reference aggregation

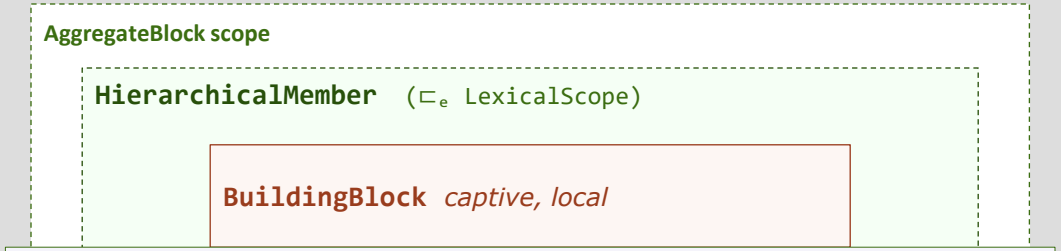



```
blockMemberAggregation :
  classOfReferenceRelation BlockMember BB
```

The BlockMember aggregates by simple reference. The BuildingBlock remains free, shareable across multiple aggregates. Network topology.

Network of building blocks

HierarchicalMember — nesting aggregation




```
hierarchicalMemberAggregation :
  classOfNestingRelation HierMember BB
```

The HierarchicalMember deploys its own LexicalScope around the BuildingBlock, capturing it. The BB becomes local and exclusive. Tree topology.

Hierarchical tree of building blocks

The mechanism: **`HierarchicalMember ⊏ₑ LexicalScope`** transforms `classOfReferenceRelation` into `classOfNestingRelation`. Same relation type, different lexical scope capability.