Systemic Framework for Enterprise Architecture & Transformation

Modularity

Introduction

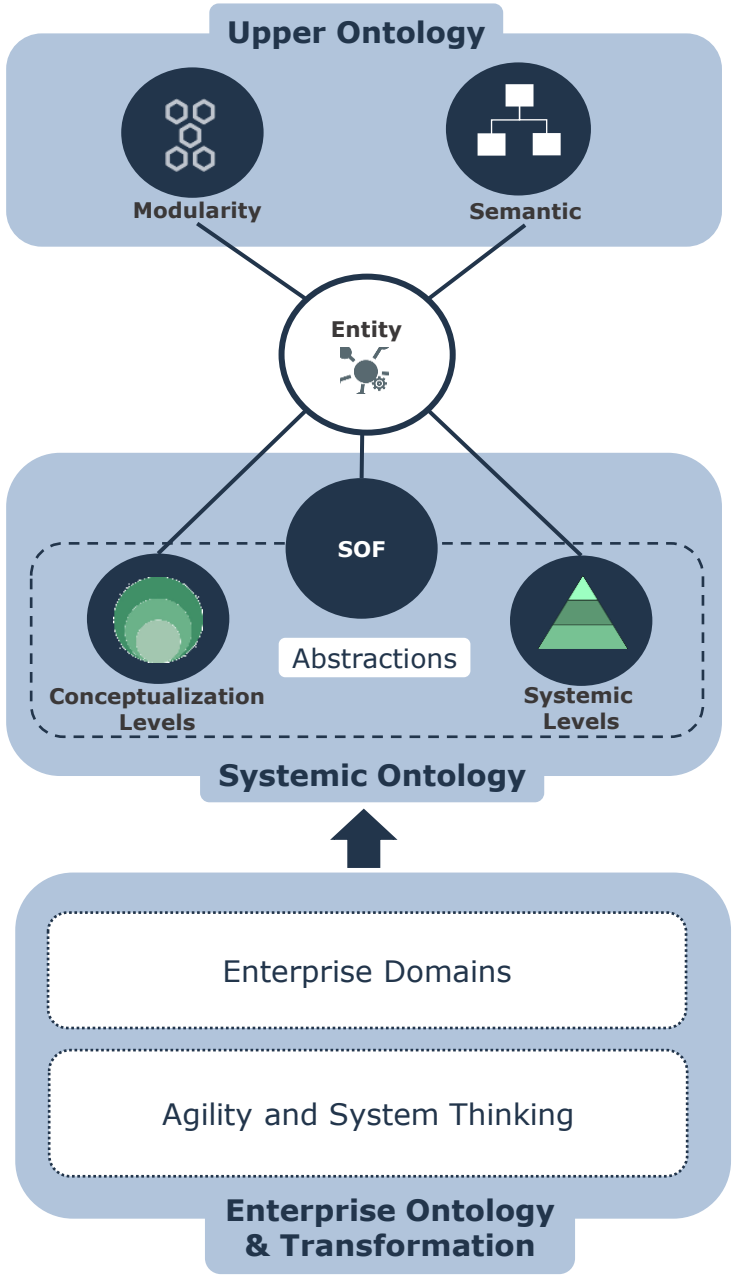
- This document is an integral component of the SysFEAT architectural framework. It provides foundations to address the challenges posed by Enterprise Architecture in the 21st century, which include :
  - Increasing complexity in system structures and behaviors.
  - Growing intricacy in architecture, 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.
- It integrates both the operating aspects of the enterprise (how it functions and interacts within its ecosystem), the transformational aspects (how it evolves and sustains over time through initiatives, asset management) and how these transformations are governed to ensure effectiveness, efficiency and reliability.

- The following slides present the foundations of enterprise modeling.

Foundations of enterprise modeling

- **Modularity** provides the syntax for building robust, manageable, and scalable architectures, based on the principles of compositionality and packaging.
- **Semantic** provides robust capabilities for classifying and composing entities, from time-bound entities (individuals) to families of concepts, enabling effective representation of meaning.
- The **Systemic Operating Framework (SOF)** serves as the overarching language that describes why and how a system operates and interacts within its ecosystems.
- **Abstractions** organizes systems and concepts in degree of abstractions, including systemic levels and conceptualization levels.
- **Enterprise Domains** formalize the various disciplines that make-up EA, ranging from enterprise road-mapping to System ArcDevOps.
- **Agility and System Thinking** ensure that the enterprise evolves and sustains over time through governed initiatives, architected for flexibility and responsiveness in complex and dynamic business environments.

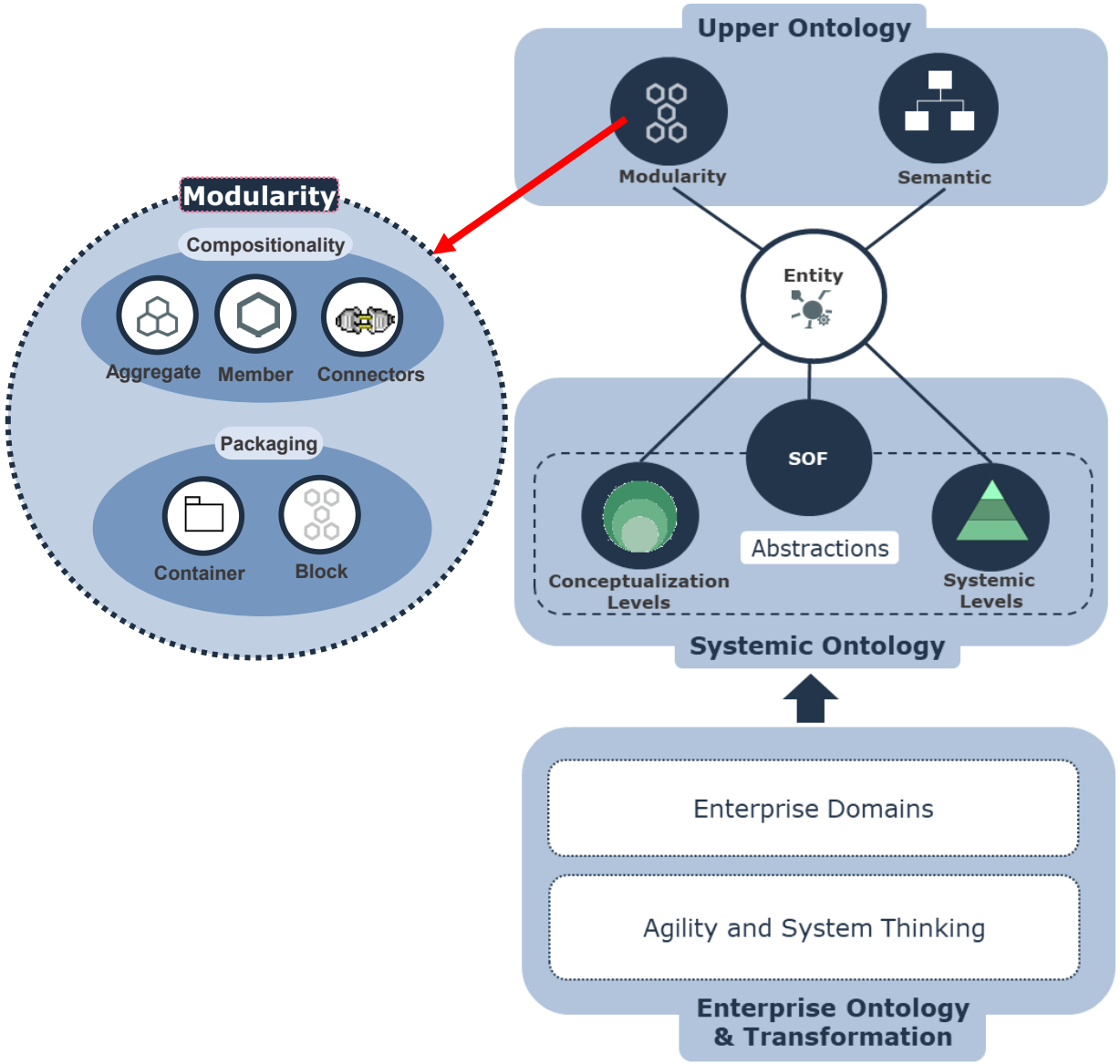


Modularity & Modeling language

- As any language, modeling languages have three aspects:
  - Syntax is the required grammar and punctuation of the language.
  - Semantics is about signification - what do we mean by a Capability?
  - The Systemic Operating Model (SOF) is about operating semantic for Enterprise Architecture.
  - Pragmatics/Architecting has to do with:
    - How to use models (modeling technics).
    - What kind of model to use to address stakeholder concerns (method)
      - ✓ Example: how to use capability modeling in enterprise transformation initiatives.
- This document presents syntactic foundations for developing modular enterprise models.

Modularity in the Architecture modeling landscape

- This document focuses on **modularity** at the **syntax level**, which is grounded on two complementary aspects: *compositionality* and *packaging*.
- **Compositionality** is the ability to assemble entities to form bigger constructs called aggregates.
  - compositionality is a **syntactic** concern that does not carry inherent semantic meaning.
  - It can be applied to both semantic relationships of composition and typology.
- **Packaging** is the ability to group autonomous-reusable building blocks in **modules** also called Packages.
- These two disciplines come hands to hands but shall not be confused.



Compositionality: Why?

Complexity Is Good;

It Is Confusion That Is Bad

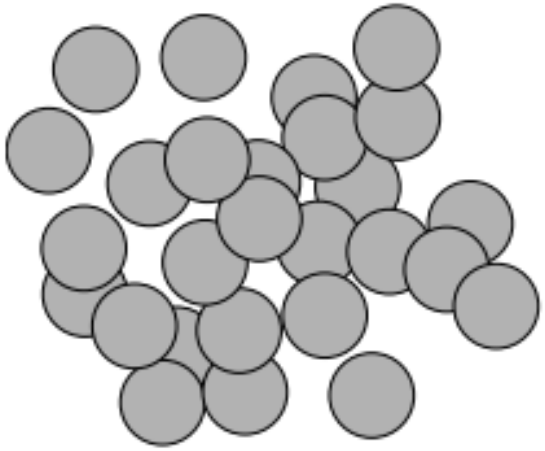
Don Norman

The DESIGN of EVERYDAY THINGS

Modularity benefits – Don Norman illustration

- Source: Don Norman – Living with complexity - page 236
  - *Count the circles simply by looking at them: don't use your hands or a pointer to help. Difficult, isn't it?*

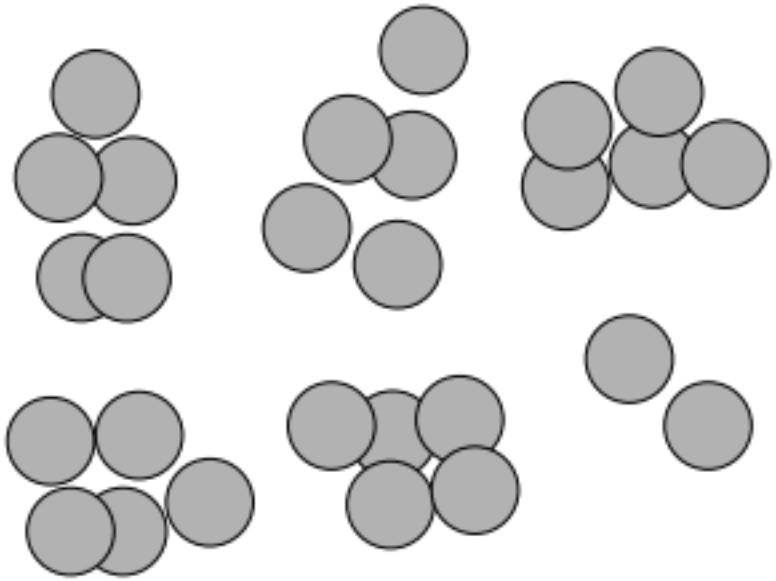
Figure 8.3



Modularity benefits – Don Norman illustration

- Source: Don Norman – Living with complexity - page 236
  - *Now count the very same items shown in figure 8.4, again without using hands or other objects as aids: much easier, isn't it?*

Figure 8.4



Modularity – The need for syntax and boundaries ….

- Extracts from James Joyce : Molly bloom’s soliloquy (Ulysse)

Just a list of words: no more separators for sentences

  - *<..> because they’re so weak and puling when they’re sick they want a woman to get well if his nose bleeds you’d think it was O tragic and that dying looking one off the south circular when he sprained his foot at the choir party at the sugarloaf Mountain the day I wore that dress Miss Stack bringing him flowers the worst old ones she could find at the bottom of the basket anything at all to get into a mans bedroom with her old maids voice trying to imagine he was dying on account of her to never see thy face again though he looked more like a man with his beard a bit grown in the bed father was the same besides I hate bandaging and dosing when he cut his toe with the razor paring his corns afraid he’d get bloodpoisoning but if it was a thing I was sick then we’d see what attention only of course the woman hides it not to give all the trouble they do <..>*

- The first reading of such a text usually leads to a feeling of disarray along with the beginning of a headache. This reading exercise shows how much we need to define the boundaries of things to be able to understand them.

The need for syntax : symbols ….

- For syntax to exist, there is a need for discontinuous signs so that words (symbols) can be identified and then put into sentences.

Just a list of letters no more separators for words.

  - <..>
    becausethey'resoweakandpulingwhenthey'resicktheywantawomantogetwellifhisnosebleedsyou'dthinkitwasOtragicandthatdyinglookingoneoffthesouthcircularwhenhesprainedhisfootatthechoirpartyatthesugarloafMountainthedayIworethatdressMissStackbringinghimflowerstheworstoldonesshecouldfindatthebottomofthebasketanythingatalltogetintoamansbedroomwithheroldmaidsvoicetryingtoimaginehewasdyingonaccountofhertoneverseethyfaceagainthoughhelookedmorelikeamanwithhisbeardabitgrowninthebedfatherwasthesamebesidesIhatebandaginganddosingwhenhecuthisoewiththerazorparinghiscooningbutifitwasathingIwassickthenwe.
    <..>

- Hoffmeyer and Emmenche - Code-Duality and the Semiotics of Nature:
  - *According to Gregory Bateson information is based on difference. A sensory end organ is a comparator, a device which responds to difference. While reading this, for instance, your eyes do not respond to the ink, but to the multiple differences between the ink and the paper.*

The need for 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.*

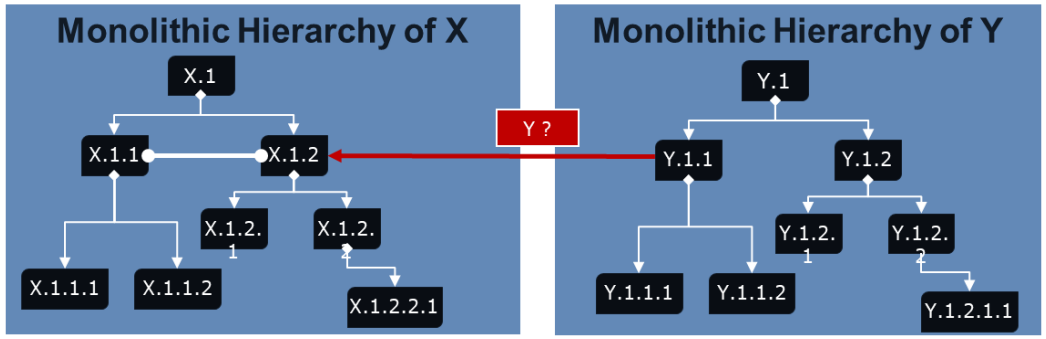
Architecture & compositionality

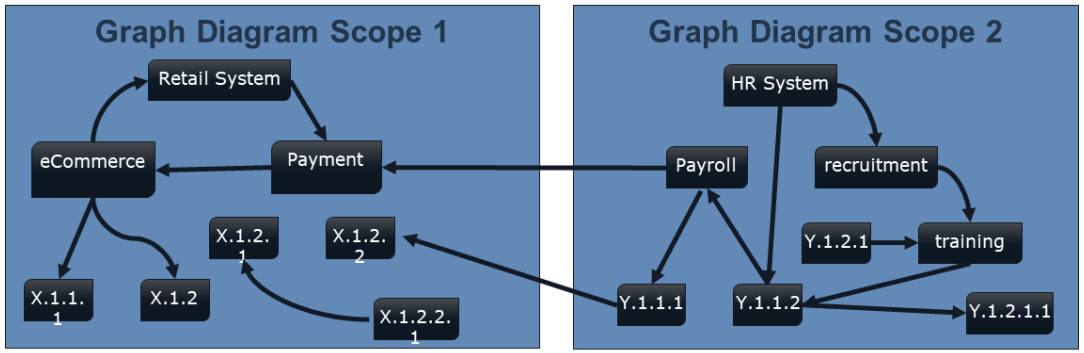
The issues of current frameworks

What is the problem ?

- To support effective development, management and transformation of enterprises and their systems architecture models must themselves be modular to deliver the following services:
  - Be able to build architecture alternatives.
  - Be able to manage catalogs/packages of reusable building blocks.
  - Be able to compare alternative architectures.
  - Be able to guide enterprise transformation, involving time and space perspectives.

- Problem:
  - The two common modeling syntaxes used for architecture descriptions – *monolithic hierarchies and flat graphs* - prevent from creating effective **building block boundaries**, thereby denying the notion of building block itself.
  - Without effective scoping principles, model-driven architecture & management cannot successfully help in designing complex adaptive systems while ensuring associated quality/security assurance.





Problem 1 : monolithic hierarchies & interconnections

- Benefits of monolithic hierarchies.
  - They follow the usual breakdown practice (Cartesian approach).
  - They provide hierarchical scope for building blocks. This sometimes represented by naming conventions, such as, “X.1.1” and “X.1.2” are in “X.1”. IDEF notations are a good illustration of monolithic hierarchies.

- Issues: monolithic hierarchies hardwire building blocks together:
  - Blocks can only be part of a single hierarchy: the single parent syndrome.
  - If multiple parent-relationship is allowed, inter-connections become undefined : the many to many relationship syndrome (see next slide).

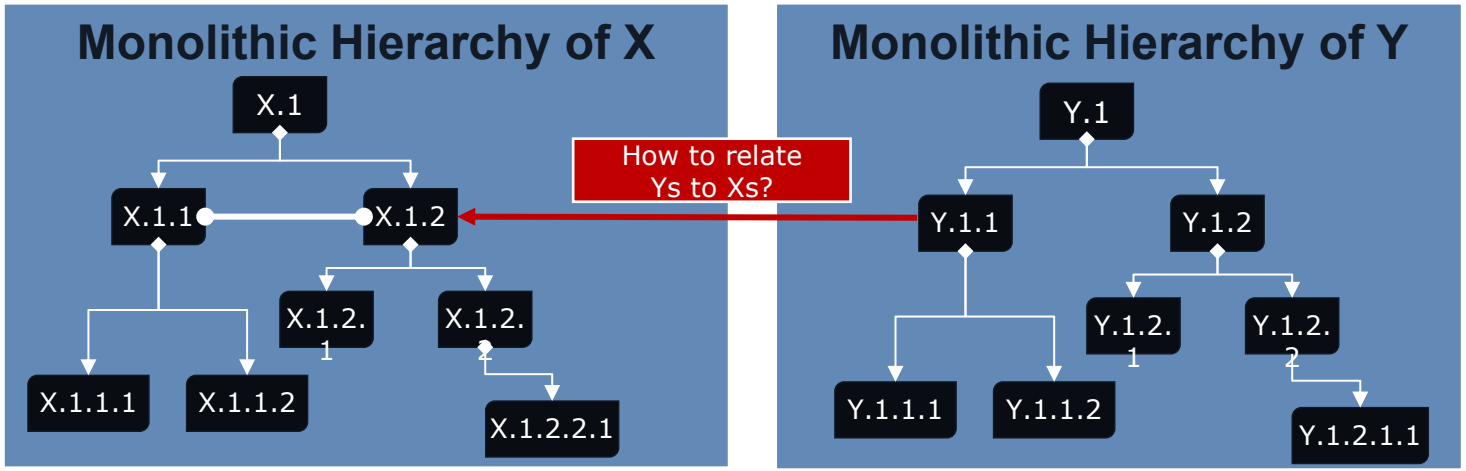
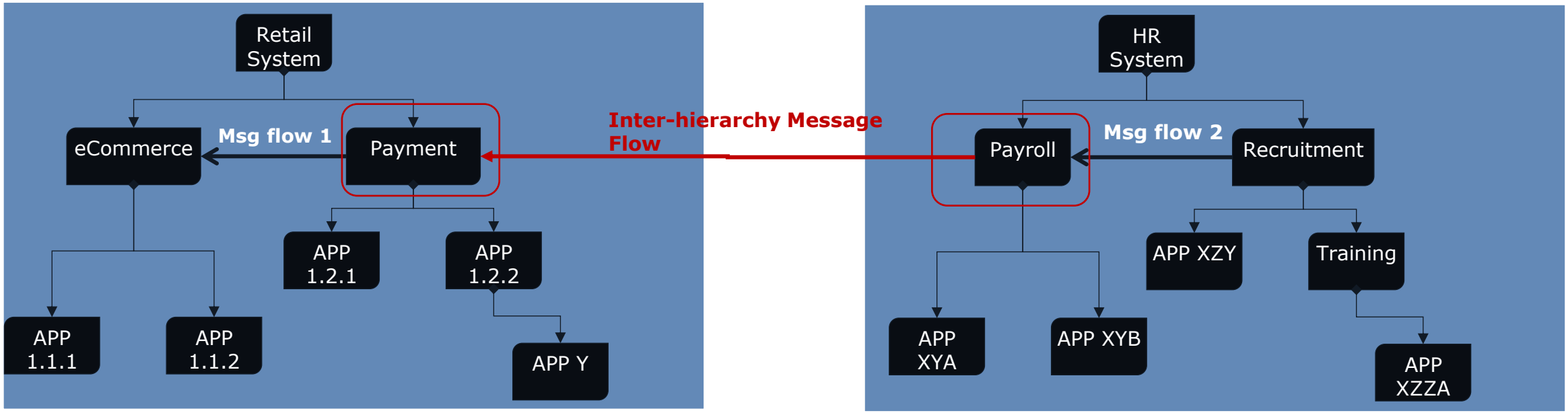


Illustration : monolithic hierarchies & interconnections

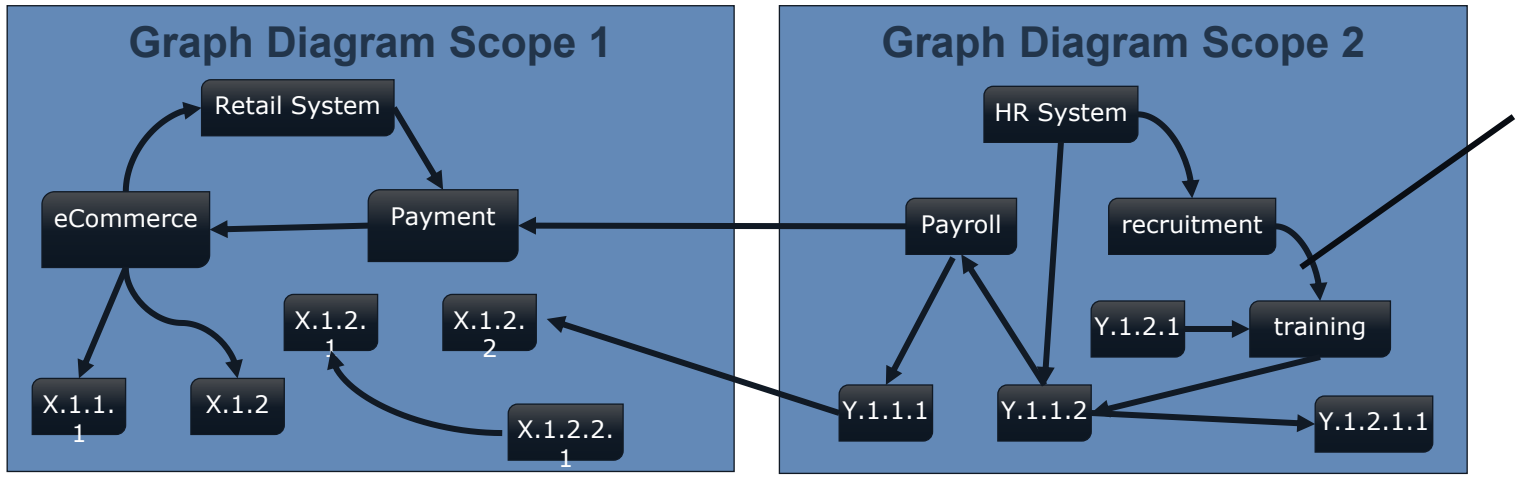
- Let's consider two monolithic application hierarchies:
  - Retail-System (on the left below) and HR-System. (on the right below).
- If *Payroll* (from *HR System*) needs to send a message to *Payment* (from *Retail System*), does this implies that:
  - *Payment* becomes part of the *HR-System* hierarchy ?
  - or that The *HR-System* depends on the *Retail-System* (cross hierarchy dependency)?
- Similar issues occur on sequences between processes, flows between processes, etc. Strict hierarchical scope prevent from having reusable, autonomous building blocks.
- The benefit of autonomous monolithic hierarchies is lost because of the need to connect multiple hierarchies.



Problem 2 : flat graphs – non-local relationships

- Benefits of flat graph models:
  - They avoid the single parent, single scope syndrome of monolithic block hierarchies.
  - They enable a natural discovery of unitary building blocks and their dependencies through story boards. For instance, may architects look at messages between software systems or events and commands through event storming or ArchiMate models.

- Issues:
  - Building Block assemblies (aggregates) have been lost: there are no more scoped relationships but a single global graph (*is Payment in HR system?*).
  - Adding a relationship at one end of the graph has undefined effects on the rest of the graph, hence building blocks do not have an autonomous definition.
  - Diagrams are often used for creating pseudo system boundaries. As mere pictures, diagrams do not provide an explicit definition of system-boundaries. We are back to Visio (See EPC & ArchiMate below)!

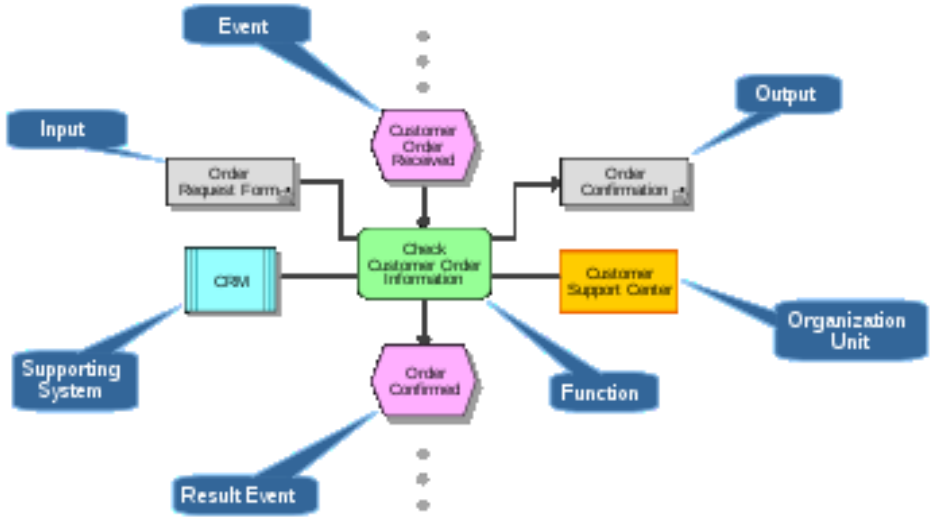


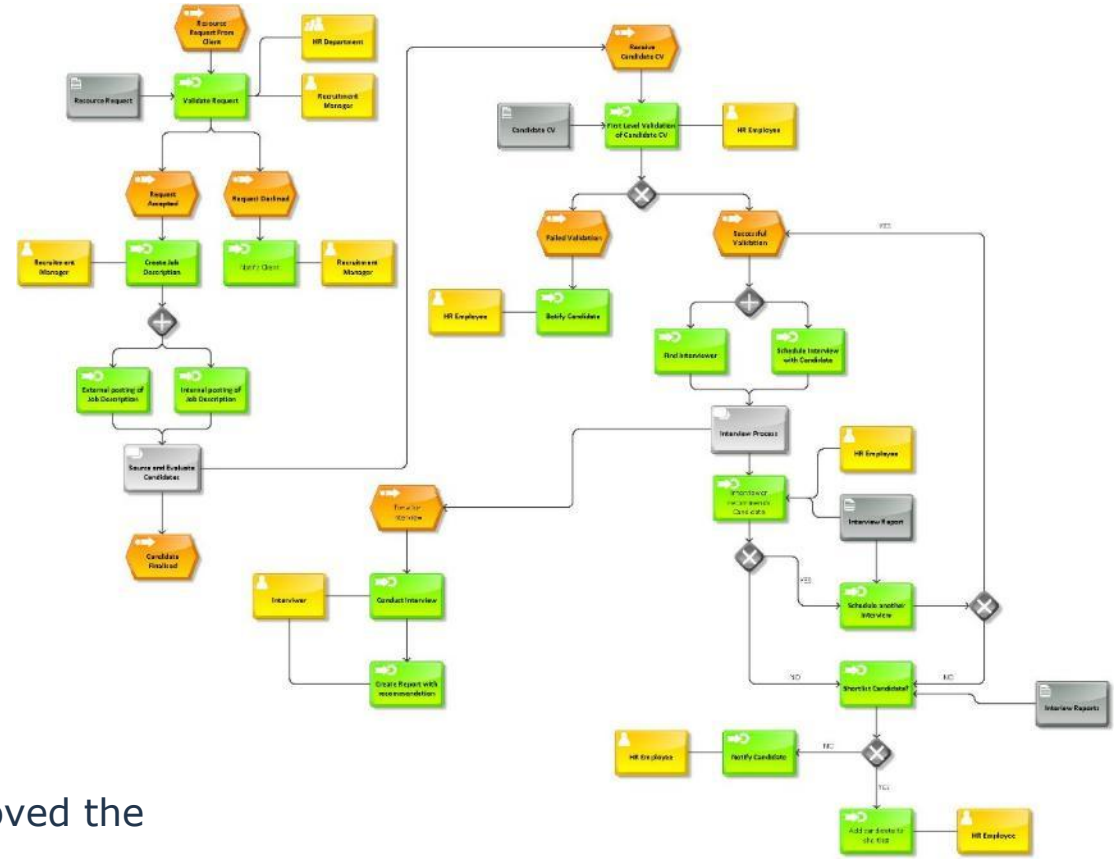
What is the impact of adding this connector ?
- Is *Recruitment* changed ?
- Is *Training* changed ?
- both ?

What is the change impact scope ? Recruitment, HR System, … the entire graph ?

Problem 2: flat graphs – the case of EPC models

- EPC Models - Extract from Wikipedia (2008):
  - *<..>Unfortunately, neither the syntax nor the semantics of EPC are well-defined.[1] EPC requires non-local semantics,[2] so that the meaning of any portion of the diagram may depend on other portions arbitrarily far away.<..>*



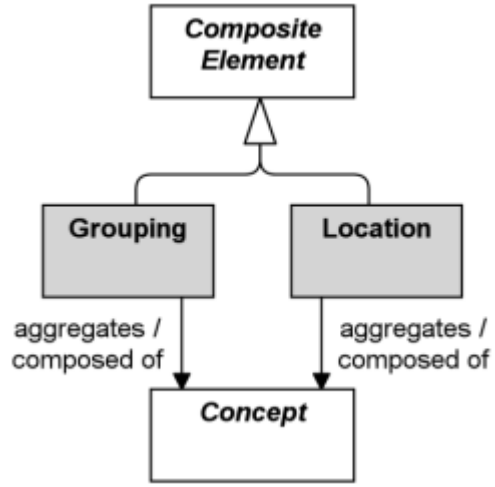


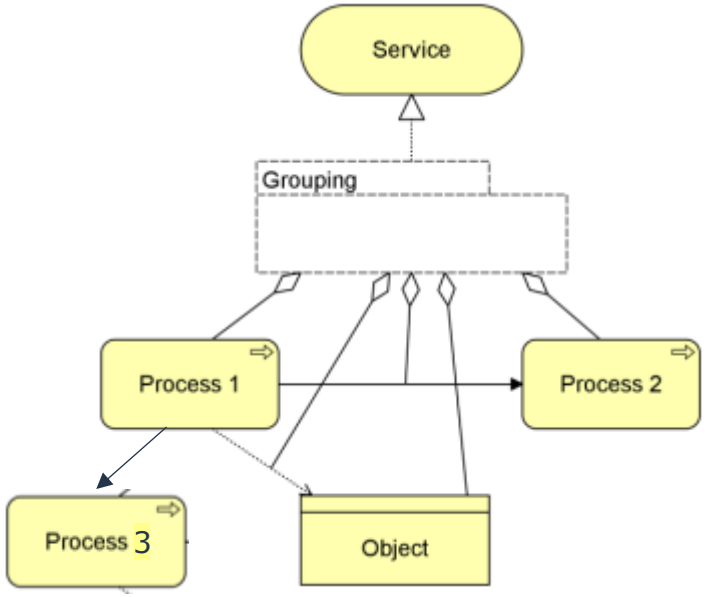
Notes:

1. Full document available here
2. Note that the last revision of EPC on Wikipedia has removed the reference to non-locality ….

Problem 2: flat graphs – the case of ArchiMate

- ArchiMate faces similar issues. It has introduced an intuitive mechanism to contextualize relationships, called *grouping* - with the following definition:
  - *The grouping element is used to group **an arbitrary group of concepts** (elements and/or relationships), which can be of the same type or of different types. The aggregation relationship is used to link the grouping element to the grouped concepts.*
- The term 'arbitrary' introduces significant ambiguity. This lack of precision allows ArchiMate experts to interpret concepts in a nearly unlimited number of ways. The challenge is compounded when defining relationships between groups of elements, where consistent rules are most critical."
- The diagram on the right reveals the issue of non-local relationships:
  - The relationships between *Process 1*, *Process 2* and *Object* need to be contextualized for the relationship with *Service* to be established, excluding the relationship from *Process-1* to *Process-3*.





Compositionality Principles

SysFEAT layered approach of relationships

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

1. Locality of relationships is supported by Kuratowski ordered pair, which embeds order and source–target asymmetry inside set theory:

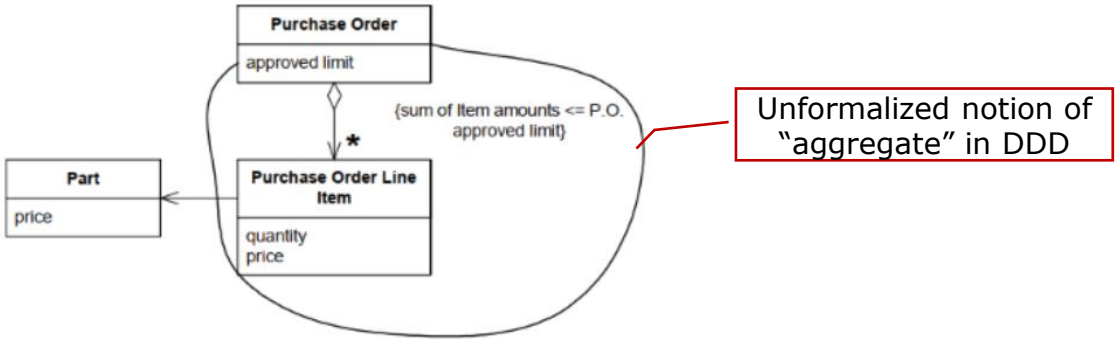
   $\langle a,b \rangle = \{ \{a\}, \{a,b\} \}$

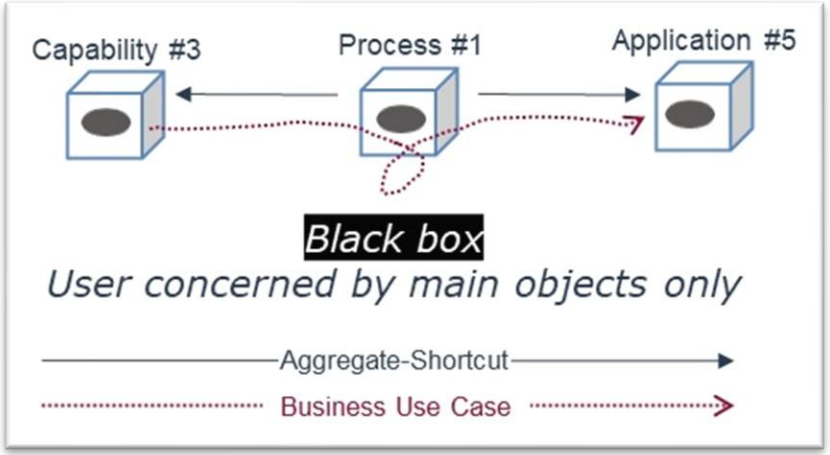
2. Lexical scoping provides the ability to nest entities: namespaces, functions inside, functions, entities inside entities.

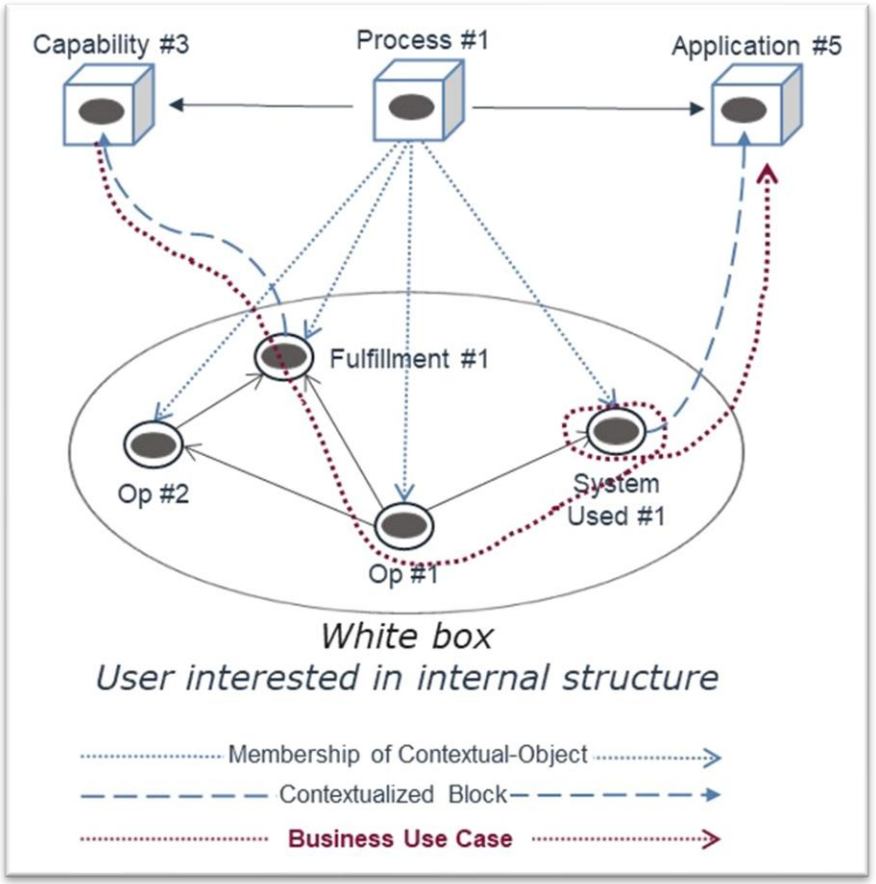
3. Compositionality provides dynamic locality for entities with emerging concepts of building blocks and aggregates (see next slide).

Aggregates

- **Aggregates** are Building Blocks, with an **internal structure** made of aggregate members
- Depending on the working context, users are sometimes interested only in the main entity (Black box), and some other times in the internal structure (White box)
- Typically, Organizations, Applications, Processes and Capabilities are aggregates.
- Remark:
  - Aggregate is a term defined in the **standard Domain Driven Design approach DDD.**
  - SysFEAT offers a formal framework for the intuitive concepts introduced by DDD.







WARNING

Compositionality VERSUS COMPOSITION

- Compositionality is a **syntactic** concept that lacks inherent semantic meaning. It is an ability to assemble entities to form bigger constructs called aggregates, but it does not define the nature of their relationships.
- Compositionality can be applied to various semantic relationships, such as composition, specialization (e.g., "generalization" in UML) or classification.
- For more details on composition, please look at the presentation on semantic.

- A frequent misconception is conflating nesting, compositionality, and composition. While these concepts may combine, they address distinct aspects of system design and should not be confused.
  - Nesting refers to lexical scoping: a syntactic structuring mechanism where entities are hierarchically contained within others.
  - compositionality deals with composite structures: how entities can be combined to form higher-level aggregates while preserving their functional integrity.
  - Composition is a whole/part semantic, which can be elementary or composite

Overview of compositionality principles

- Compositionality is the ability of constructing complex Building Blocks (Aggregates) by local assembly of related entities, creating a unified composite that exhibits emergent properties—qualities that surpass the simple sum of its individual constituents.
- At its core, compositionality bridges two key perspectives: hierarchies (local nesting of related Building Blocks) and networks (connections between Bounded Aggregates). To achieve effective compositionality, four essential characteristics must be present:
  - Reified Relationships: the transformation of a binary relationship between a source and a target entity into a distinct entity called an Aggregate Member, which represents the relationship itself, embedded within its source.
  - Bounded Aggregates : aggregates that encapsulate their internal structure behind a boundary, ensuring clarity and modularity.
  - Boundaries: connection points that express as an ability to connect, in accordance with connection definitions: Connection Entities.
  - Connection Entities : these new kind of entities defines connectivity relationships between assembled aggregates, serving as the glue that binds them together. Examples include Events and Service Interfaces.

Aggregate Blocks & Contextualization (Aggregation)

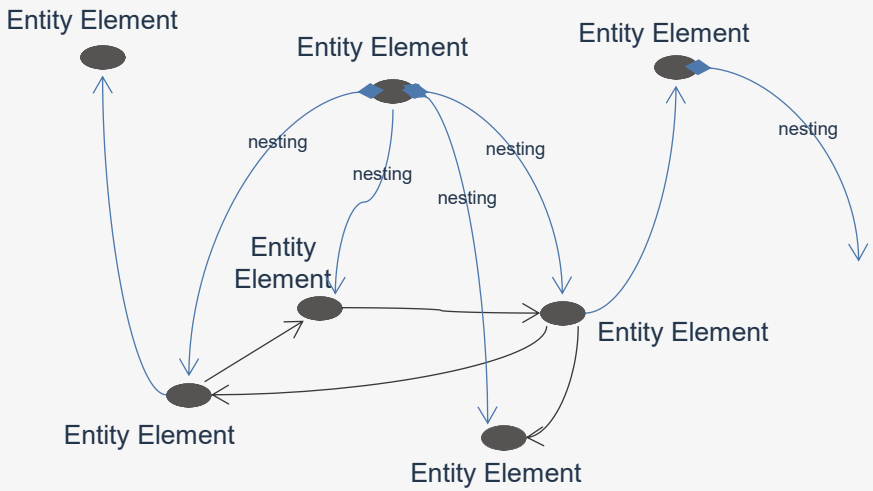
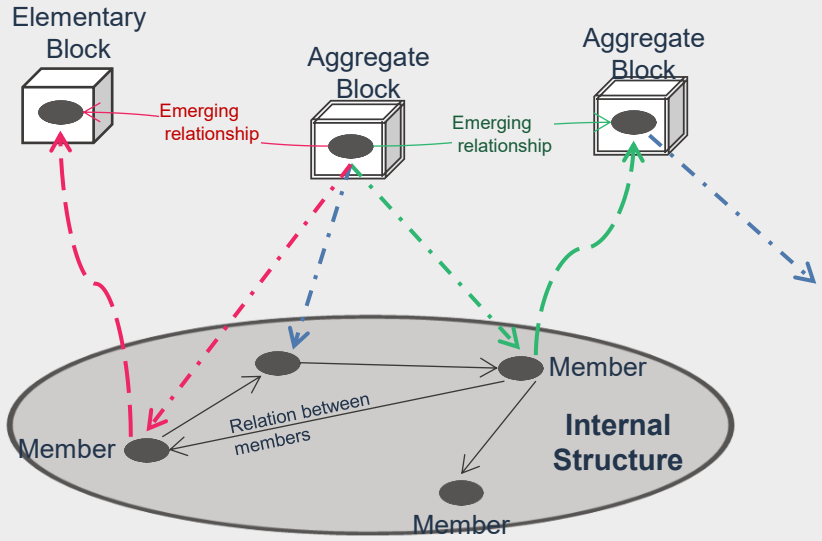
- An Aggregate is a Building Block which has an internal structure made of Aggregate Members :
  - Typically, attributes of a class, steps of a process are aggregate members.
  - Agents, Processes, Capabilities, Data Objects are aggregates.

- Aggregation allows defining characteristics for the aggregated Building Block that only apply within its parent Aggregate. This enables expressing the emergent properties of the composite structure.

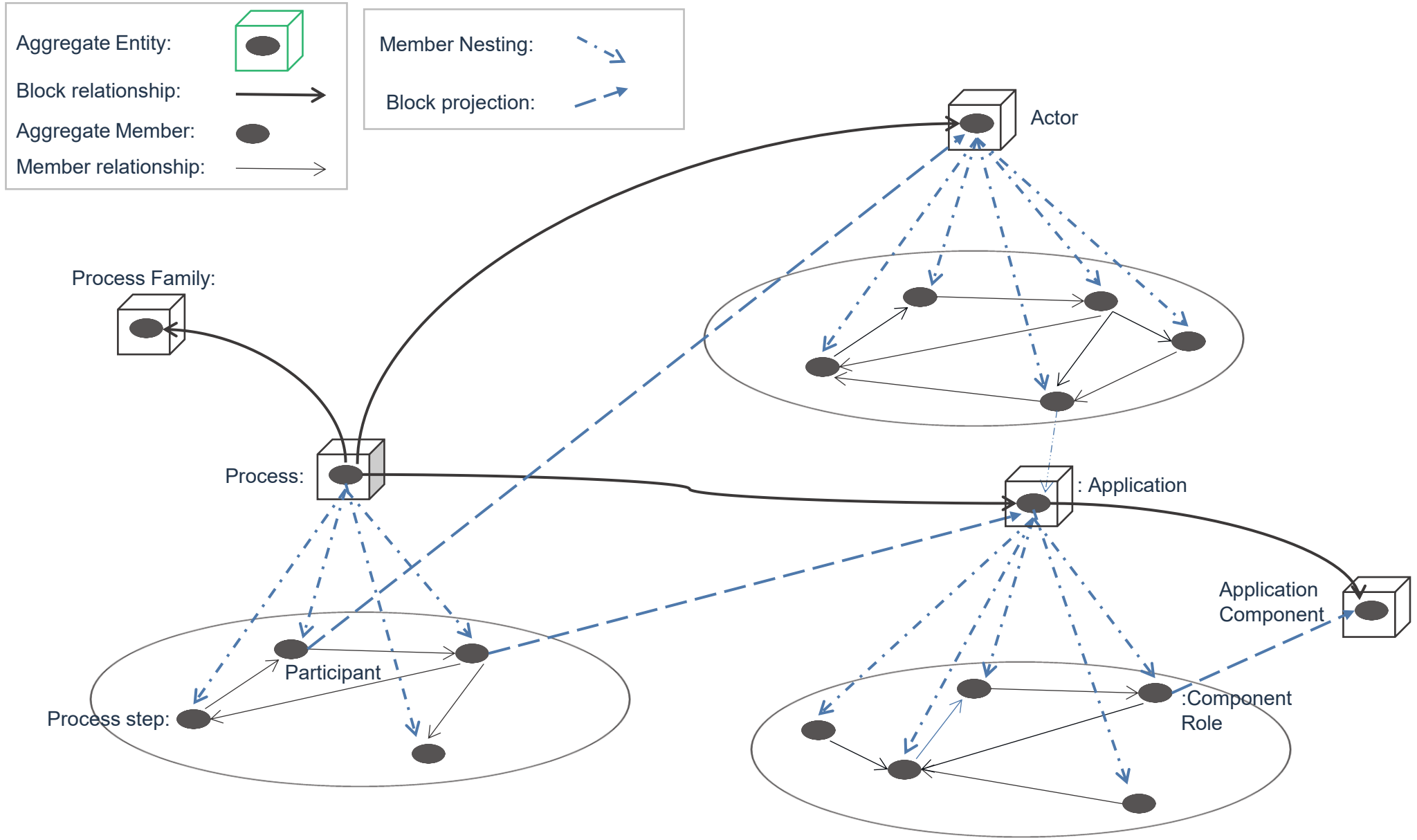
NOTE: The term "Aggregate" originates from the widely recognized Domain-Driven Design (DDD) framework.

Compositionality: towards layered graphs

- Compositionality induces a new method for structuring graphs of entities and relationships, replacing the conventional Entity/Relationship model with a dual-layered graph framework.
  - The Elementary graph is a traditional directed graph that includes nesting relationships.
  - The secondary graph employs nesting relationships as a mechanism to delineate the internal structure of aggregates and infer emerging relationships between them.

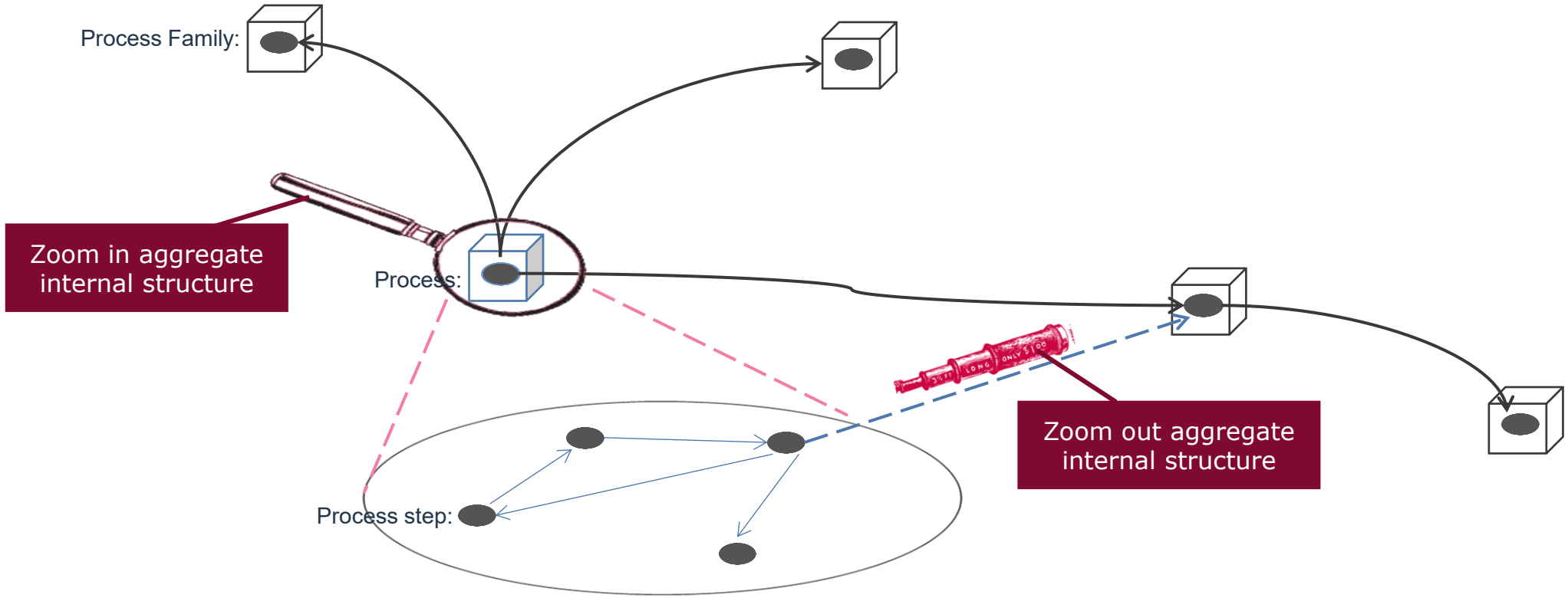
Elementary Graph	
Secondary Graph	

Secondary Graph illustration - BPMN



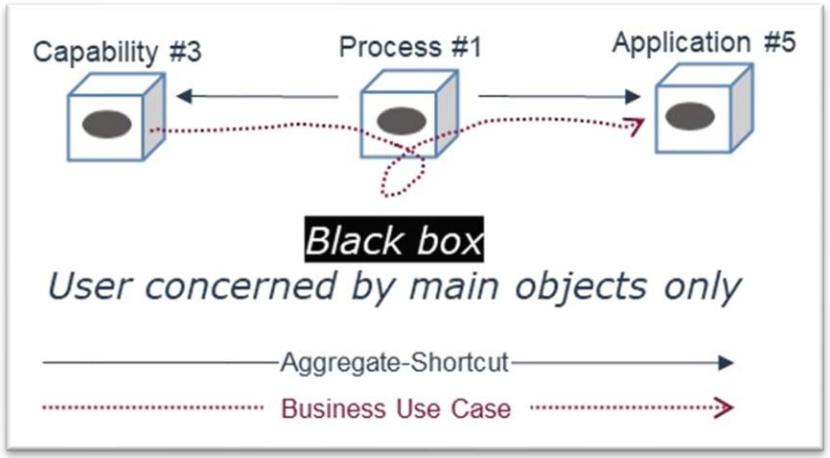
Layered graphs and navigation

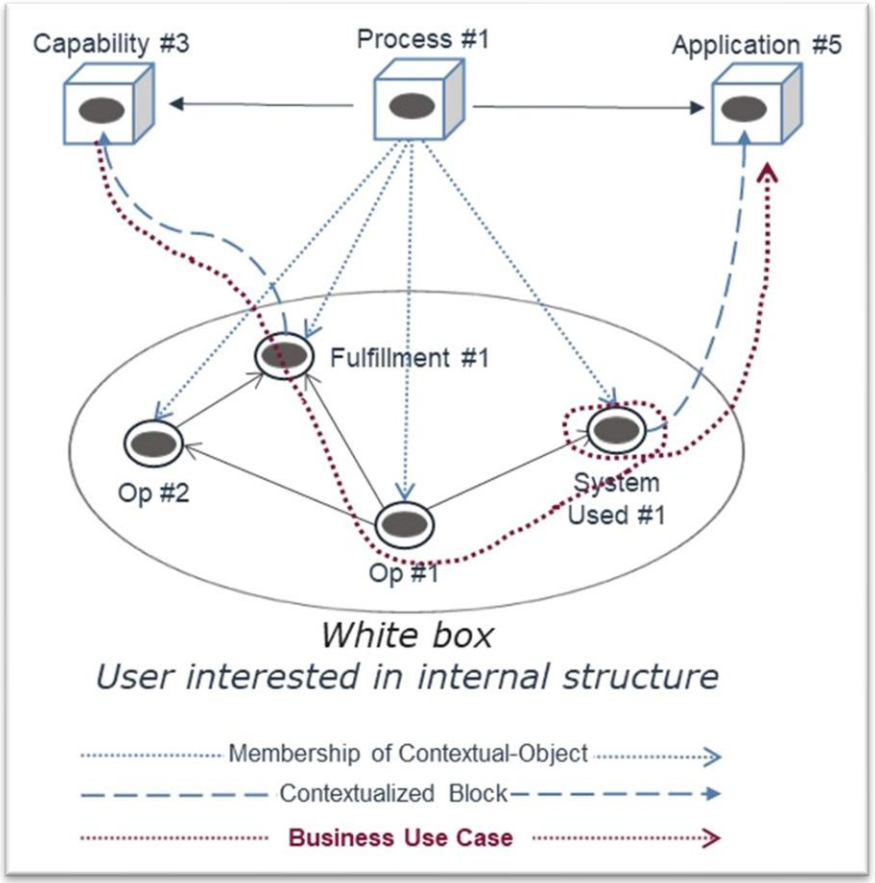
- Depending on the context of their work, users may prioritize either the external aspects of aggregates (Black box) or delve into their internal structure (White box) at different times.“
- Structural zooming is a novel approach of graph navigation to address these needs.
  - It integrates zoom, fusion, or morphing to enable users to scrutinize aggregate details while maintaining visibility of higher-level aggregates and their interconnections.
  - It encompasses two pivotal functions: zoom-in, which delves into the internal structure of aggregates, and zoom-out, which ascends to reveal direct relationships between aggregates.



Aggregate structures versus Views

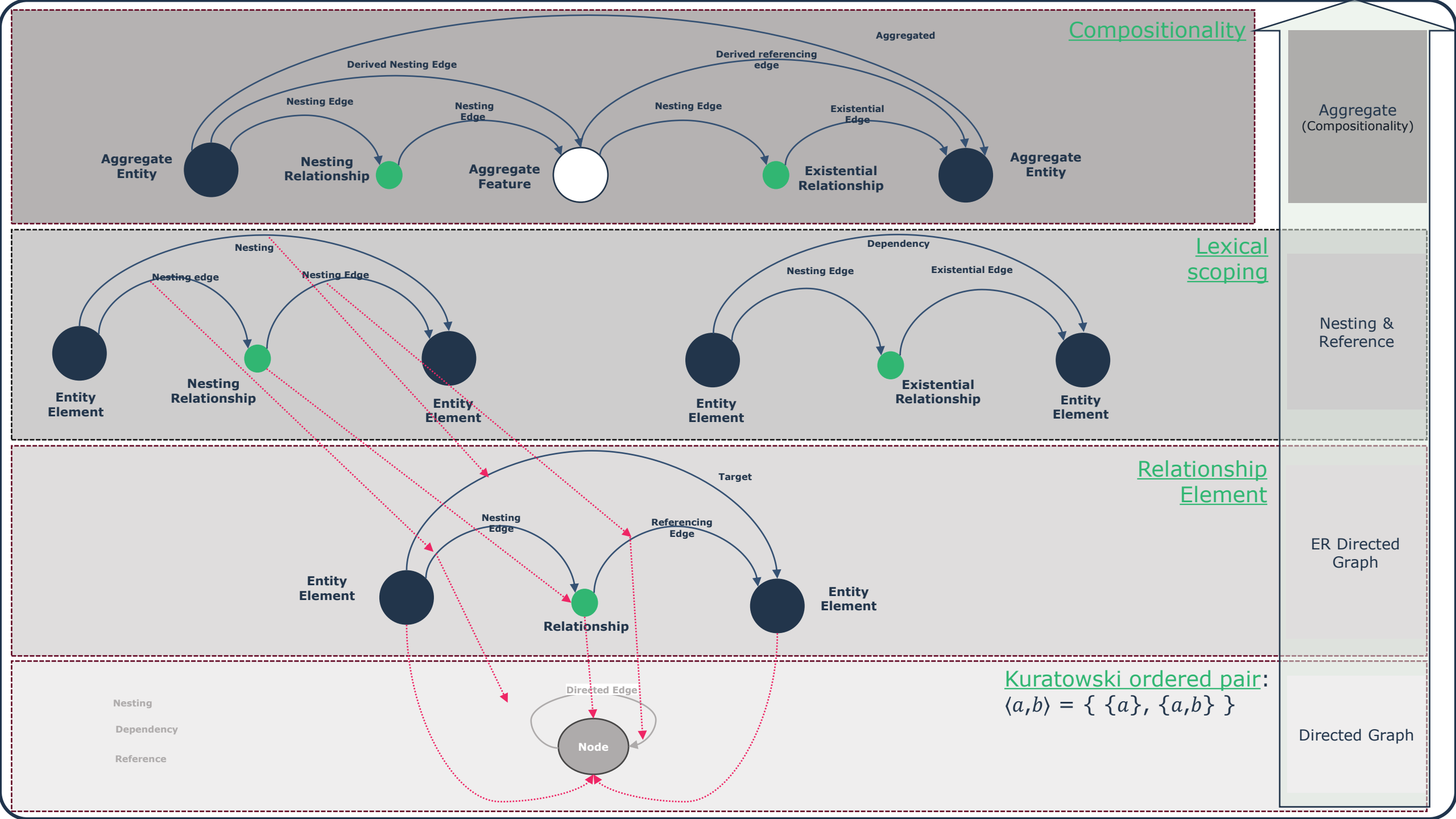
- Emerging relationships should not be confused with relationship paths required to build views.
- Views are traversal of aggregate structures. They are built to respond to specific processing of portions of concepts graphs.





Conclusion on Graphs

- Standard Graph Theory, as defined by the tuple G = (V, E), lacks the formal machinery to represent hierarchical locality or containment. The model is flat and possesses only connectivity, not ability for compositionality.
- Therefore, knowledge graphs can’t directly encode scoped local meanings: everything is global by default.
- The lack of native locality in graphs is a real limitation for using knowledge graphs to direct LLMs, because LLMs (like programming languages) thrive on scope, hierarchy, and compositionality.
- The modularity principles of SysFEAT aim at providing modular connectable structures, using a layered approach of relationships.


Compositionality
Aggregated
Derived Nesting Edge
Derived referencing edge
Nesting Edge
Nesting Edge
Nesting Edge
Existential Edge
Aggregate Entity
Nesting Relationship
Aggregate Feature
Existential Relationship
Aggregate Entity
Aggregate (Compositionality)
Lexical scoping
Nesting
Nesting edge
Nesting Edge
Entity Element
Nesting Relationship
Entity Element
Dependency
Nesting Edge
Existential Edge
Entity Element
Existential Relationship
Entity Element
Nesting & Reference
Relationship Element
Target
Nesting Edge
Referencing Edge
Entity Element
Relationship
Entity Element
ER Directed Graph
Kuratowski ordered pair:
⟨a,b⟩ = { {a}, {a,b} }
Directed Edge
Node
Nesting
Dependency
Reference
Directed Graph

Packaging & Modules

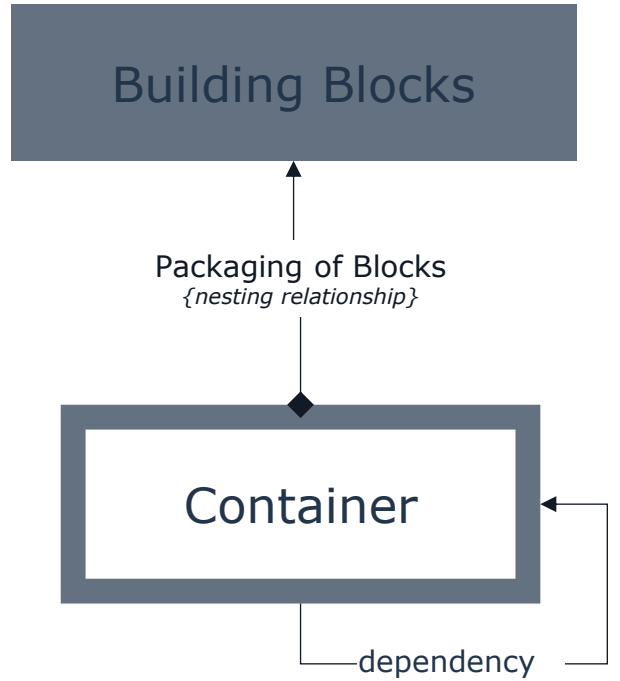
Packaging

- Packaging is the ability to group building blocks into modules commonly referred to as "packages" or "packages" in software engineering. The general concept for packages in the concept of Container.
- Containers are the means to group and version the different components of a system.
  - In the case of software, this components comprise codes, data, configurations
  - In the case of models, this components comprise aggregate building blocks and their connections.
- Containers dependencies must be identified and mastered to automate the delivery of modules and ensure the deployment processes in different environments.
- Containers must be tested, built, and versioned to be ready for continuous deployment.

- Packaging is an essential aspect of incremental delivery and a key foundation for modular enterprise modeling and architecting.

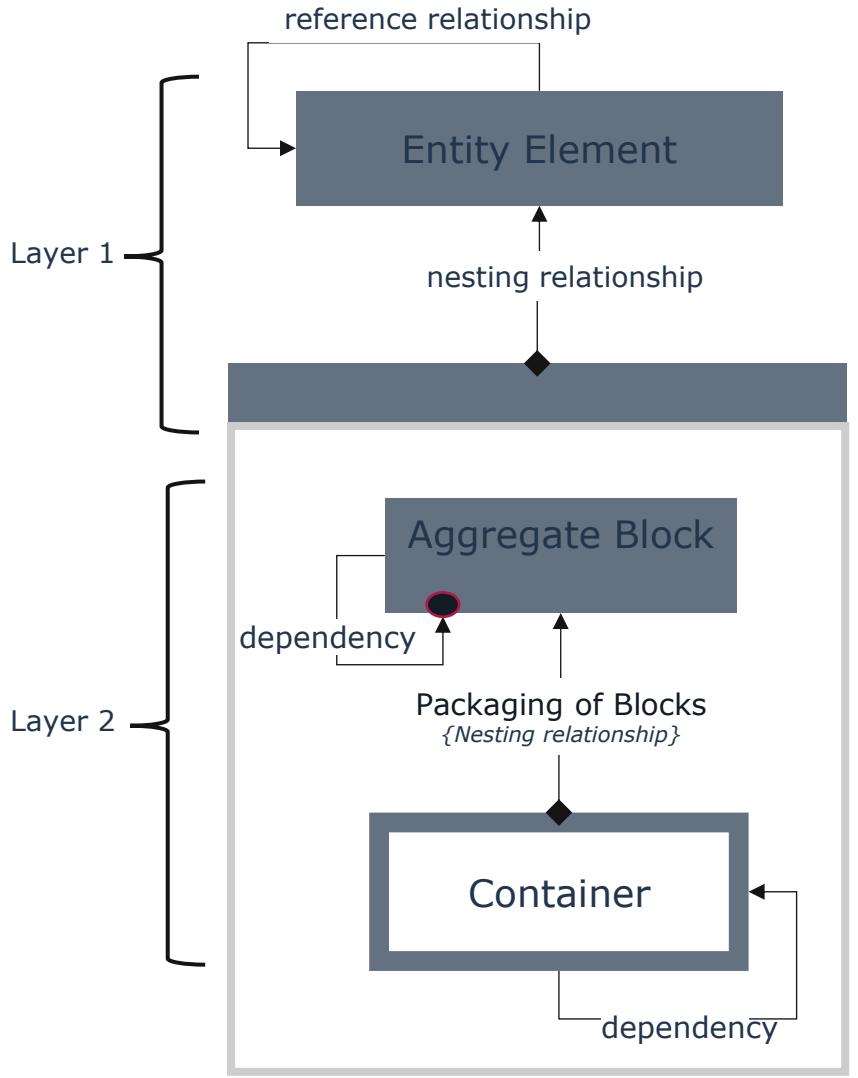
Building Blocks & Containers

- As reusable units, Building Blocks have an independent existence. Thus, they cannot be nested into other Blocks that would hide their existence.
- Because of their independent existence, they must belong to an independent artifact whose sole purpose is the modular management of building blocks: containers.
- Containers are dedicated to the modular management of building blocks:
  - They hold the building blocks to which they provide existence.
  - They can provide a namespace for building blocks.
  - They potentially have dependencies on other containers (see next slide).
- Versioning of modules in EA is a separate concern that will be addressed in a future section on building block management.



Summary of the dependency stack

- Dependencies are built on a stack of two nesting layers:
  1. Ownership of relationships and model elements
     1. Relationship must be directed (ownership of relationships)
     2. Model elements must be owned (nesting relationship) by a cluster of model elements (process owns operation, library owns process).
  2. Clusters & dependencies
     1. Clusters are either:
        1. building blocks: reusable, independent clusters (process owns operation).
        2. Containers: owner of building blocks (library owns process).
     2. dependencies between building blocks creates dependencies between modules.



Container dependencies

- Dependencies between building blocks (red arrows below) result in inferred dependencies between modules that package them.
- Dependencies between modules must be made explicit through a **dependency analysis process**.