SOFISMO

Your code pilot

Modular Modelling Language Design

KISS Workshop, 14 April 2009

ASWEC, Gold Coast, Australia

Modelling languages differ from coding languages

Observation: Given that in the context of software the term coding is often used interchangeably with programming, it is instructive to compare the dictionary definitions of *to code* and *to model* to understand the not-so-subtle difference in intent:

- *to code: express (a statement or communication) in an indirect or euphemistic way*
- *to model: devise a representation, especially a mathematical one of (a phenomenon or system)*

Coding can be viewed as having to deal with someone else's representation (program notation or otherwise). This is exactly what happens when we work with third party implementation technologies and when mapping to such technologies in a template language.

Modelling can be viewed as dealing with a representation that is fit for purpose. This is what happens when we capture knowlegde in a domain specific language that is grounded in established domain terminology.

Need for tools that enforce proven best practices

- Textual grammar specifications
  lack the clarity of graphical models
- Behavioural features
  are irrelevant to language design
- No need to carry baggage
  from a UML/OO mind set
- Modular language and artefact design
  must become a primary concern

Best Practices for Modelling Language Design (1)

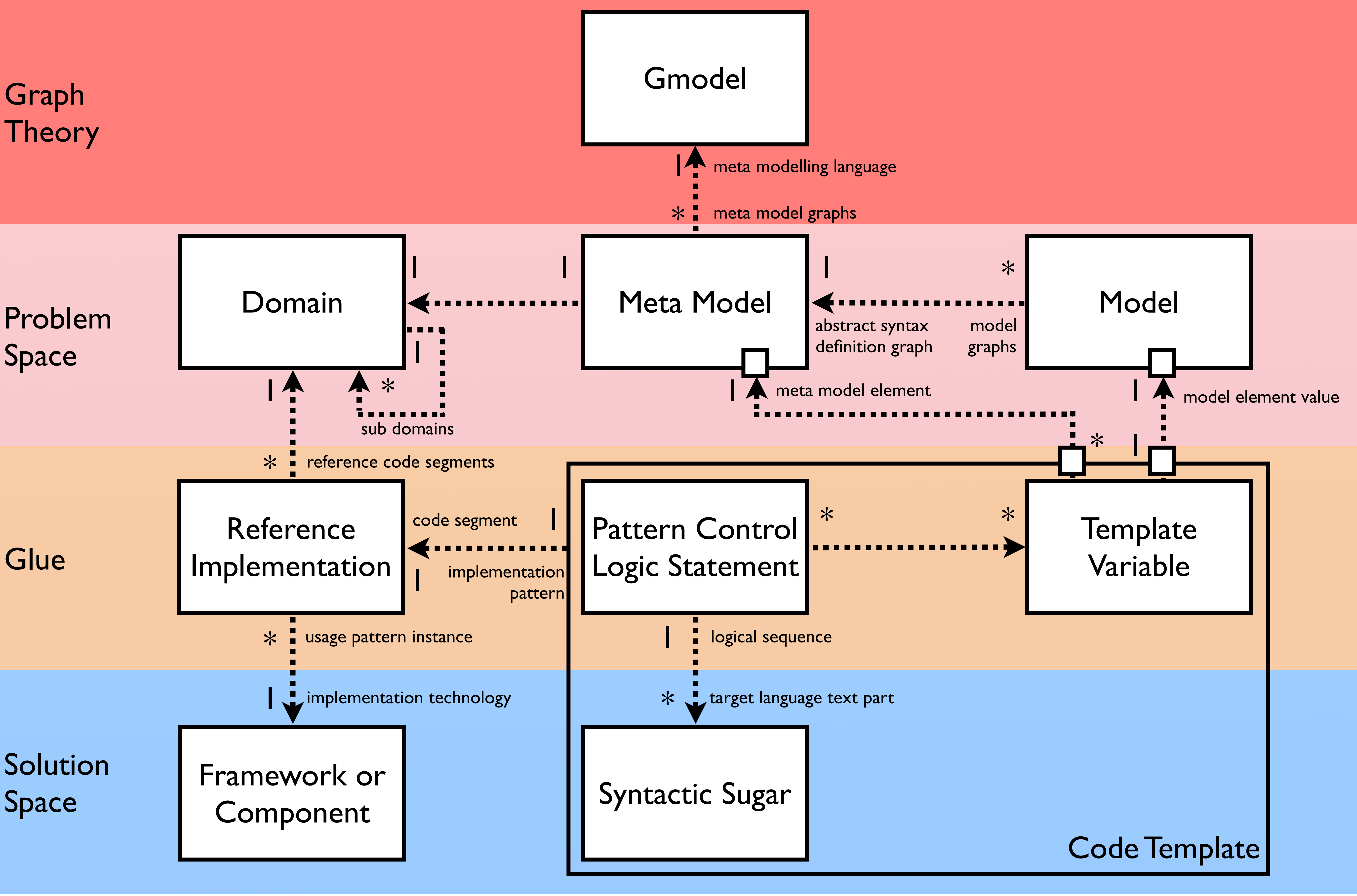
1. Partition a domain into well-defined areas of knowledge, each of which relates to exactly one role
2. All artefacts are based on information produced by a specific role as a result of a specific event
3. A DSL for each kind of artefact (modular meta models)
4. The meta model of a DSL has exactly one root node that relates to the modelled artefact
5. Variants of meta models and model instances are expressed as extensions of a common root
6. Artefacts are the only granularity at which versioning is applied

Best Practices for Modelling Language Design (2)

7. Artefacts are locked for other users while being edited
8. An explicit modularisation mechanism is required within a DSL if model instances are unlimited in size (modular models)
9. Meta models are artefacts as well
10. Multiple meta models are joined via a small set of gateway meta classes
11. No circular or bi-directional dependencies between meta models are allowed
12. The set of models associated with a meta model is recorded as part of the meta model artefact or via a reference to the meta model within each model artefact

Graph theory as the main source of terminology

- A universal representation
  that does not imply any particular implementation for language structures
- Minimizes risk of confusing meta modelling concepts
  with concepts from the language being modelled
  or with concepts from the languages used in the solution space
- Simplifies exploitation of powerful algorithms
  for graph based computations and transformations



Graph Theory
Problem Space
Glue
Solution Space
Gmodel
meta modelling language
meta model graphs
Domain
Meta Model
Model
abstract syntax definition graph
model graphs
meta model element
model element value
sub domains
reference code segments
Reference Implementation
code segment
implementation pattern
Pattern Control Logic Statement
Template Variable
usage pattern instance
logical sequence
implementation technology
target language text part
Framework or Component
Syntactic Sugar
Code Template

Interoperability

- Hinges on the ability to exchange model-based artefacts
- Shared abstract syntax for articulating language definitions
- Meta language must be able to serve as pivot in model-based tool chains
- Software vendors want to - and need to - compete in terms of implementations
- To be relevant, any shared meta language must be extensible

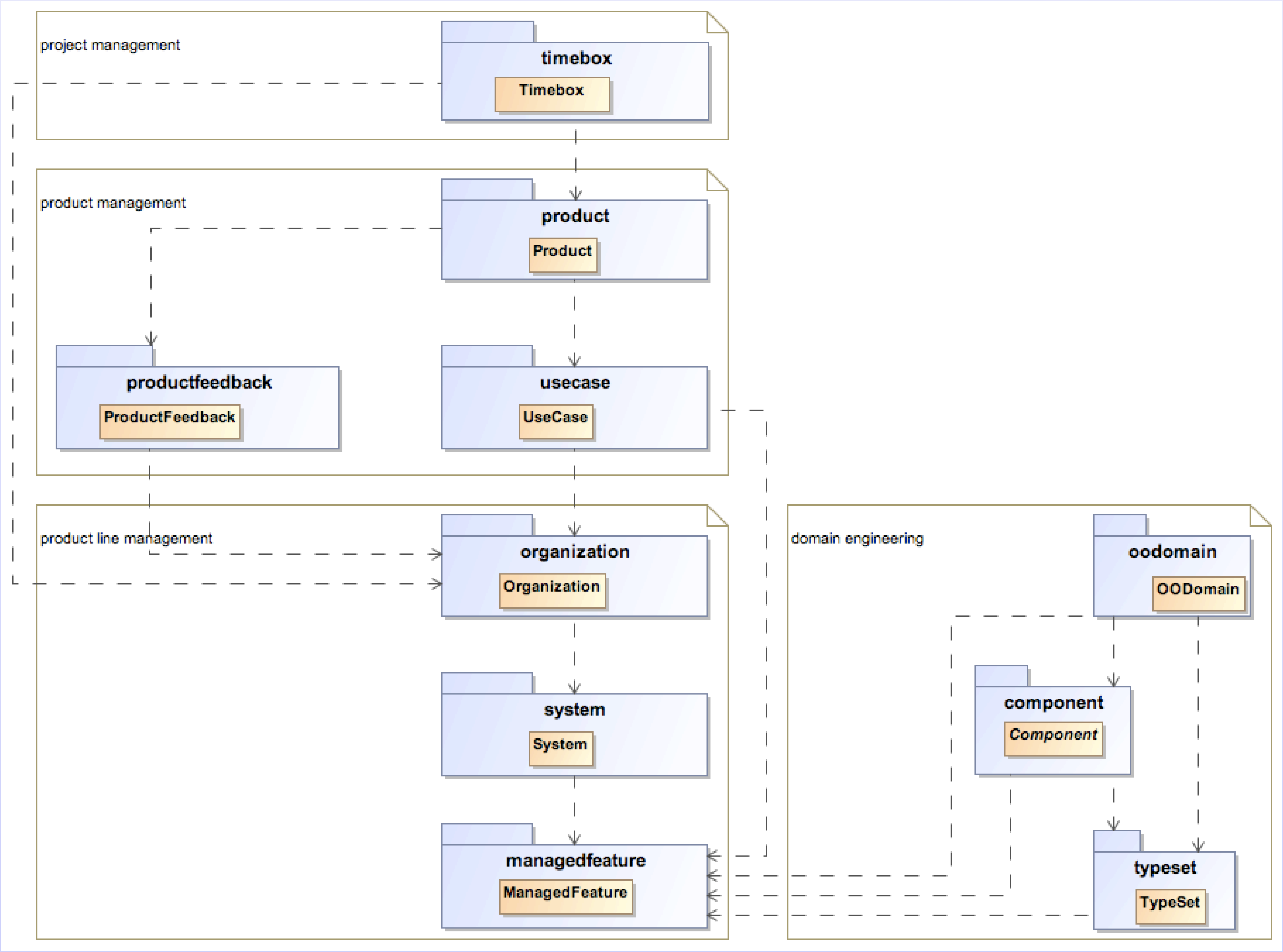
Simplicity

- Specification of an abstract syntax boils down to the specification of language elements and the allowable links between them. Some of which may be navigable in both directions, some only in one.
  - ★ **Abstract syntax specification** is isomorphic to a **graph**
  - ★ **Language elements** correspond to **vertices**
  - ★ **Links** between language elements correspond to **edges**
  - ★ The **role of a vertex in a link** corresponds to an **edge end**
- These elementary graph theoretical concepts form the basis of Gmodel
- They are complemented with a minimal set of additional concepts that are useful for meta modelling and dependency management (**generalization/specialization**, **properties**, and **module references**)

Modularity

- **Fractal decomposition** of graphs into subgraphs
- **Modularity as a 1st class** concept in all language designs
- **Relationships between models as 1st class** language design concepts
- All modelling takes place within an **architectural context**
- Architecture is articulated in an **architecture owner graph**
- **Disallow circuits** across module boundaries

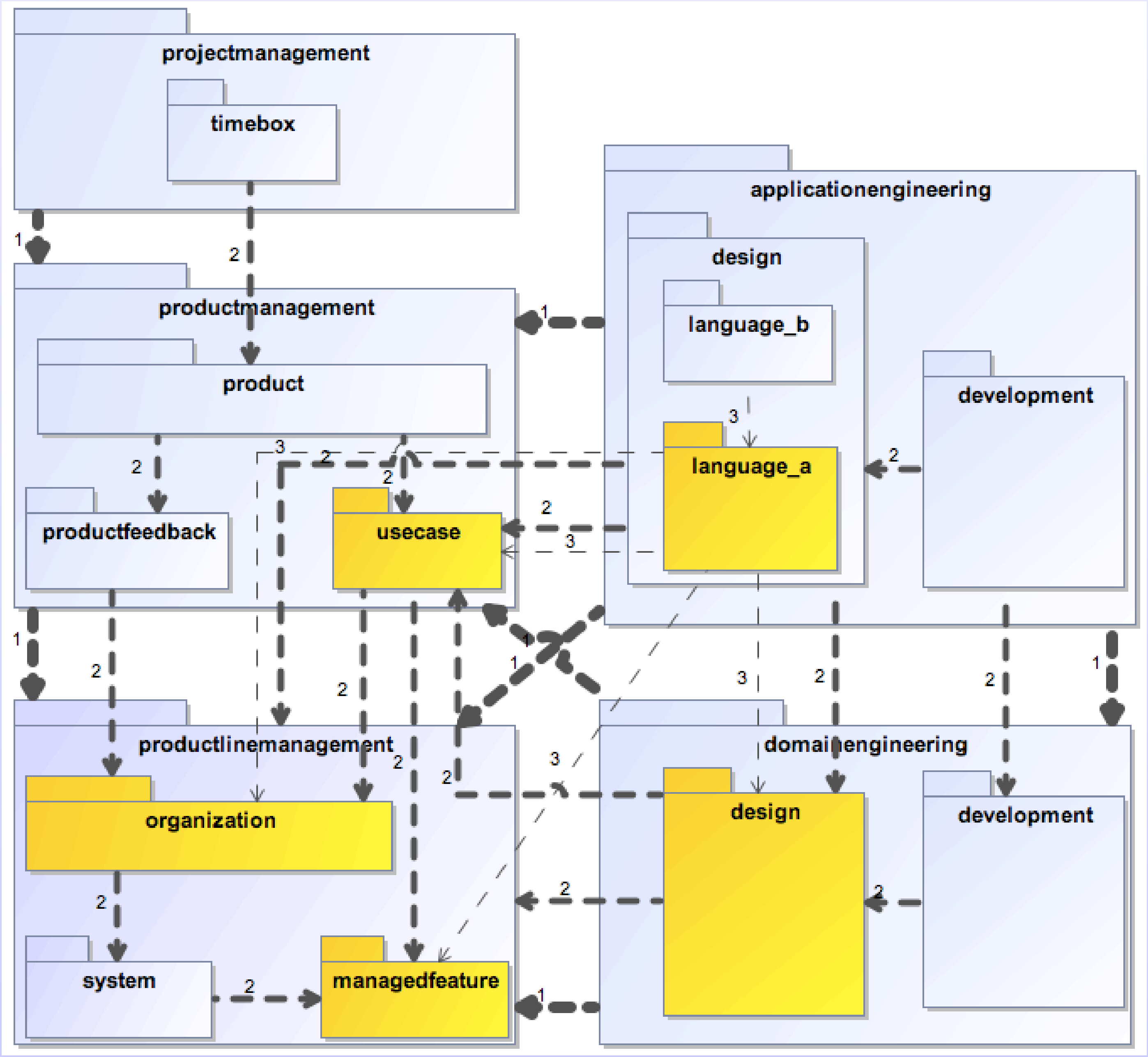
Modularity (Example of subgraphs)



SOFISMO

Your code pilot

Modularity (Example of dependency management)

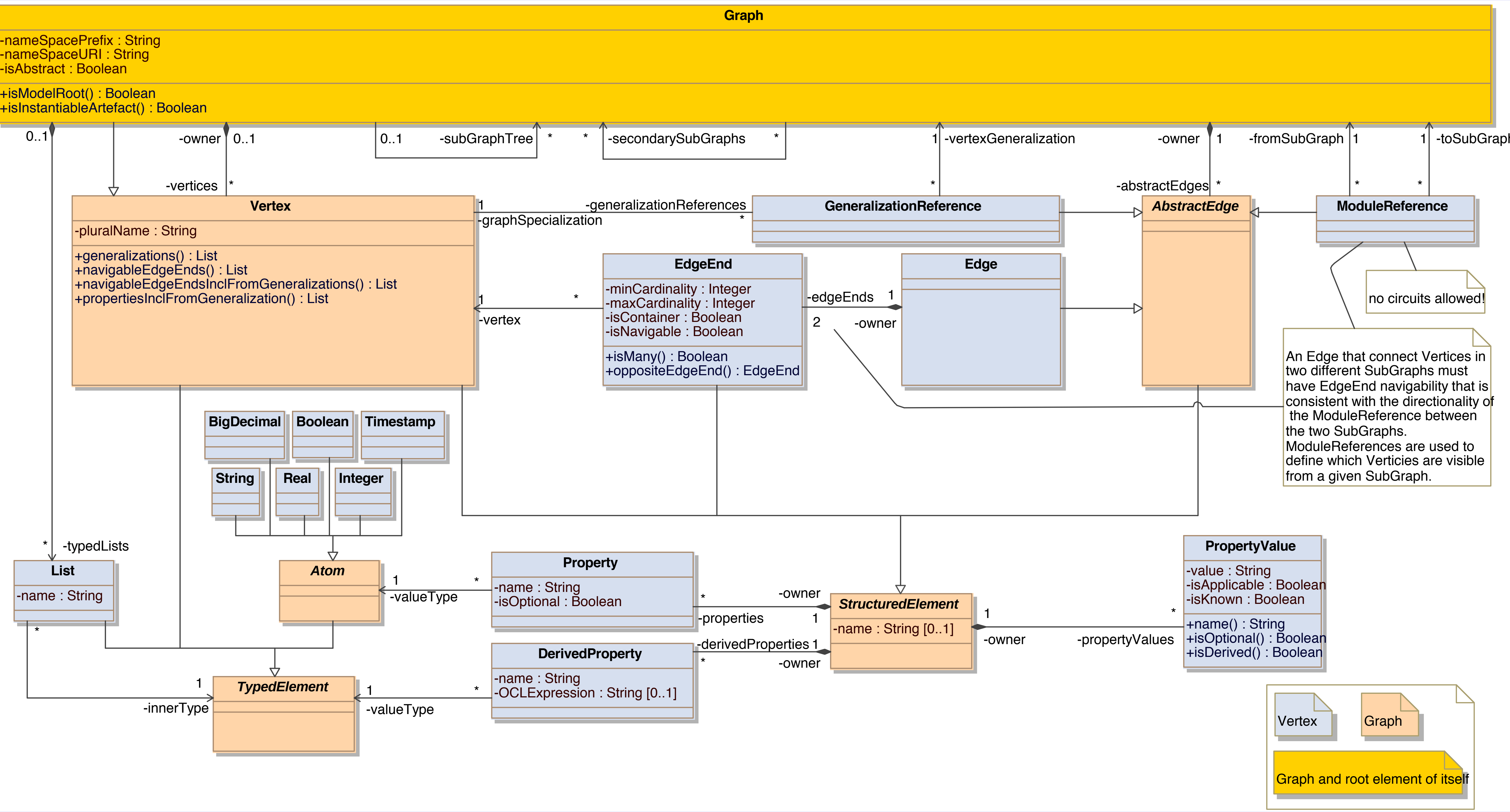


Agile Collaboration

- User interacts with models via a repository that abstracts away the physical storage system as far as possible
- A repository may be geographically distributed
- A user modifying a graph artefact obtains an exclusive lock
- Encourage frequent integration across projects
- Don't provide support for easy management of monolithic models

Extensibility

- For creating a pivot meta language for interoperability between multiple meta languages
- For providing specialized functionality for graph manipulations in specific domains
- For integration with various version management systems
- For meeting custom auditability requirements
- For adding multi-lingual support
- For accommodating **your** specific needs ...
- **For keeping the minimal core small & simple,** otherwise interoperability will remain a pipe dream


Graph
-nameSpacePrefix : String
-nameSpaceURI : String
-isAbstract : Boolean
+isModelRoot() : Boolean
+isInstantiableArtefact() : Boolean
0..1
-owner
0..1
0..1
-subGraphTree
*
*
-secondarySubGraphs
*
1
-vertexGeneralization
-owner
1
-fromSubGraph
1
1
-toSubGraph
-vertices
*
*
-abstractEdges
*
*
*
Vertex
-pluralName : String
+generalizations() : List
+navigableEdgeEnds() : List
+navigableEdgeEndsInclFromGeneralizations() : List
+propertiesInclFromGeneralization() : List
1
-generalizationReferences
-graphSpecialization
*
GeneralizationReference
AbstractEdge
ModuleReference
EdgeEnd
-minCardinality : Integer
-maxCardinality : Integer
-isContainer : Boolean
-isNavigable : Boolean
+isMany() : Boolean
+oppositeEdgeEnd() : EdgeEnd
1
*
-vertex
-edgeEnds
1
2
-owner
Edge
no circuits allowed!
An Edge that connect Vertices in two different SubGraphs must have EdgeEnd navigability that is consistent with the directionality of the ModuleReference between the two SubGraphs. ModuleReferences are used to define which Verticies are visible from a given SubGraph.
BigDecimal
Boolean
Timestamp
String
Real
Integer
*
-typedLists
List
-name : String
*
Atom
1
-valueType
*
Property
-name : String
-isOptional : Boolean
*
-owner
-properties
1
StructuredElement
-name : String [0..1]
1
-owner
-propertyValues
*
PropertyValue
-value : String
-isApplicable : Boolean
-isKnown : Boolean
+name() : String
+isOptional() : Boolean
+isDerived() : Boolean
-derivedProperties
1
*
-owner
DerivedProperty
-name : String
-OCLExpression : String [0..1]
1
TypedElement
-innerType
1
-valueType
*
Vertex
Graph
Graph and root element of itself

Modelling of Graphs

- **Vertices, Edges, Edge Ends** - as to be expected
- The content of a graph can be organized into a
Sub Graph Tree (non-overlapping sets of sub graphs)

★ containment semantics for physical modularization of graphs

- The sub graph tree may also contain
Secondary Sub Graphs

★ the mechanism needed to represent overlapping sets of sub graphs

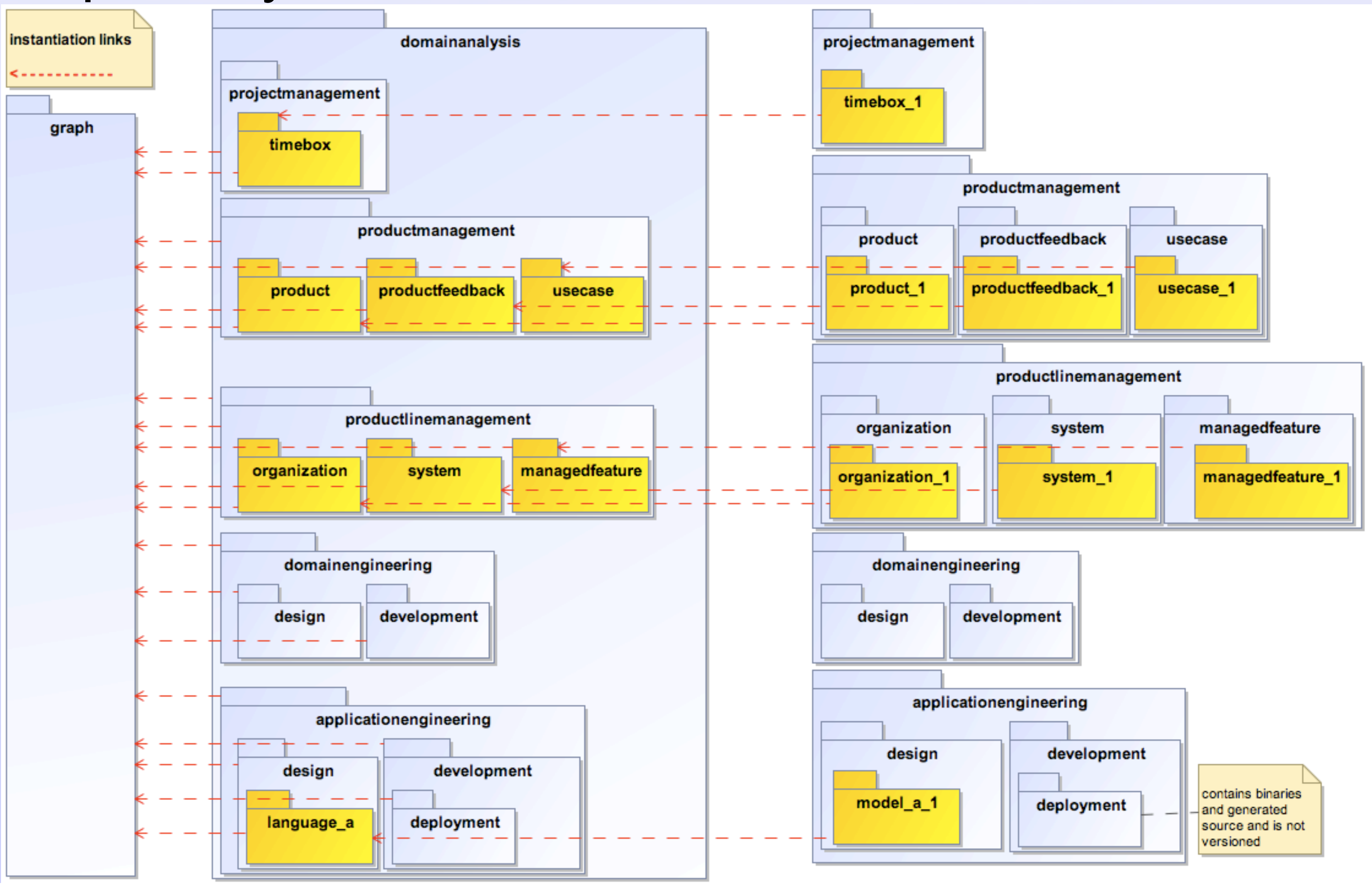
Modelling of Modelling Languages (1)

- Generalization: A graph may have one or more generalizations. All generalizations of a graph are graphs as well.
- Is abstract: A graph has a property to indicate whether it can be instantiated or whether it only serves as an abstract generalization that has instantiable specializations.
- Element: All vertices (or graphs) in the encoding of Gmodel are either direct or indirect specializations of Element.
- Metatype: Each element is an instance of a metatype, which provides the basis for distinguishing meta levels.
- **No artificial limit to the number of meta levels**. Any concrete (non-abstract) graph can be used in the role of a meta model

Modelling of Modelling Languages (2)

- Structured element: Distinction between structured elements that may have properties (vertices, abstract edges, and edge ends) and other elements that may not have properties.
- A graph is a specialization of a vertex: An element that appears as a vertex in a graph, may at the same time be expanded into a sub graph that contains further vertices and edges.
- **All non-empty graphs are independent physical artefacts**

Repository structure = domain structure



SOFISMO

Your code pilot

Thank you!

Jorn Bettin
jbe @ sofismo.ch
Skype jorn_bettin
+41 62 891 0987