MULTI 2015

2nd Int. Workshop on Multi-Level Modelling

MODELS 2015 Ottawa

An algebraic instantiation technique illustrated by multilevel design patterns

Zoltan Theisz and Gergely Mezei

www.huawei.com

BME AUT

HUAWEI

Outline

- **Practical motivation**
  - Multi-level meta-modelling propriety solutions in modern telecom management
- **Theoretical motivation**
  - Juan de Lara's SoSyM paper on multi-level meta-modelling patterns
- **Dynamic Multi-Layer Algebra**
  - Theoretical introduction (structure, functions, bootstrap, dynamic instantiation)
  - Examples (syntax with compact notation)
- **Multi-level meta-modelling patterns**
  - Type-Object pattern
  - Dynamic Features

Practical Motivation

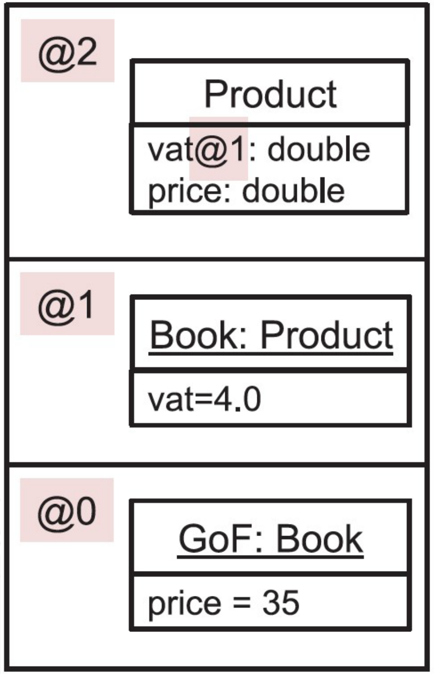
- **Modern telecom network management is getting more centralised**
  - Network devices are managed via Software Defined Networking (SDN)
  - Network services are virtualised and managed as Virtual Network Functions (VNF)
  - Global service orchestrators keep all data in model-based repositories
  - Model manipulation indirectly influences the operation of complex multi-operator, multi-vendor, multi-technology services and devices in the physical network and data centres
  - Complex telecom services are gradually created by stake-holders in an ecosystem
  - Model-based orchestration solutions must support both design- and run-time modelling
  - Flexible management of modelled elements is needed
  - Model repositories not only store instances, but also keep references to their types

- **Modelling in telecom management must be DevOps-enabled**

Theoretical Motivation

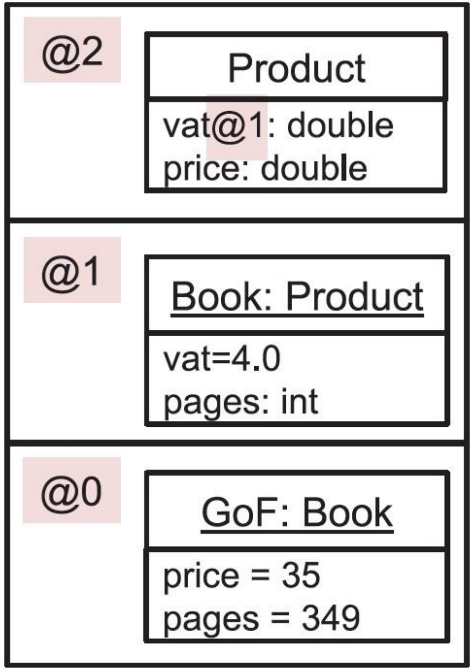
- Instantiation is key operation between model and meta-model levels
- Linguistic instantiation is well-established by current methodologies
- Ontological meta-modelling is carried out by architects at design-time
- Combined linguistic and ontological model design may become complex
- Instantiation can also connect design-time and run-time models
- Complex software ecosystems may require partial instantiation by steps
- Multi-level instantiation support by potency notion is generic
- The levelling of partial instantiation is to be pre-defined in potency

Theoretical Motivation (Type-Object pattern)



- **Intent:**
  - Explicit modelling of types and their instances
  - Types can be added dynamically
  - Types define features that are known a priori
  - Instances concretise type features
- **Usage:**
  - Model-based telecom management systems for
    - flexible introduction of new types on demand
    - establishment of 3-level modelling for devices (device kind -> device type -> device instance)

Theoretical Motivation (Dynamic Features)



- **Intent:**
  - Adding new features to existing types
  - Instances can concretise new features
- **Usage:**
  - Model-based telecom management systems for
    - flexible extension of types on demand
    - supporting type ecosystems among various tools (multiple stake-holders handle devices differently and independently of each other)

Dynamic Multi-Layer Algebra

- **Concepts**
  - Formal definition (ASM based)
  - Dynamic (partial) instantiation
  - Extendable initialisation

- **Components**
  - Modelling structure and functions
  - Built-in constructs (Bootstrap)
  - Dynamic instantiation mechanism

Data Representation (Labels)

Labelled Directed Graph (Nodes, Edges, Labels)

- Both Nodes and Edges can have following Labels:
  - **ID**: globally unique ID of model entity
  - **Name**: name of model entity
  - **Cardinality**: cardinality of model entity
  - **Meta**: ID of meta-model entity
  - **Value**: value of model entity (used only for attributes)
  - **Attributes** (children): list of attributes
    - Attributes are virtual nodes with the root as a model element (complex tree structure)

Data Representation (Universes)

Superuniverse $|\mathfrak{A}|$ of a state $\mathfrak{A}$ of Dynamic Multi-Layer Algebra

- Universes defined:
  - › $\mathbf{U_{Bool}}$: contains logical values {true/false}
  - › $\mathbf{U_{Number}}$: contains rational numbers $\{\mathbb{Q}\}$ and infinity ∞
  - › $\mathbf{U_{String}}$: contains character sequences of finite length
  - › $\mathbf{U_{ID}}$: contains all the possible entity IDs
  - › $\mathbf{U_{Basic}}$: contains elements from $\{U_{Bool} \cup U_{Number} \cup U_{String} \cup U_{ID}\}$

- All universes contain **undef** representing an undefined value

Data Representation (Labels + Universes)

Labels of entities take values from universes

- Entity X has following Name-Value mappings
  - X_{Name}: U_{String}
  - X_{ID}: U_{ID}
  - X_{Meta}: U_{ID}
  - $X_{Cardinality}$: $[U_{Number}, U_{Number}]$
  - X_{Value}: U_{Basic}
  - X_{Attrib}: $U_{ID}[]$

Example: $Book_{ID}=42, Book_{Meta}=123, Book_{Cardinality}=[0,\infty], X_{Value}=$ undef, $Book_{Attrib}=[]$

Compact notation: **{"Book", 42, 123, [0, inf], undef, []}**

ASM Functions (Shared Functions)

Shared functions: represent model entity (current) configuration

› Can be modified either by algebra or environment (e.g. $X_{IDConcreteObjec} := X_{NewMetaDefinition}$)

- $Name(ID): \begin{cases} name, if\ \exists X:\ X_{ID} = ID\ \wedge X_{Name} = name \\ undef, otherwise \end{cases}$
- $Meta(ID): \begin{cases} Y_{ID}, if\ \exists X, Y: X_{ID} = ID\ \wedge X_{Meta} = Y_{ID} \\ undef, otherwise \end{cases}$
- $Card(ID): \begin{cases} [low, high], if\ \exists X:\ X_{ID} = ID\ \wedge \\ \quad X_{Cardinality} = [low, high] \\ undef, otherwise \end{cases}$
- $Value(ID): \begin{cases} val, if\ \exists X:\ X_{ID} = ID\ \wedge X_{Value} = val \\ undef, otherwise \end{cases}$
- $Attrib(ID, Idx): \begin{cases} attrib, if\ \exists X, i:\ X_{ID} = ID\ \wedge \\ X_{Attrib}[Idx] = attrib \\ undef, otherwise \end{cases}$

ASM Functions (Derived Functions)

Derive functions: represent calculations

- Cannot change the model
- Only obtain or restructure existing information

- $Contains(ID_1, ID_2): \begin{cases} true, if\ \exists c, idx: c = Attrib(ID_1, idx) \wedge \\ (c_{ID} = ID_2 \vee Contains(c_{ID}, ID_2)) \\ false,\ otherwise \end{cases}$

- $DeriveFrom(ID_1, ID_2): \begin{cases} true,\quad \exists x, y: x_{ID} = ID_1 \ \wedge \ \exists y: y_{ID} = ID_2 \\ \wedge\ (x_{Meta} = y \vee DeriveFrom(x_{Meta}, y)) \\ false,\ otherwise \end{cases}$

Built-in Constructs (Basic Types)

Basic entities (“reification” of DMLA’s universes)

› Entities required to represent basic types for Meta (otherwise X_{Meta}: U_{ID} in ASM by default)

- **Bool** :- U_{Bool}
- **Number** :- U_{Number}
- **String** :- U_{String}
- **ID** :- U_{ID}
- **Basic** :- U_{Basic}

  › Bool, Number, String and ID inherit from Basic (Note: $U_{Basic} = \{U_{Bool} \cup U_{Number} \cup U_{String} \cup U_{ID}\}$)

  › Other basic types such as Date, Double etc. could be introduced similarly

Principal Entities

- **Attribute**:
  - {"Attribute", $ID_{Attribute}$, $ID_{Attribute}$, [0, inf], undef, [{"Attributes", $ID_{Attributes}$, $ID_{Attribute}$, [0, inf], undef,[]}]}
- **AttribType**:
  - {"AttribType", $ID_{AttribType}$, $ID_{Attribute}$, [0, 1], undef, [{"AType", ID_{AType}, $ID_{AttribType}$, [0, 1], ID_{ID},[]}]}
- **Node**:
  - {"Node", ID_{Node}, ID_{Node}, [0, inf], undef, [{"Attributes", $ID_{Attributes}$, $ID_{Attribute}$, [0, inf], undef,[]}]}
- **Edge**:
  - {"Edge", ID_{Edge}, ID_{Edge}, [0, inf], undef, [{"Attributes", $ID_{Attributes}$, $ID_{Attribute}$, [0, inf], undef,[]}, {"EdgeSrc", $ID_{EdgeSrc}$, ID_{Src}, [1, 1], ID_{Node},[]}, {"EdgeTrg", $ID_{EdgeTrg}$, ID_{Trg}, [1, 1], ID_{Node},[]}]}
  - {"Src", ID_{Src}, $ID_{Attribute}$, [1, 1], undef, [{"SrcType", $ID_{SrcType}$, $ID_{AttribType}$, [0, 1], ID_{Node},[]}]}
  - {"Trg", ID_{Trg}, $ID_{Attribute}$, [1, 1], undef, [{"TrgType", $ID_{TrgType}$, $ID_{AttribType}$, [0, 1], ID_{Node},[]}]}

Principal Entities (Attribute-like)

- **Attribute**:
  - {“Attribute”, **ID**$_{\textbf{Attribute}}$, **ID**$_{\textbf{Attribute}}$, [0, inf], undef,
    [
    {“Attributes”, ID$_{\text{Attributes}}$, **ID**$_{\textbf{Attribute}}$, [0, inf], undef,[]}
    ]
    }
- **AttribType**:
  - {“AttribType”, ***ID***$_{\textit{AttribType}}$, **ID**$_{\textbf{Attribute}}$, [0, 1], undef,
    [
    {“AType”, ID$_{\text{AType}}$, ***ID***$_{\textit{AttribType}}$, [0, 1], ID$_{\text{ID}}$,[]}
    ]
    }

Note: ID$_{\text{ID}}$ refers to Basic Types

Principal Entities (Type-like)

- **Node**:
  - {"Node", $\mathbf{ID_{Node}}$, $\mathbf{ID_{Node}}$, [0, inf], undef, [{"Attributes", $ID_{Attributes}$, $ID_{Attribute}$, [0, inf], undef,[]}]}
- **Edge**:
  - {"Edge", ID_{Edge}, ID_{Edge}, [0, inf], undef,
[
{"Attributes", $ID_{Attributes}$, $ID_{Attribute}$, [0, inf], undef,[]},
{"EdgeSrc", $ID_{EdgeSrc}$, $\underline{\mathbf{ID_{Src}}}$, [1, 1], ID_{Node},[]},
{"EdgeTrg", $ID_{EdgeTrg}$, $\underline{\mathbf{ID_{Trg}}}$, [1, 1], ID_{Node},[]}
]}
  - {"Src", $\underline{\mathbf{ID_{Src}}}$, $ID_{Attribute}$, [1, 1], undef, [{"SrcType", $ID_{SrcType}$, $ID_{AttribType}$, [0, 1], $\mathbf{ID_{Node}}$,[]}]}
  - {"Trg", $\underline{\mathbf{ID_{Trg}}}$, $ID_{Attribute}$, [1, 1], undef, [{"TrgType", $ID_{TrgType}$, $ID_{AttribType}$, [0, 1], $\mathbf{ID_{Node}}$,[]}]}

Entity Examples

- **Simple Attribute:**
  - attr String Age
  - {“Age”, $ID_{AgeAttribute}$, $ID_{Attribute}$, [1, 1], undef, [{“AgeType”, $ID_{AgeType}$, $ID_{AttribType}$, [0, 1], ID_{Number} ,[]}]}

- **Complex Attribute:**
  - complex Name { attr String FirstName, attr String LastName}
  - {“Name”, ID_{Name}, $ID_{Attribute}$, [1, 1], undef, [
    {“FirstName”, $ID_{FirstName}$, $ID_{Attribute}$, [1, 1], undef ,[{“FNType”, ID_{FNType}, $ID_{AttribType}$, [0, 1], ID_{String} ,[]}]}
    {“LastName”, $ID_{LastName}$, $ID_{Attribute}$, [1, 1], undef ,[{“LNType”, ID_{LNType}, $ID_{AttribType}$, [0, 1], ID_{String} ,[]}]}]}

Dynamic Instantiation

- **Structure definition and bootstrap represent models as states of DMLA**
- **Instantiation can create many different instances of the same type without violating meta definition constraints**
- **Model manipulation may result in valid or invalid models**
- **Instantiation is checked by formulae (Helper & Validation Formulae)**
  - Helper formula example: $\varphi_{CardinalityCheck}(C, I)$: ¬DeriveFrom(I, $ID_{Attribute}$) ∨ Card(Meta(I))[0] ≤ Count(a: ∃i: a=Attrib(C, i) ∧ $\varphi_{InstCounter}$(I, a)) ≤Card(Meta(I))[1]
  - Validation formula example: $\varphi_{EntityIns}(I, M)$: {∃c, idx: Attrib(I, idx) = c ∧ $\varphi_{IsValid}$(c, Meta(c))} ∨ Value(I) ≠ undef
- **Instantiation procedure verifies formulae after each partial instantiation**

Instantiation Procedure

- Iterative process
- Instantiates at least one entity (e.g. attribute) in each step
- Based on annotated attributes and principal entities
- Consists of instructions having abstract selector and action functions
- Verifies 7 instantiation validation formulae

Algorithm The instantiation algorithm

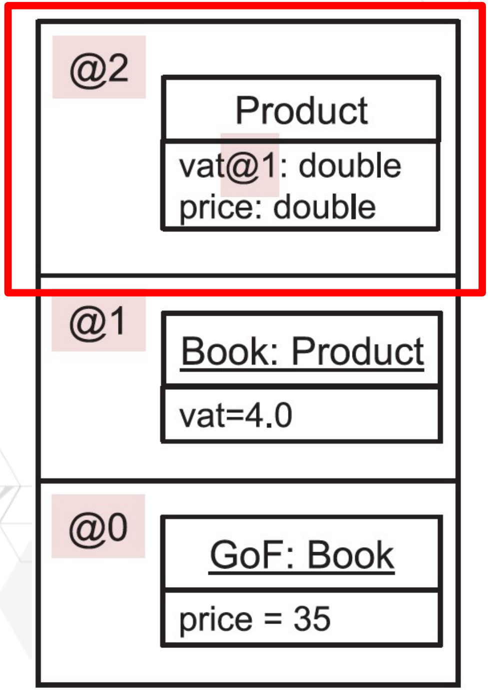
```
rule Instantiate(ID_SubjectEntity, Instructions)
for all λ_selector, λ_action in Instructions do
    for all SelectedEntity in λ_selector(ID_SubjectEntity) do
        λ_action(SelectedEntity)
```

Instantiation Examples

- **Simple attribute:**
  - {"Age", $ID_{AgeAttribute}$, $ID_{Attribute}$, [1, 1], undef, [{"AgeType", $ID_{AgeType}$, $ID_{AttribType}$, [0, 1], ID_{Number} ,[]}]}
    - {"Age", $ID_{ConcreteAgeAttribute}$, $ID_{AgeAttribute}$, [1, 1], 23, []}

- **Complex attribute:**
  - {"Name", ID_{Name}, $ID_{Attribute}$, [1, 1], undef, [
    {"FirstName", $ID_{FirstName}$, $ID_{Attribute}$, [1, 1], undef ,[{"FNType", ID_{FNType}, $ID_{AttribType}$, [0, 1], ID_{String} ,[]}]}
    {"LastName", $ID_{LastName}$, $ID_{Attribute}$, [1, 1], undef ,[{"LNType", ID_{LNType}, $ID_{AttribType}$, [0, 1], ID_{String} ,[]}]}]}
    - {"ConcreteName", $ID_{ConcreteName}$, ID_{Name}, [1, 1], undef, [
      {"FirstName", $ID_{ConcreteFirstName}$, $ID_{FirstName}$, [1, 1], "John" ,[]}
      {"LastName", $ID_{ConcreteLastName}$, $ID_{LastName}$, [1, 1], "Smith" ,[]}]}

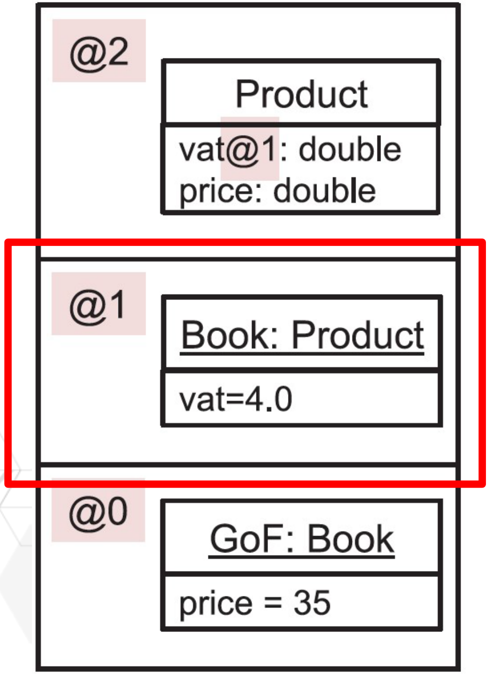
Type-Object pattern (Level 2)



{“Product”, $ID_{Product}$, ID_{Node}, [0, inf], undef,
[
{“vat”, ID_{Vat}, $ID_{Attribute}$, [1, 1], undef,[
{“vatType”, $ID_{VatType}$, $ID_{AttribType}$, [0, 1], ID_{Number}, []},

{“price”, ID_{Price}, $ID_{Attribute}$, [1, 1], undef,[
{“priceType”, $ID_{PriceType}$, $ID_{AttribType}$, [0, 1], ID_{Number}, []}
]
}

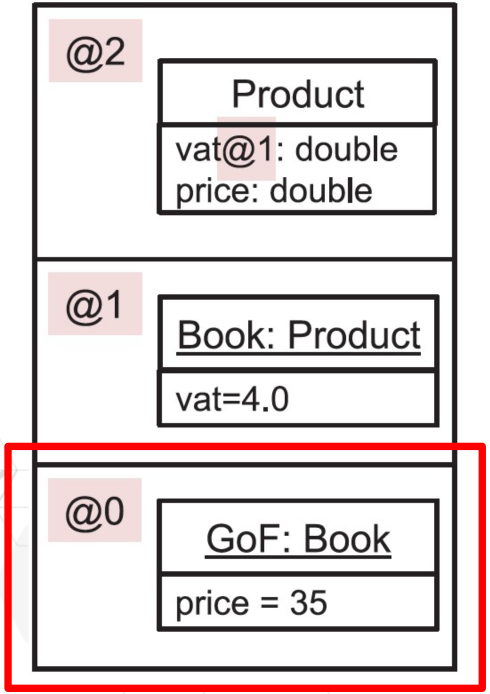
Type-Object pattern (Level 1)



{“Book”, ID_{Book}, $ID_{Product}$, [0, inf], undef,
[
{“vat”, $ID_{ConcreteVat}$, ID_{Vat}, [1, 1], 4,[]},

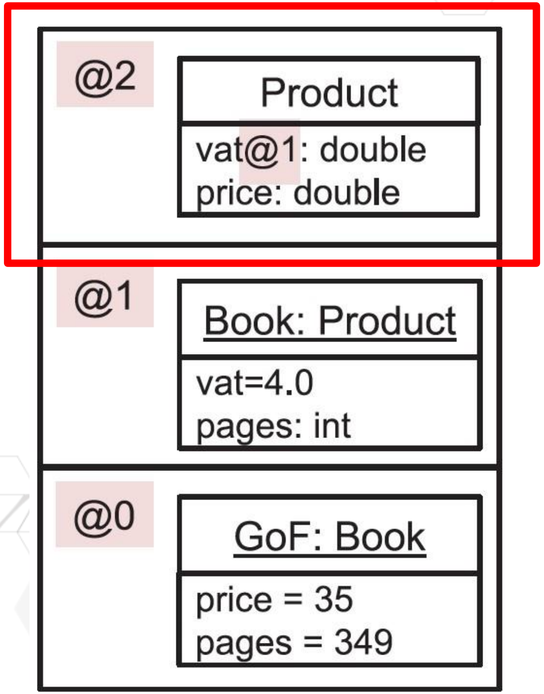
{“price”, ID_{Price}, $ID_{Attribute}$, [1, 1], undef,[
{“priceType”, $ID_{PriceType}$, $ID_{AttribType}$, [0, 1], ID_{Number}, []}
]
}

Type-Object pattern (Level 0)



{“GoF”, $ID_{ConcreteBook}$, ID_{Book}, [0, inf], undef,
[
{“vat”, $ID_{ConcreteVat}$, ID_{Vat}, [1, 1], 4,[]},
{“price”, $ID_{ConcretePrice}$, ID_{Price}, [1, 1], 35,[]}
]
}

Dynamic Features (Level 2)

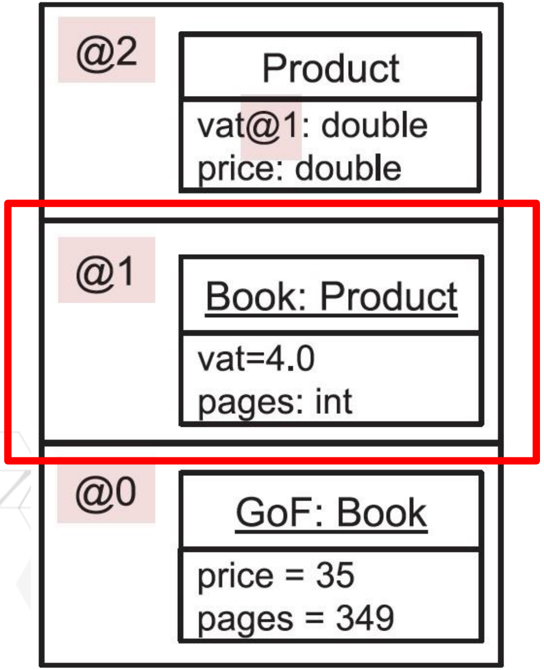


{“Product”, $ID_{Product}$, ID_{Node}, [0, inf], undef,
[
{“vat”, ID_{Vat}, $ID_{Attribute}$, [1, 1], undef,[
{“vatType”, $ID_{VatType}$, $ID_{AttribType}$, [0, 1], ID_{Number}, []},

{“price”, ID_{Price}, $ID_{Attribute}$, [1, 1], undef,[
{“priceType”, $ID_{PriceType}$, $ID_{AttribType}$, [0, 1], ID_{Number}, []},

{“Attributes”, $ID_{Attributes}$, $ID_{Attribute}$, **[0, inf]**, *undef,[]}*
]
}

Dynamic Features (Level 1)

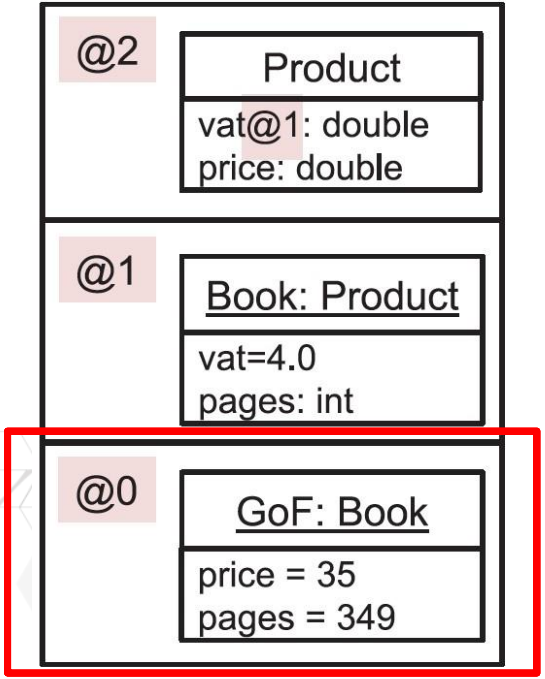


{“Book”, ID_{Book}, $ID_{Product}$, [0, inf], undef,
[
{“vat”, $ID_{ConcreteVat}$, ID_{Vat}, [1, 1], 4,[]},

{“price”, ID_{Price}, $ID_{Attribute}$, [1, 1], undef,[
{“priceType”, $ID_{PriceType}$, $ID_{AttribType}$, [0, 1], ID_{Number}, []},

{“pages”, ID_{Pages}, $ID_{Attribute}$, [1, 1], undef,[
{“pagesType”, $ID_{PagesType}$, $ID_{AttribType}$, [0, 1], ID_{Number}, []}
]
}

Dynamic Features (Level 0)



{"GoF", $ID_{ConcreteBook}$, ID_{Book}, [0, inf], undef,
[
{"vat", $ID_{ConcreteVat}$, ID_{Vat}, [1, 1], 4,[]},
{"price", $ID_{ConcretePrice}$, ID_{Price}, [1, 1], 35,[]},
{"pages", $ID_{ConcretePages}$, ID_{Page}, [1, 1], 349,[]}
]
}

Summary & Future Work

- **Multi-level meta-modelling patterns are well-known, but used in proprietary implementation in design-time & run-time**
- **Dynamic Multi-Layer Algebra is a novel multi-level modelling approach**
  - Precise semantics – defined in ASM notation
  - Flexible constraints – can work with customised Bootstrap entities
  - Dynamic instantiation – abstract selector and action functions (black-box approach)
  - Platform and implementation independent – portable DMLA executor possible
- **Multi-level meta-modelling patterns can be expressed in DMLA**
- **Future work**
  - Experiment with various bootstraps (e.g. reified selectors, operators, *node-edge equality*)
  - Implementation of self-referring DMLA (reified implementation entities in bootstrap)

Thank You!

Any Questions?