The bicycle challenge in DMLA, where validation means correct modeling

Gergely Mezei, Zoltán Theisz, Dániel Urbán, Sándor Bácsi

Budapest University of Technology and Economics
Department of Automation and Applied Informatics

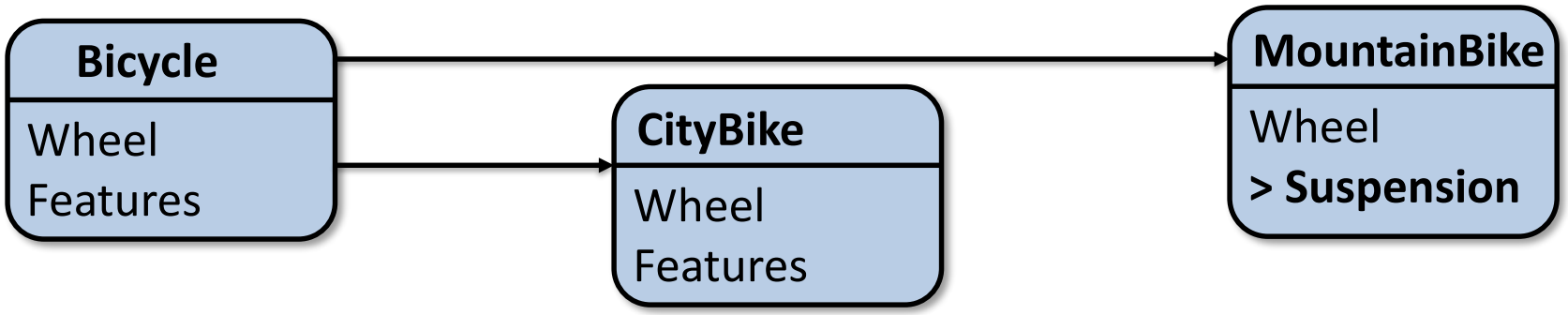
Multi 2018

Dynamic Multi-Layer Algebra

- Core (the “HW”)– Data structure
  - Based on Abstract State Machines
  - Data structure and management
  - 4-tuple $\{X_{ID}, X_{Meta}, X_{values,} X_{Attributes}\}$
- Bootstrap (the “operating system”)
  - Set of entities, enabler of modeling
  - Defines metamodeling foundation
  - Basic building blocks (modelling and operations)
- DMLAScript (the “programming language”)
  - The “sugar”
  - Higher abstraction level interface (no tuples)
  - Always compiled to entities

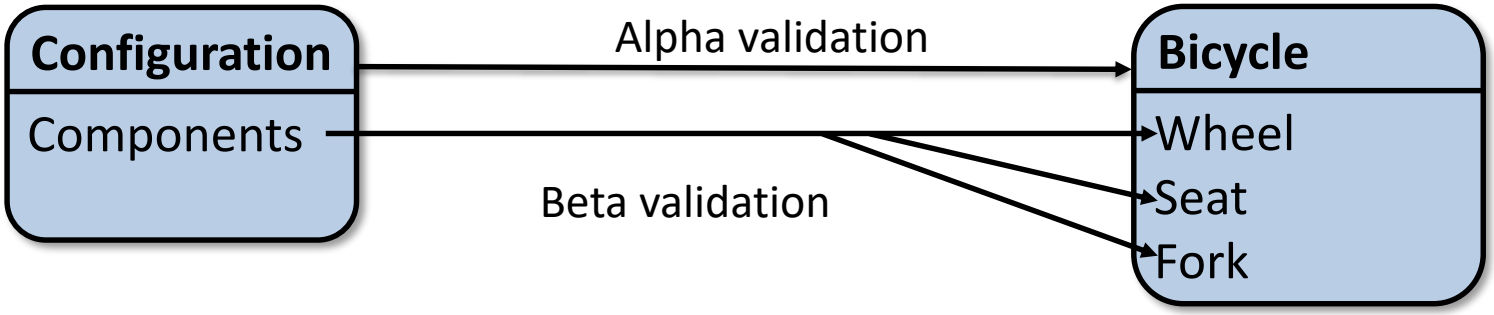
DMLA – Instantiation

- Fluid metamodeling (#Ulrich: RC1, RC2)
  - Intention: support stepwise, partial refinement
    - Concretization everything at once (a whole level) is rigid
  - Entities/attributes are instantiated individually
  - (Partial) Instantiation – mixture of
    - Concretization: the abstraction level is lowered
    - Cloning: the entity/feature remains intact (#Ulrich RC7)



DMLA – Validated operations

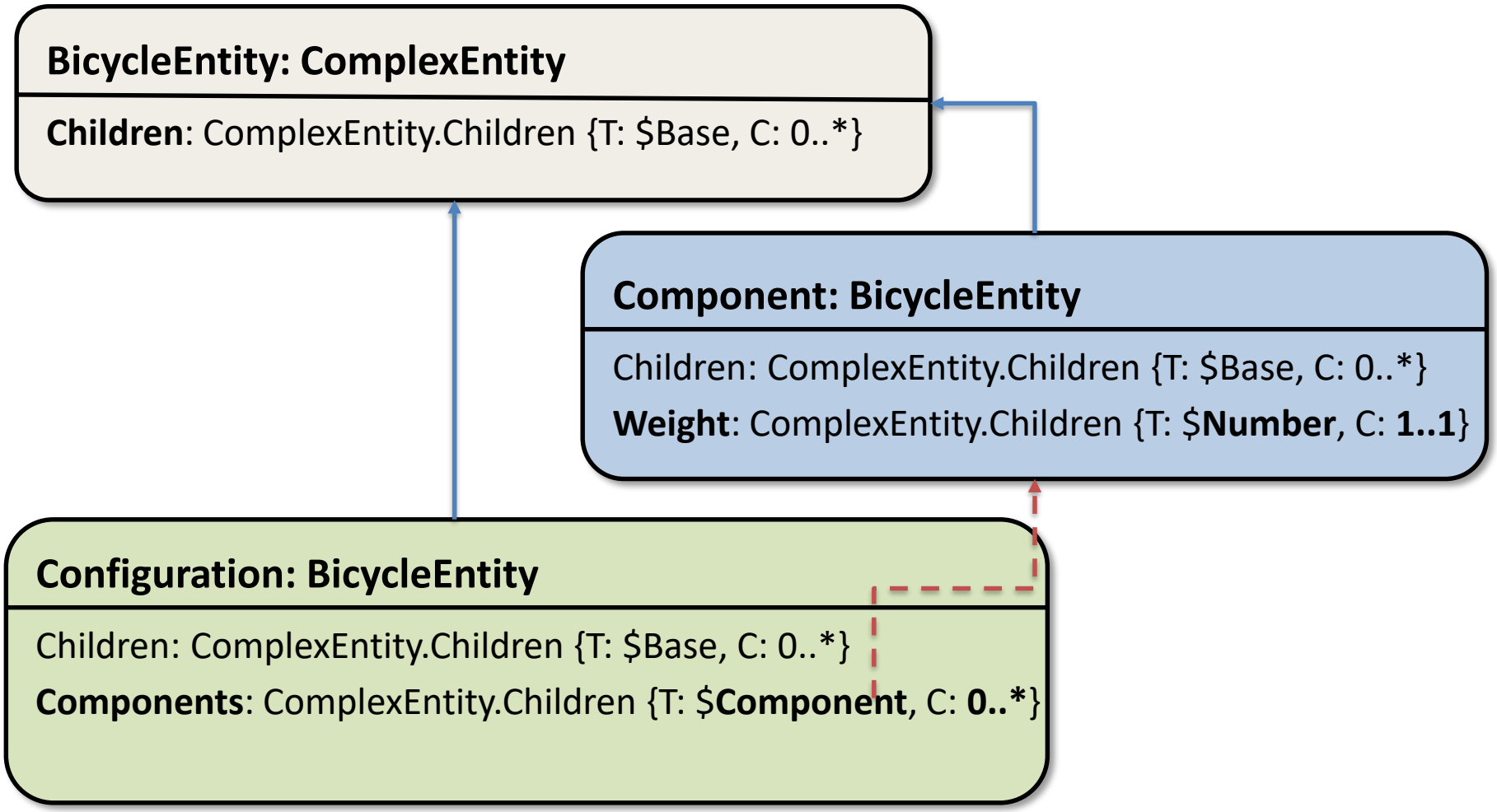
- Goal: self validating bootstrap (without an external language)
  - Key: we need to *model the operations*
  - AST elements → Bootstrap
  - Operation definitions are built from entities
    → A high level script language (DMLAScript) was invented
- Validation formulae
  - *Alpha*: meta – instance (1:1, e.g. type)
  - *Beta*: meta – set of instances (1:n, e.g. cardinality)
  - *Gamma*: instance – whole model (1:*, e.g. uniqueness)
  - Entities can extend their formulae, but *validation is always enforced through the hierarchy up to the root element*



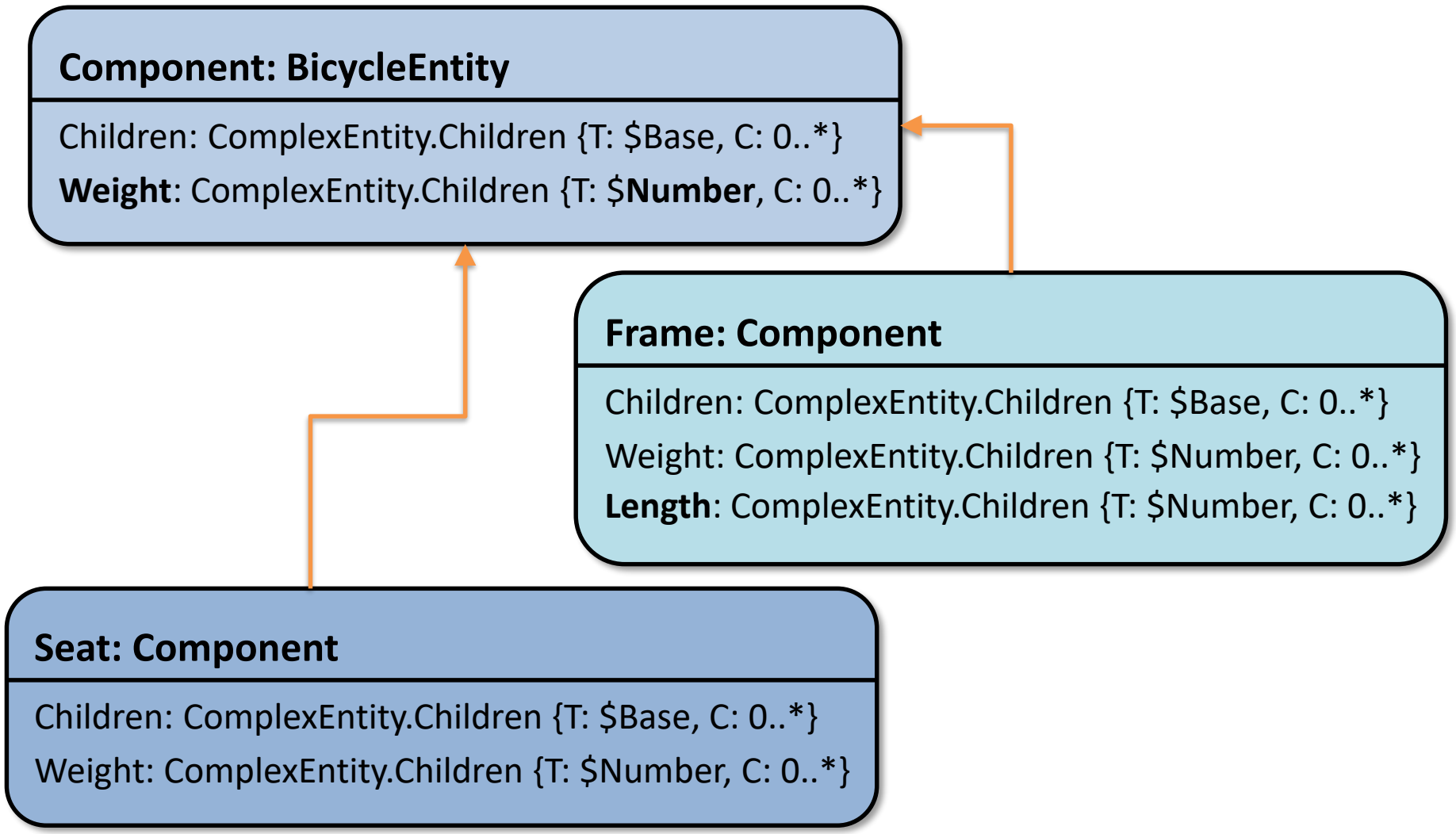
DMLA - slots

- Slots – features of entities
  - Constraints – reusable validation logic
    - Type, Cardinality (#Ulrich: RC3)
    - Operation signature
    - Must-Fill-Once
    - Extendable and fully modeled validation (e.g. filtered cardinality)
  - ComplexEntity.Children
    - Universal type, unlimited cardinality
    - The source of adding new features

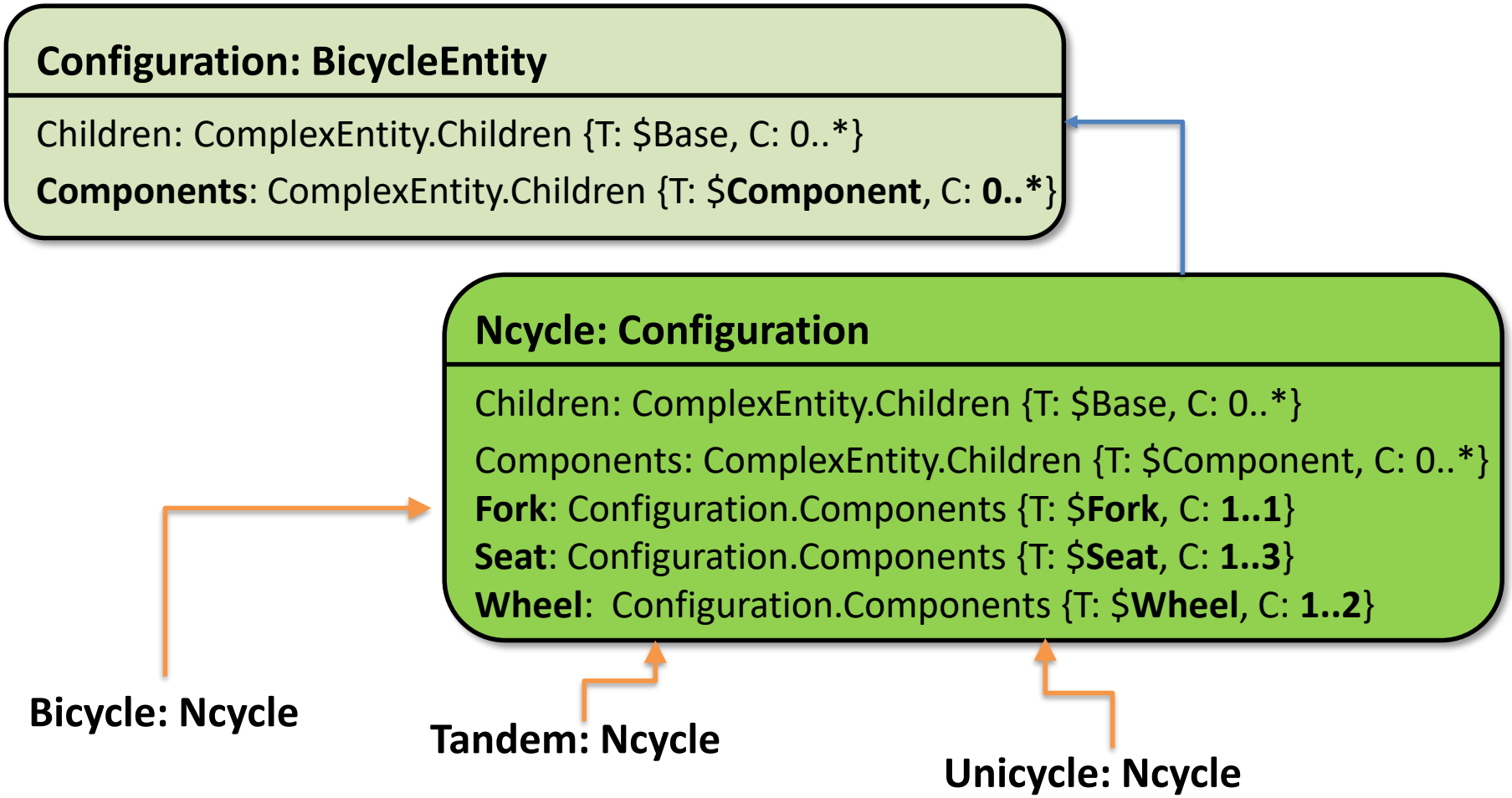
Slots


BicycleEntity: ComplexEntity
Children: ComplexEntity.Children {T: $Base, C: 0..*}
Component: BicycleEntity
Children: ComplexEntity.Children {T: $Base, C: 0..*}
Weight: ComplexEntity.Children {T: $Number, C: 1..1}
Configuration: BicycleEntity
Children: ComplexEntity.Children {T: $Base, C: 0..*}
Components: ComplexEntity.Children {T: $Component, C: 0..*}

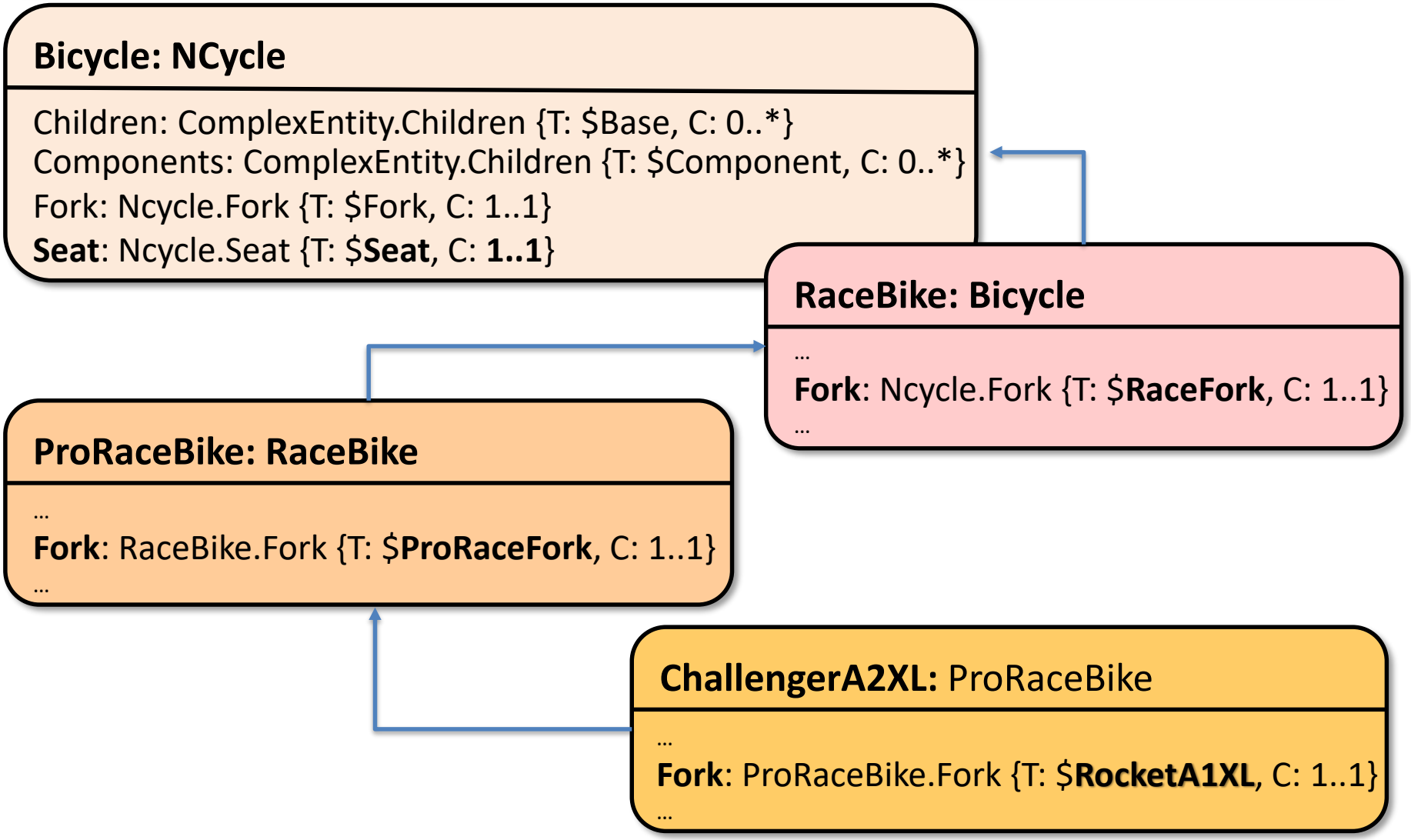
Inheritance “emulation”



Gradual type constraints



Gradual type constraints



Concrete objects

- What does “physical object” mean?
  - You can touch it?
  - Can you touch a concrete bike, or only its components?
  - Is a concrete bicycle without wheels still a bike? Do the wheels still belong to the bike?
  - Serial number of components are unique – but this stands only for concrete components

Concrete objects

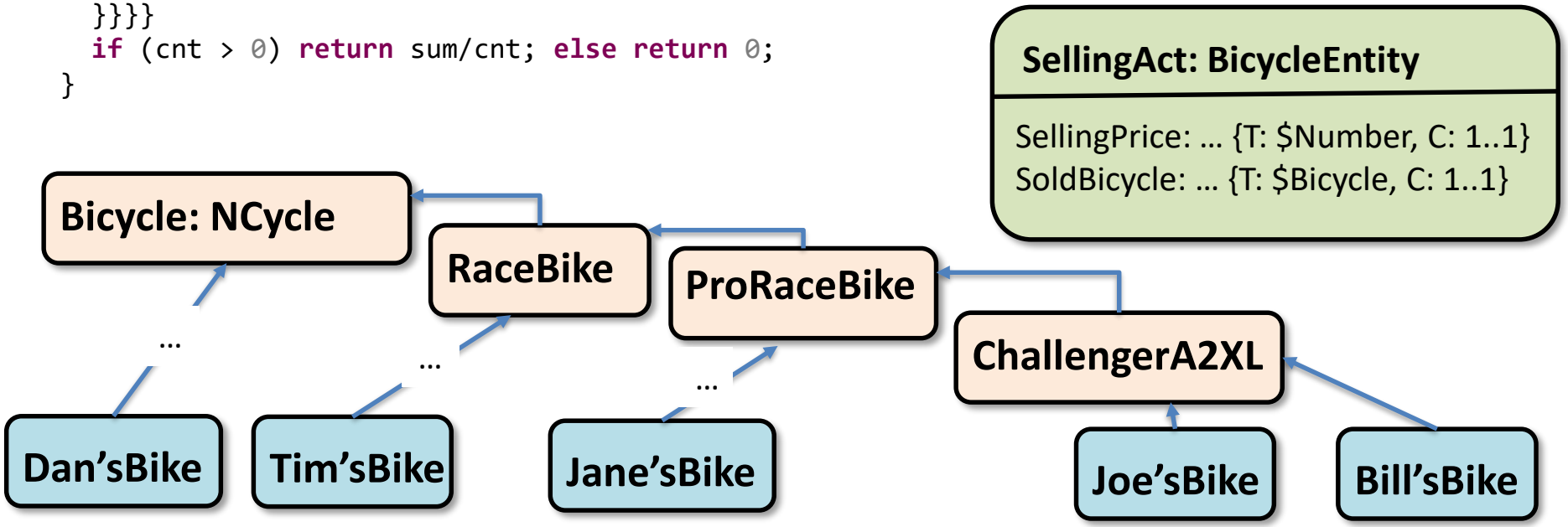
- Concrete objects
  - All primitive slots are filled with a value
  - All non-primitive slots have a concrete value
  - Has no more “free” slots
- Human + DMLA validation
  - Flag-driven validation
  - Concreteness is “claimed”
  - The statement is validated by DMLA

Derived attributes – built-in calculations

- Get average sales price of a
  - …concrete model
  - …a category of models
  - …a type of bicycle
- Why do not we use the instantiation chain?
- Calculation = built-in operation
  - Added on a higher level (Bicycle)
  - Executed on arbitrary level

Derived attributes – built-in calculations

```
operation Number ID::GetAvarageActualSalesPriceMethod()
{
 Number sum = 0; Number cnt = 0;
 forall(entity in GetAllEntities) {
  if (DerivesFrom($SellingAct, entity)) {
   if (GetAttributeValue(entity, $BicycleEntity.AbstractEntity)==null) {
    if(DerivesFrom(this, GetAttributeValue(entity, $SellingAct.SoldBicycle))) {
     cnt = cnt + 1 ;
     sum = sum + GetAttributeValue(entity, $SellingAct.SellingPrice);
     // sum = sum + GetAttributeValue(entity, $Bicycle.SalesPrice);
  }}}}
  if (cnt > 0) return sum/cnt; else return 0;
}
```



Summary

- Thank you for the challenge!
- Solved (almost) all requirements in DMLA
  - Patterns were created during the solution
  - The approach may express more complex scenarios (e.g. complex cardinality)
- Currently working on…
  - VM over DMLA
  - New language over DMLAScript for domain modeling
  - Handle multiple inheritance (diamond pattern)
  - Incremental and parallel validation

Thank You & Any Questions?

Feel free to check the poster as well

Dynamic Multi-Layer Algebra

http://www.aut.bme.hu/Pages/Research/VMTS/DMLA

This work was performed in the frame of **FIEK 16-1-2016-0007** project, implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the FIEK 16 funding scheme. The research has been supported by the European Union, co-financed by the European Social Fund. (**EFOP-3.6.2-16-2017-00013**).