The bicycle challenge in DMLA, where validation means correct modeling

Gergely Mezei, Dániel Urbán, Sándor Bácsi (DAAI, Budapest University of Technology and Economics, Budapest, Hungary)
Zoltán Theisz (evopro Systems Engineering Ltd., Hauszmann Alajos str. 2, Budapest, Hungary)

Step-by-step evolution of a concept

Something

>>

„Ncycle“ (bicycle, unicycle, ...)
- Has wheel, seats, ...
- Has weight

>>

Bicycle
- Two wheels, one seat, ...
- Has weight

>>

Mountain bike
- Constraints on wheels, seats
- Weight is in a range

>>

Model XZ362
- Concrete wheel, seat model
- Weight is set

>>

My „Black Thunder”
- Concrete, physical bicycle
- Fully concretized entity

The DMLA approach

Abstract State Machine

Core

„The hardware”

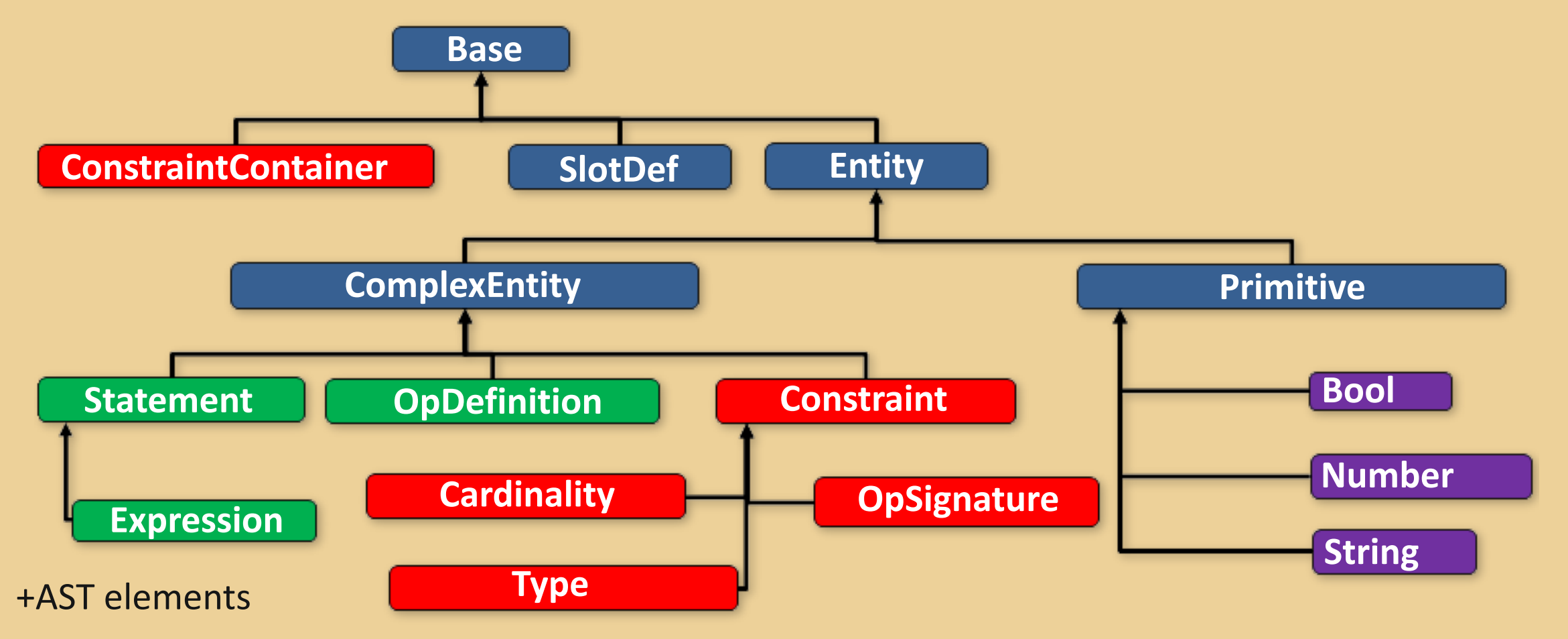
Data structure
- ASM State: snapshot of the model
- Model - nodes, edges, attributes (entities)
- Labeled directed graph
- Entities have labels – entities are 4-tuples

ID:	unique ID
Meta:	reference to meta element
Value:	concrete values
Attributes:	list of contained entities

Management
- Derived functions – obtaining information
- Shared functions – snapshot evolution

Bootstrap | Bootstrap2 | Bootstrap3

„The operating system”
- Set of entities - enabler of multi-level meta-modeling
- Flexible and swappable
- Multi-purpose meta hierarchy



DMLAScript

„The programming language”
- Hides tuples, more user-friendly
- Entities & relations *(Context illusion for related tuples)*
- *Operations (User-friendly operation language)*
  - Xtext-based
  - Fully modeled by 4-tuples
  - Described by Abstract Syntax Tree (AST) built from entities (statements, expressions, if, for, variable, ...)
  - OperationDefinition with signature constraints (return type, parameters, ...)
  - OperationCall

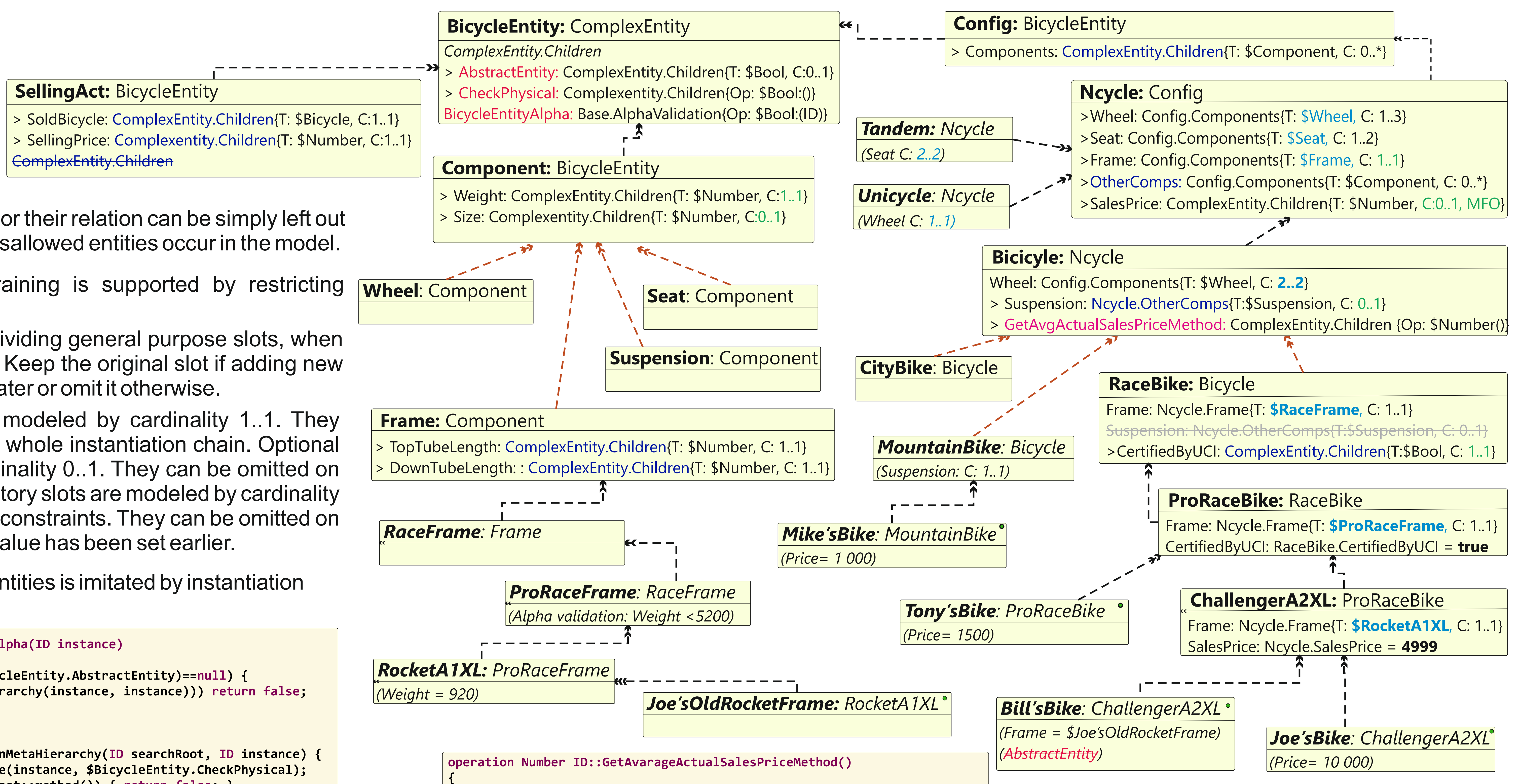
Validation
- All entities support validation based on formulae
- Alpha: meta against instance /e.g. type conformance/
- Beta: meta against entities (in-context check) /e.g. cardinality/
- Gamma: all instances of a certain entity /e.g. unique serial number/
- Validation mechanism:
  1. Loop over all entities
  2. Obtain meta by the Meta ASM function
  3. Fetch all alphas – hierarchy chain up to Base
  4. Combine alphas (meta, instance) by AND
  5. Combine entity validations by AND

Fluid multi-level metamodeling
- Fluid levels
  - Stepwise refinement of domain concepts is decoupled
  - Meta - Instance is interpreted between two entities
  - Entities can refer to other entities on any levels
  - Consequence of real multi-level modeling
  - Entities are instantiated individually
- Partial instantiation
  - Stepwise refinement: attributes are entities -> also independent
  - Some of the features remain untouched, others are concretized
  - Entity instantiation = copy or concretize slots

Slots
- General placeholder for data (e.g. fields/properties)
- Constraints attached to slots
  - Type constraint
  - Cardinality
  - Operation signature
  - Must Fill Once
  - Self-described & extendable
- ComplexEntity.Children
- Stepwise concretization/constraining

Patterns

Notations
- ·····» Instantiation
- ● Concrete object
- X: Y Entity sketch only


SellingAct: BicycleEntity
> SoldBicycle: ComplexEntity.Children{T: $Bicycle, C:1..1}
> SellingPrice: Complexentity.Children{T: $Number, C:1..1}
~~ComplexEntity.Children~~
BicycleEntity: ComplexEntity
ComplexEntity.Children
> AbstractEntity: ComplexEntity.Children{T: $Bool, C:0..1}
> CheckPhysical: Complexentity.Children{Op: $Bool:()}
BicycleEntityAlpha: Base.AlphaValidation{Op: $Bool:(ID)}
Config: BicycleEntity
> Components: ComplexEntity.Children{T: $Component, C: 0..*}
Component: BicycleEntity
> Weight: ComplexEntity.Children{T: $Number, C:1..1}
> Size: Complexentity.Children{T: $Number, C:0..1}
Tandem: Ncycle
(Seat C: 2..2)
Unicycle: Ncycle
(Wheel C: 1..1)
Ncycle: Config
>Wheel: Config.Components{T: $Wheel, C: 1..3}
>Seat: Config.Components{T: $Seat, C: 1..2}
>Frame: Config.Components{T: $Frame, C: 1..1}
>OtherComps: Config.Components{T: $Component, C: 0..*}
>SalesPrice: ComplexEntity.Children{T: $Number, C:0..1, MFO}
Wheel: Component
Seat: Component
Suspension: Component
Frame: Component
> TopTubeLength: ComplexEntity.Children{T: $Number, C: 1..1}
> DownTubeLength: : ComplexEntity.Children{T: $Number, C: 1..1}
Bicicyle: Ncycle
Wheel: Config.Components{T: $Wheel, C: 2..2}
> Suspension: Ncycle.OtherComps{T:$Suspension, C: 0..1}
> GetAvgActualSalesPriceMethod: ComplexEntity.Children {Op: $Number()}
CityBike: Bicycle
RaceBike: Bicycle
Frame: Ncycle.Frame{T: $RaceFrame, C: 1..1}
~~Suspension: Ncycle.OtherComps{T:$Suspension, C: 0..1}~~
>CertifiedByUCI: ComplexEntity.Children{T:$Bool, C: 1..1}
MountainBike: Bicycle
(Suspension: C: 1..1)
RaceFrame: Frame
Mike'sBike: MountainBike
(Price= 1 000)
ProRaceBike: RaceBike
Frame: Ncycle.Frame{T: $ProRaceFrame, C: 1..1}
CertifiedByUCI: RaceBike.CertifiedByUCI = true
ProRaceFrame: RaceFrame
(Alpha validation: Weight <5200)
Tony'sBike: ProRaceBike
(Price= 1500)
ChallengerA2XL: ProRaceBike
Frame: Ncycle.Frame{T: $RocketA1XL, C: 1..1}
SalesPrice: Ncycle.SalesPrice = 4999
RocketA1XL: ProRaceFrame
(Weight = 920)
Joe'sOldRocketFrame: RocketA1XL
Bill'sBike: ChallengerA2XL
(Frame = $Joe'sOldRocketFrame)
(~~AbstractEntity~~)
Joe'sBike: ChallengerA2XL
(Price= 10 000)

P0: Prohibition of features or their relation can be simply left out since validation will fail if disallowed entities occur in the model.

P1: Gradual type constraining is supported by restricting constraints on slots.

P2: Create new slots by dividing general purpose slots, when new features are needed. Keep the original slot if adding new features may be required later or omit it otherwise.

P3: Mandatory slots are modeled by cardinality 1..1. They must be kept all along the whole instantiation chain. Optional slots are modeled by cardinality 0..1. They can be omitted on any level. Optional-mandatory slots are modeled by cardinality 0..1 and the MustFillOnce constraints. They can be omitted on any level supposing their value has been set earlier.

P4: Inheritance between entities is imitated by instantiation

```
operation Bool ID::BicycleEntityAlpha(ID instance)
{
 if (GetAttribute(instance, $BicycleEntity.AbstractEntity)==null) {
  if(!(CallCheckPhysicalOnMetaHierarchy(instance, instance))) return false;
 return true;
}

operation Bool CallCheckPhysicalOnMetaHierarchy(ID searchRoot, ID instance) {
 Object method = GetAttributeValue(instance, $BicycleEntity.CheckPhysical);
 if(method!=null && !call searchRoot::method()) { return false; }
 ID meta = call $Meta(instance);
 if(meta== $BicycleEntity) return true;
 return CallCheckPhysicalOnMetaHierarchy(searchRoot, meta);
}
```

P10: Custom validation can be driven by flags. If the flag is presented, the validation is turned off, if the flag is omitted, the validation is switched on.

```
operation Number ID::GetAvarageActualSalesPriceMethod()
{
 Number sum = 0; Number cnt = 0;
 foreach(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;
}
```

P11: Derived properties are added to entities as operations. In case of summation, the instantiation chain can be used in a layer-transparent fashion.

P5: Enum and bitfield-like requirements are modeled as an enum type with its slot-less instances.

P6: Slots can represent signature-driven operation definitions at all concretization levels.

P7: Alpha formula completes an entity's type semantics beyond meta-hierarchy.

P8: Global validation requirements are satisfied by gamma formulas.

P9: Soft validaton, i.e., filtering features are supported by operations attached to the entities.

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

The Bicycle challenge:
http://www.wi-inf.uni-duisburg-essen.de/MULTI2018

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).