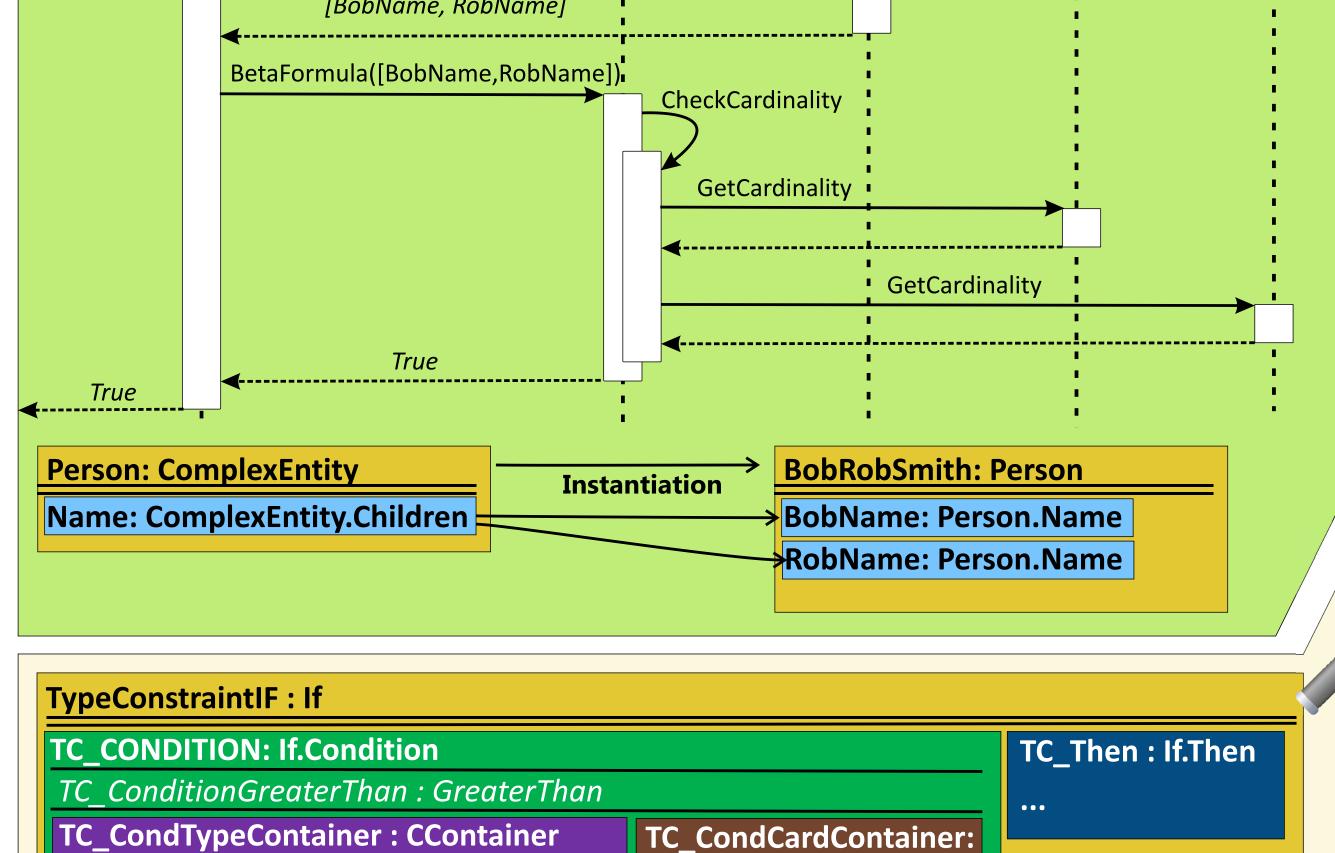
Self-Describing Operations for Multi-level Meta-modeling



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



Bootstrap2 Bootstrap3 Bootstrap Core "The operating system" "The hardware" Set of entities - enabler of multi-level meta-modeling Data structure Abstract Flexible and swappable ASM State: snapshot of the model Multi-purpose meta hierarchy Model - nodes, edges, attributes (entities) Core entities – Basic structure of modeling Labeled directed graph Primitives – Concrete, atomic values Entities have labels → entities are 4-tuples Constraint handling entities – Validation framework State ID: unique ID AST entities – Operation structure reference to meta element Meta: Value: Base concrete values Attributes: list of contained entities ConstraintContainer **Entity SlotDef** achine Management **Primitive** ComplexEntity Derived functions – obtaining information • Shared functions – snapshot evolution Bool **OpDefinition** Constraint **Statement** Label manipulation, entity creation/deletion Number **Cardinality OpSignature** Modifications internal (by algebra) **Expression** String Modifications external (by the user) **Type** +AST elements Tuples are visualized by DMLAScript "Bricks" of operations are entities, everything is modelled **DMLAScript** "The programming language" Operations (User-friendly operation language) Entities & relations (Context illusion for related tuples) Xtext-based Fully modeled by 4-tuples Entity1: Meta1 { Described by Abstract Syntax Tree (AST) built from entities @CContainer: Meta1.MSlot.CntConstrContainer = (statements, expressions, if, for, variable, ...) **CConstraint**: Meta1.MSlot.CntConstrContainer.Constr { slot CValue: Meta1.MSlot.CntConstrContainer.Constr.Value = 1; OperationDefinition with signature constraints (return type, parameters, ...) slot ESlot: MSlot = true; OperationCall X_{values} Xattribute ESlot2; Validation "The kernel protection mechanism" **Entity1**: Meta1 Validation elements **ESlot: MSlot** ESlot2 true • All entities support validation based on formulae alpha & beta **CContainer: CntConstrContainer** • Alpha: meta against instance /e.g. type conformance/ **CConstraint: Constr** • Beta: meta against entities (in-context check) /e.g. cardinality/ **CValue: Value** Validation mechanism: 1. Loop over all entities 2. Obtain meta by the Meta ASM function 3. Fetch all alphas – hierarchy chain up to Base Example: validation sequence 4. Combine alphas (meta, instance) by AND BobName: BobRobSmith: RobName: 5. Combine entity validations by AND Person.Name Person Person.Name Person.Name Person Constraints AlphaFormula GetRelevantAttributes(Person.Name) Recurring logic in validation (BobRobSmith) [BobName, RobName] (e.g. Type - Cardinality - Operation signature) BetaFormula([BobName,RobName]) Attached to slots CheckCardinality • Constraint alpha&beta: validate the slot the constraint is attached to GetCardinality • Lifecycle alpha&beta: manage the lifecycle of the constraint



ID type = call this::\$Helper.GetTypeType(); Bool isInclusive = call this::\$Helper.GetTypeIsInclusive(); Bool unfilled = type==null || isInclusive==null; ID[] typeConstraints = call instance::\$Helper.GetRelevantConstraints(\$Bootstrap.Type); if(s: ze(typeConstraints)>1) return false; Objet[] values = call \$Values(instance); values==null || size(values)==0) return true; if(unfilled && size(typeConstraints) == 0) return false; ID actualType = type; Bool actualIsInclusive = isInclusive; if(size(typeConstraints) == 1) { ID instTypeConstraint = index<ID>(typeConstraints, 0); actualType = call instTypeConstraint::\$Helper.GetTypeType(); actualIsInclusive = call instTypeConstraint::\$Helper.GetTypeIsInclusive(); if(actualType==null || actualIsInclusive==null) return false; for(Object value : values) { if (actualIsInclusive) { if(!call \$DerivesFromOrEquals(actualType, value)) return false; if(!call \$DerivesFrom(actualType, value)) return false; return true;

CContainer

TC CondTypeConstraint: TypeConstraint

TC_AlphaSlot: ConstraintAlpha

TypeConstraintCAlpha

operation Bool ID::TypeConstraintCAlpha(ID meta, ID instance)

Example: validation code