Formalization of object constraints in AutomationML

Josef Prinz

inpro

3d AutomationML user conference

October 7th. 2014

This work is supported by the BMBF (german Ministry of Education and Research), funding code 01M3204A, as part of the project “EfA: Entwurfsmethoden für Automatisierungssysteme mit Modellintegration und automatischer Variantenbewertung” (2012-2015)

What are constraints

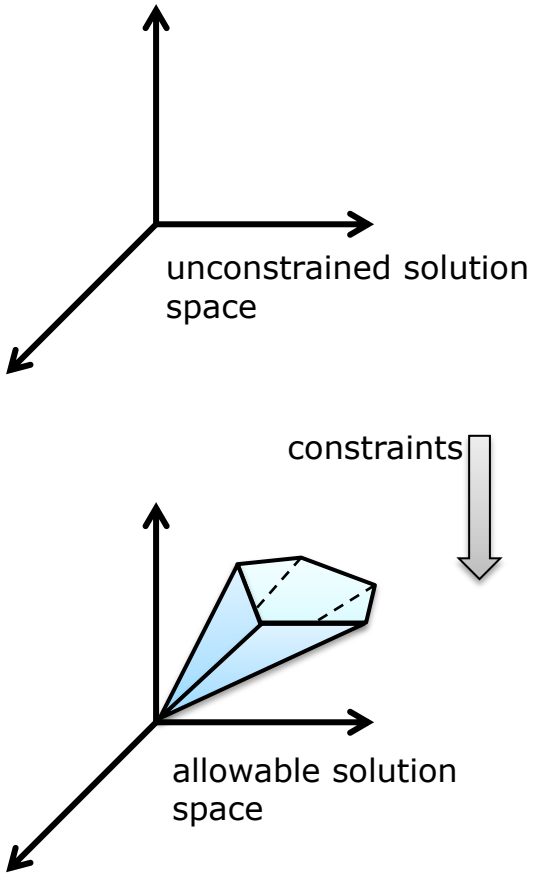
- **Wikipedia**
  - A constraint is something that plays the part of a physical, social or financial restriction.
- **Constraints in Engineering**
  - Constraints are conditions that we need to happen or would like to happen with a design (model).
  - For example, the size of all the manufacturing equipment in a workcell cannot exceed the overall size of the cell.

Loading Gauge, Structure Gauge Constraint:

The clearance between train and tunnel is often small in London Underground.

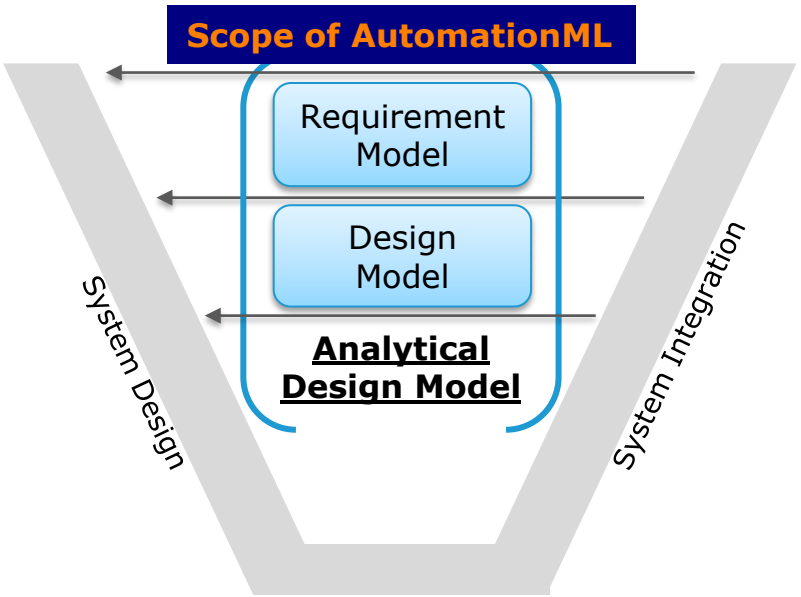
What are constraints

- **Feasible Designs**
  - A feasible Design is a design, where all constraints are satisfied
- **Constraint Sources in Engineering**
  - Requirement Specifications
  - Design Guides
  - Safety Regulations
  - Technology Constraints
  - Component / Interface Compatibility



Why constraints are useful in AutomationML

- AutomationML transports not only "data" but **Design Models**
- AutomationML connects not only tools but **Design Processes**
- Information flows between Design Processes include requirement specifications
- The Implementation of requirements in a design will be reviewed and verified by a client
- AutomationML should be capable of transporting design- and associated requirement data

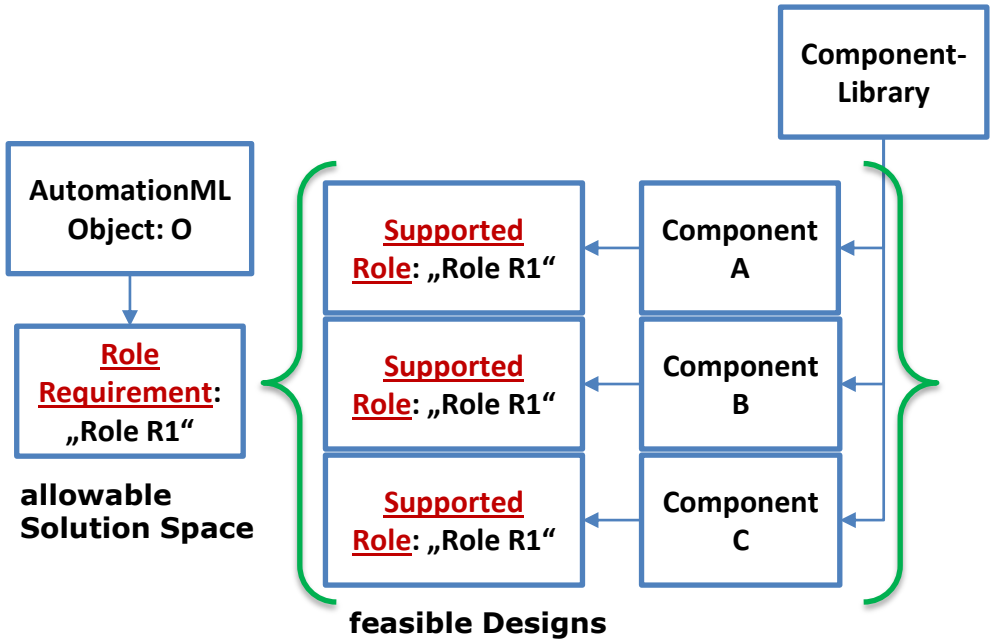


in VDI2206: Design methodology for mechatronic Systems

simplified view

AutomationML is capable to model Constraints

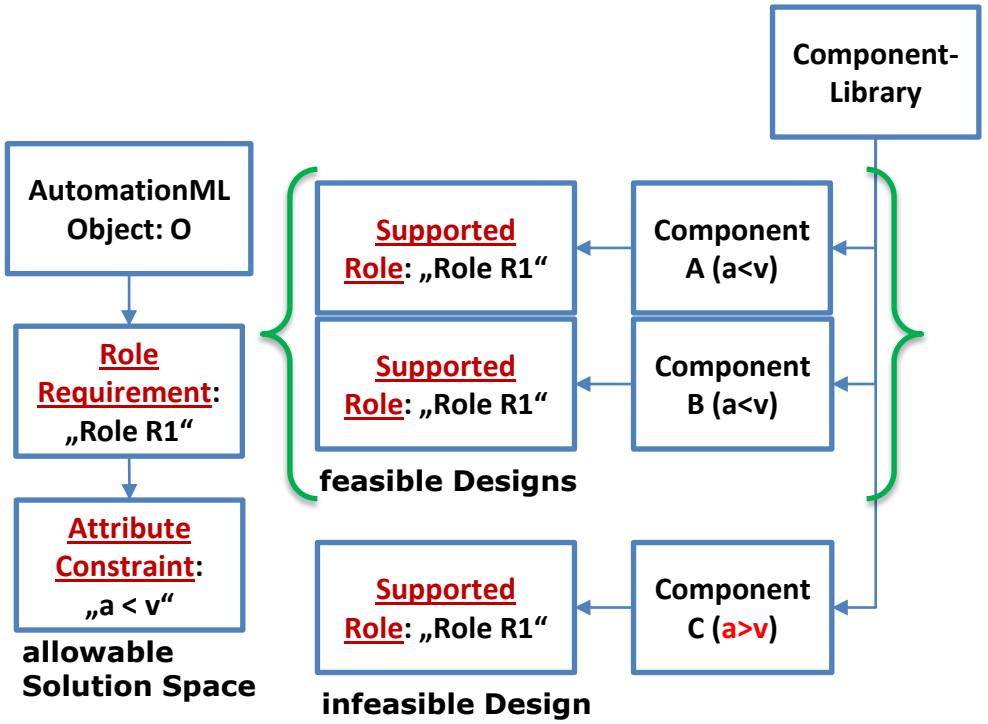
- **RoleRequirements**
  - RoleRequirements in AutomationML restrict the allowable solution space on the component level
  - For Example a Work cell-Model Element with an *InternalElement* with RoleRequirement "Robot" requires a Component with a SupportedRoleClass "Robot"



Constrained Solution Space with AutomationML RoleRequirements

AutomationML is capable to model Constraints

- **Attribute Constraint**
  - An Attribute Constraint defines Restrictions for the allowable Values of an Attribute
  - For Example a Work cell-Model Element with an *InternalElement* with RoleRequirement "Robot" and a minimal load capacity.



Constrained Solution Space with AutomationML Attribute Constraint

AutomationML needs extensions to model more types of design constraints

- **Dependency Constraints**
  - Structure Gauge > Load Gauge + ε
- **Configuration Constraints**
  - Component A can’t be used together with Component B in the same design
- **Functional Constraints**
  - The Sum of all values x of all Attributes of Components of Type T should always be lower then value y of Component C.

Loading Gauge, Structure Gauge Constraint

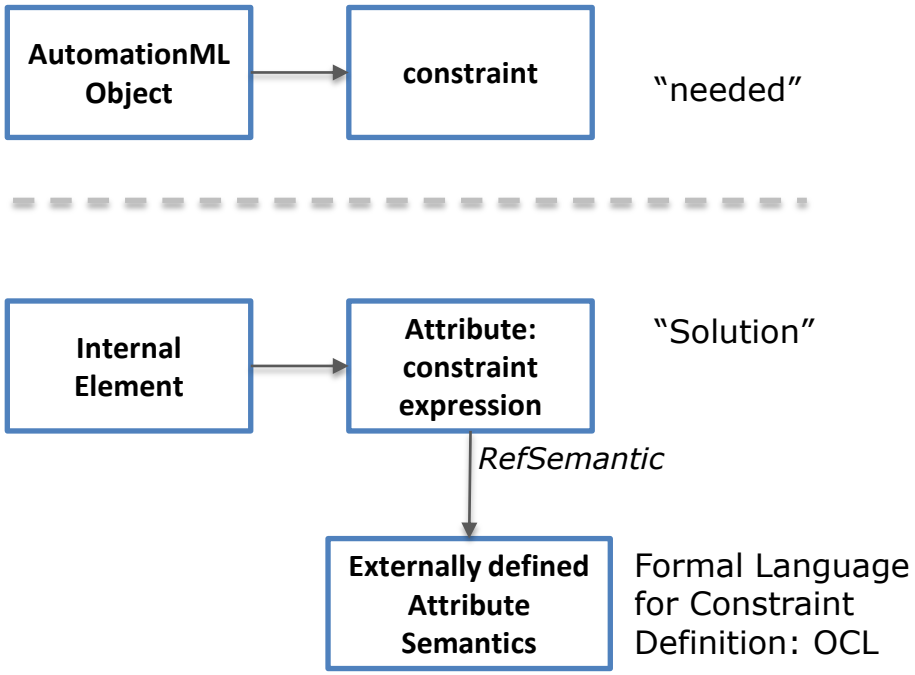
Constraint Models in AutomationML

- **Constraint Association**
  - Constraints should be associated to Elements of the Design Model
- **Constraint Evaluation**
  - Constraints should be checkable (with a software tool)
- **Constraint Context**
  - Constraint should be expressed in terms of the Design Model

Constraint Violation Example

AutomationML can be extended

- **Constraint Association**
  - AutomationML allows the reference of externally defined element-semantics
- **Constraint Evaluation**
  - OCL "Object Constrained Language" is a Formal Language for Constraint Definitions available as an OMG Standard
- **Constraint Context**
  - OCL "Object Constrained Language" has a Model Binding (to UML - Models)



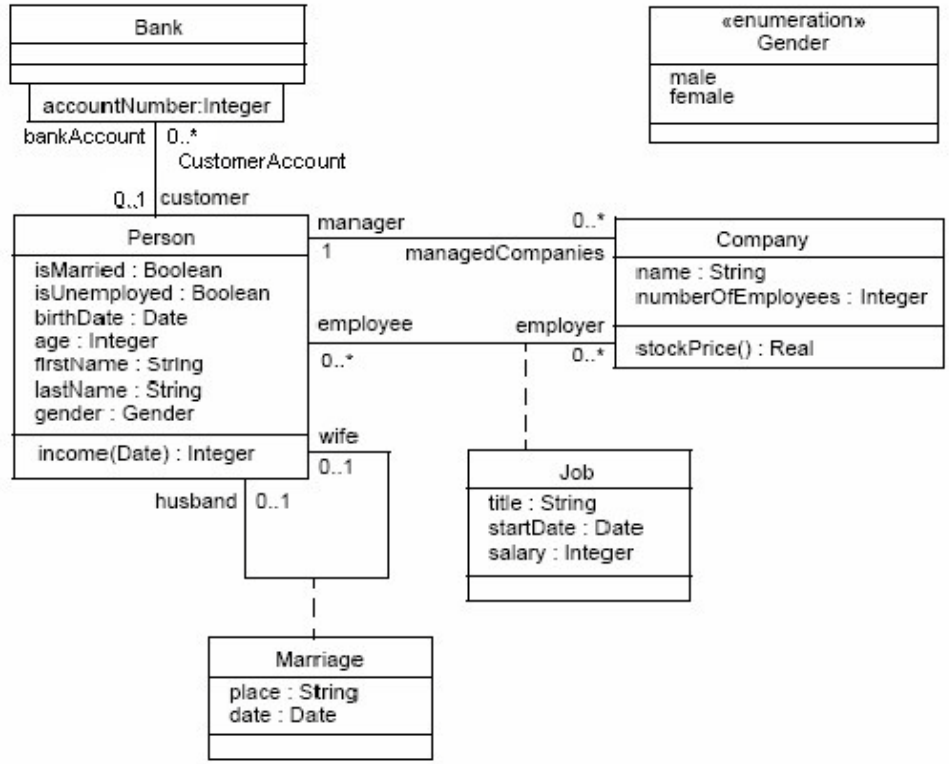
http://www.omg.org/spec/OCL/

Object Constraint Language (OCL)

- OCL is a formal Language for the specification of Model Constraints
- OCL is part of UML (Unified Modeling Language)
- OCL is OMG Standard and ISO Standard ISO/IEC 19507
- OCL is “a formal language that remains easy to read and write” (according to the documentation)
- OCL is embedded in UML, so it is used for associations with design models. It *could work* for AutomationML Models as well.
- With OCL, you can’t induce any changes into the design models.
- OCL current Version is 2.4, released on February 2014
  – difficult to parse, our Tool uses Version 2.3.1
- For Version 2.5 a BNF Grammar is announced
- OCL Tools are available in the JAVA World (Eclipse Plugins: EMF, Dresden OCL)

OCL Integration in AutomationML

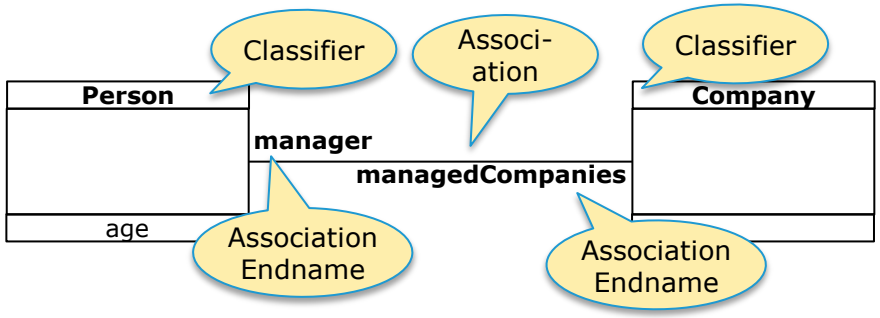
- AutomationML Models are not UML-Models
- Binding Alternatives
  - Usage of Elements of the CAEX Meta Model in OCL Constraints
    - **let E = *InternalElement*(*Name*= 'E')**
    - **let a = E.*Attribute* (*Name*='a')**
    - **a.*Value* < 100**
  - Usage of Elements of the AutomationML Model in OCL Constraints
    - **E.a < 100**
- How can we create an AutomationML Model – OCL Binding?



UML Class Diagram used in the OCL-Documentation

OCL Integration in AutomationML

- **OCL – AutomationML Binding**
  - Nearly no Class to Class Associations in AutomationML
  - Associations between AutomationML Objects in a Design Model are produced in engineering processes.
  - The Associations relate AutomationML Objects (Instances)
  - There are "no" Association-Endnames in AutomationML Associations

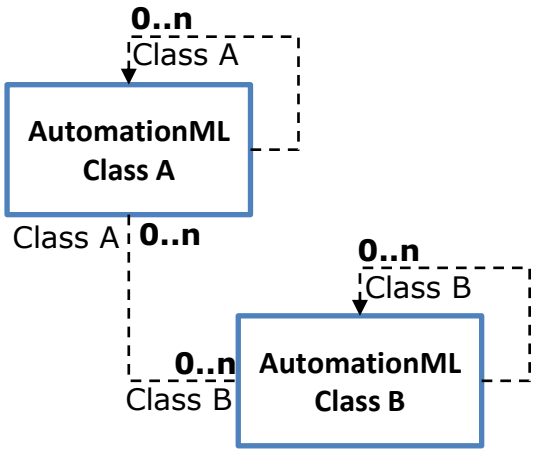


- **OCL – UML Binding**
  - Names of Classes are Classifiers
  - OCL Constraints are defined for Classes (with a context-Definition)
  - For the Navigation to associated Objects, the opposite Association-Endnames are used, for example:

```
context Company
inv:
self.manager.agey<55
```

OCL AutomationML – Binding Definitions

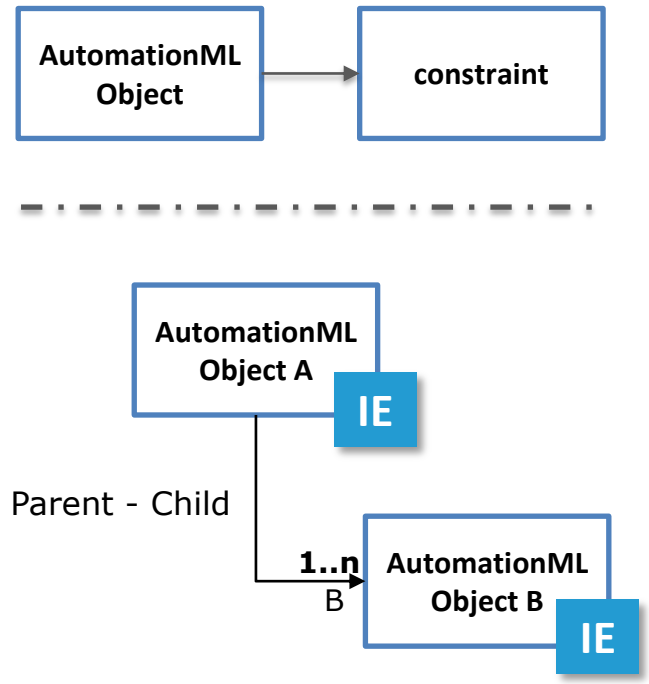
1. All AutomationML Classes (SystemUnitClass, RoleClass) can be used in Classifiers
   - The Class-Model defines an "unconstrained solution space"
   - In an "unconstrained solution space" each class may be associated with any other class and itself. This results in **n * (n + 1) / 2 possible Associations**
2. The OCL-implicit Namingrule is applied to get Association Endnames: "If no explicit name is available an implicit name is taken from **the name of the class** to which the end is attached ".



Implicit Definitions of AutomationML Class to AutomationML Class Associations

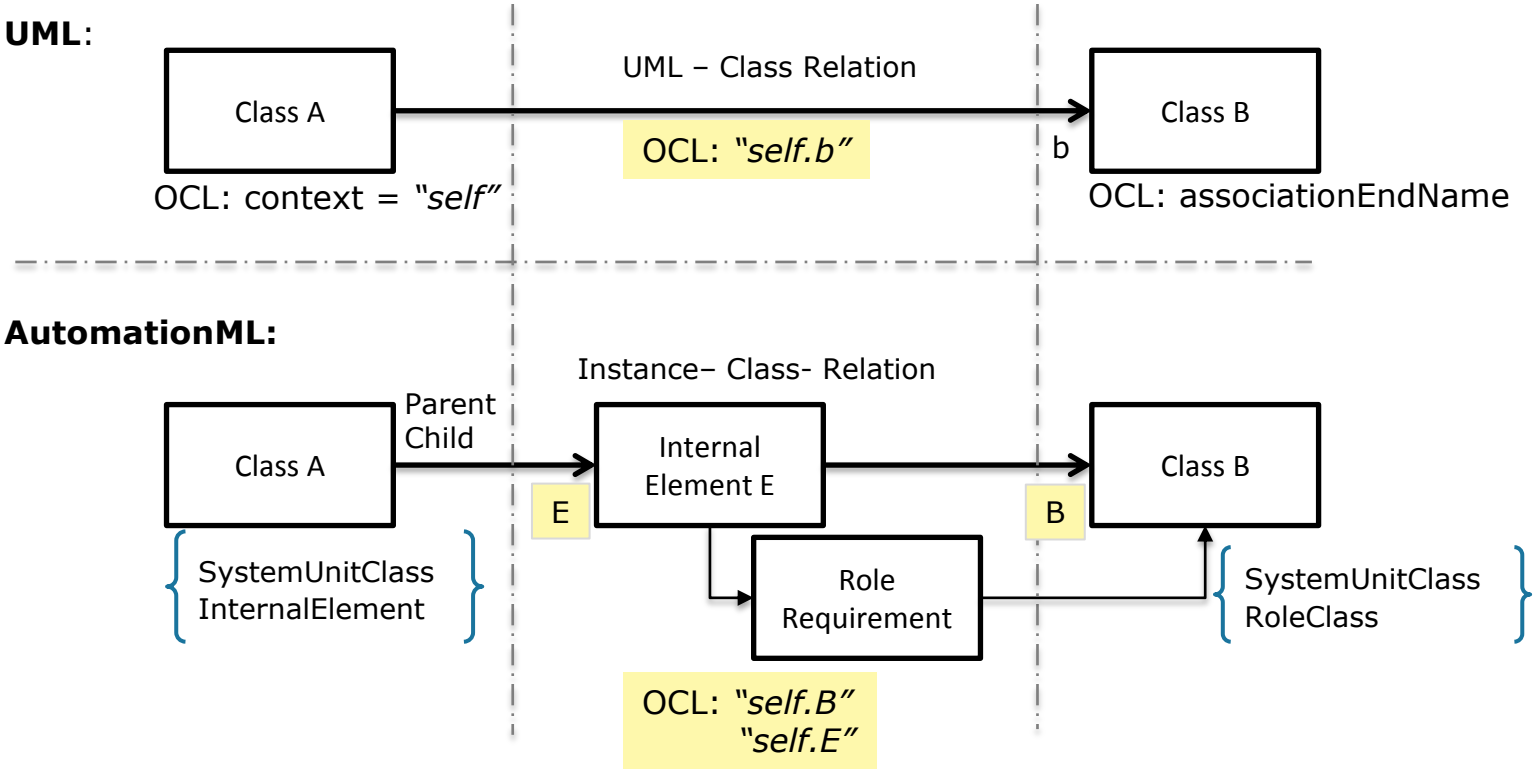
OCL AutomationML – Binding Definitions

3. Names of AutomationML Objects are valid Context Classifiers
   - This is possible, because the AutomationML Object (InternalElement) has direct Associations to OCL-Constraints.
4. Names of Child-AutomationML Objects are valid Association Endnames (opposite End of a parent child relation, seen from the Parent)



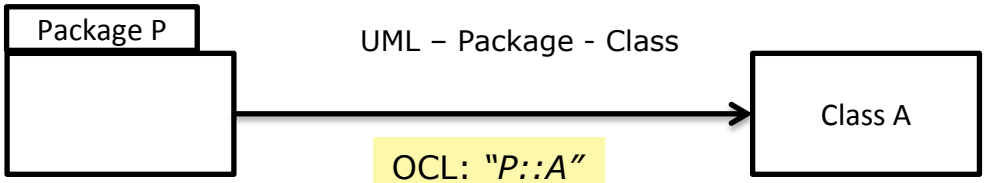
Explicit Associations in an AutomationML-Model defined with Parent-Child-Relations

OCL AutomationML – Binding



OCL AutomationML – Binding

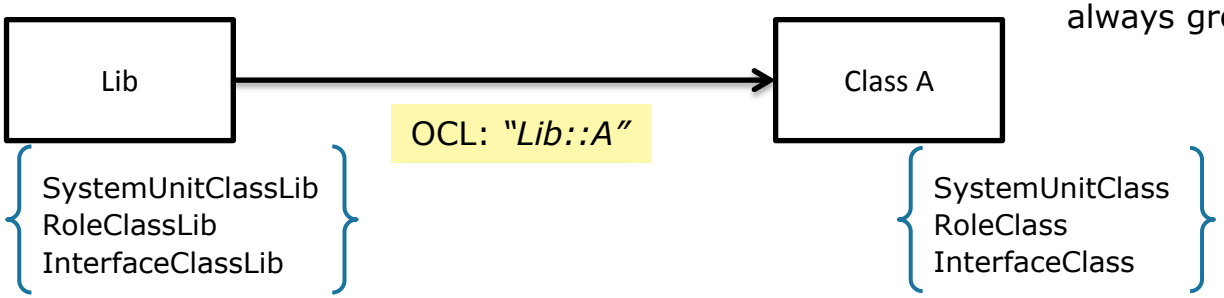
UML:



Classes may be grouped in Packages

AutomationML:

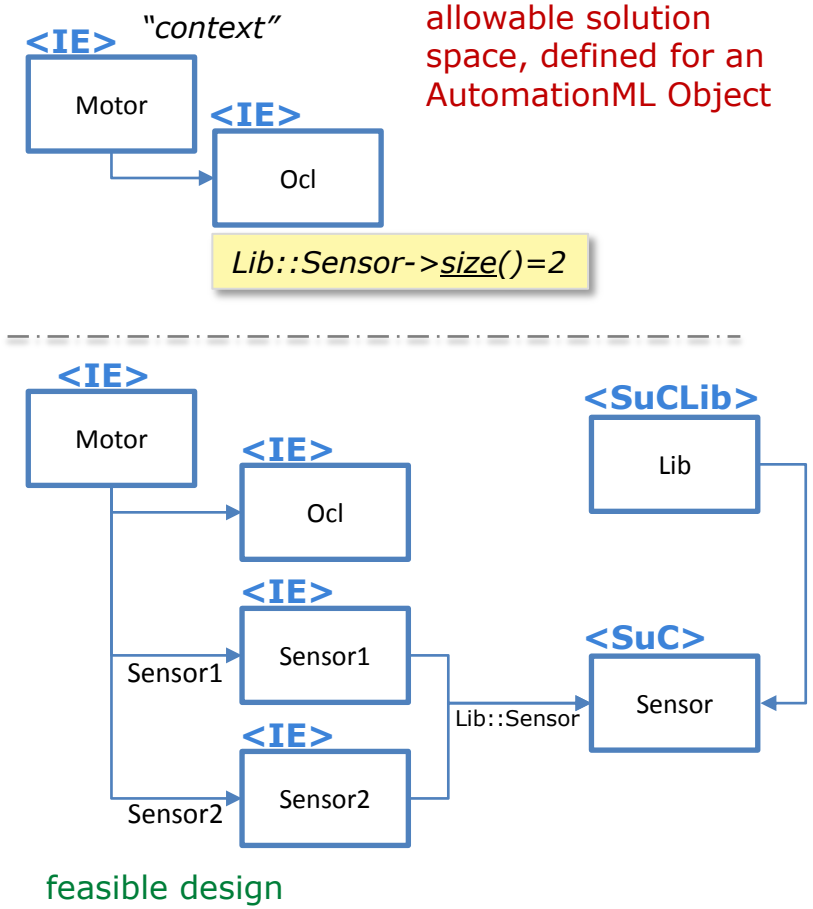
AutomationML Classes are always grouped in Libraries



5. Names of AutomationML Libraries are Package Names for the Classifiers of the AutomationML Classes in the Library

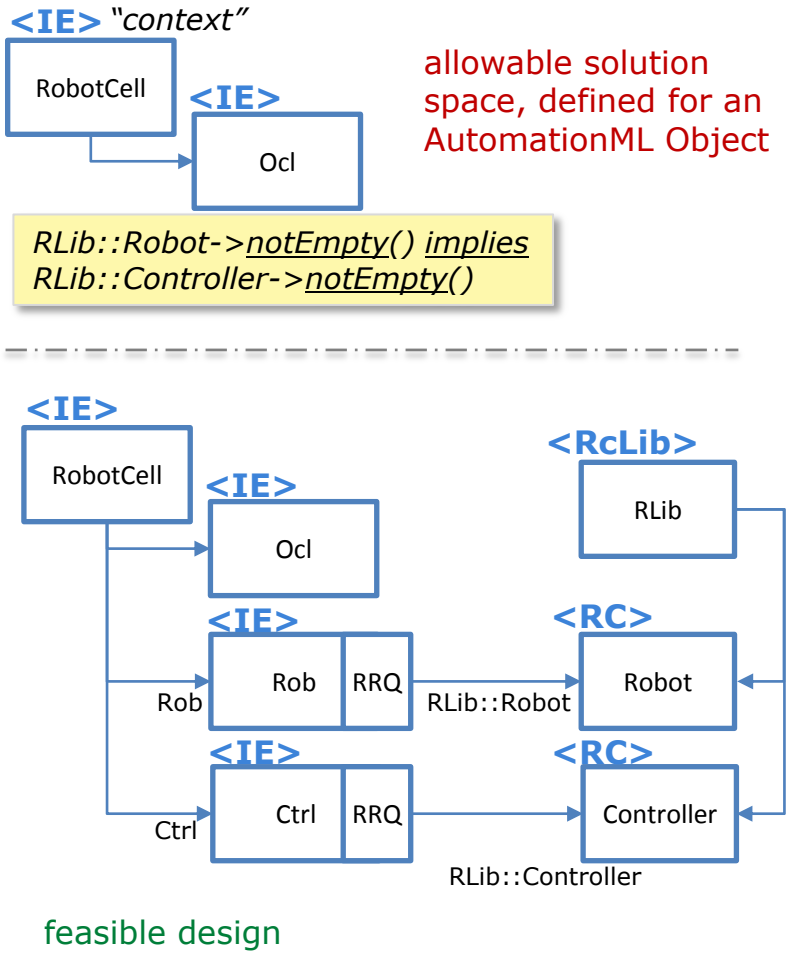
Some Examples

- **Configuration Constraint**
  - „->“ is a Set operator
  - *size* returns the number of elements in the set
  - ‚Sensor‘ defines the associated class in the library ‚Lib’
  - A feasible design should include two Sensors as child elements to the ‘Motor’-Element



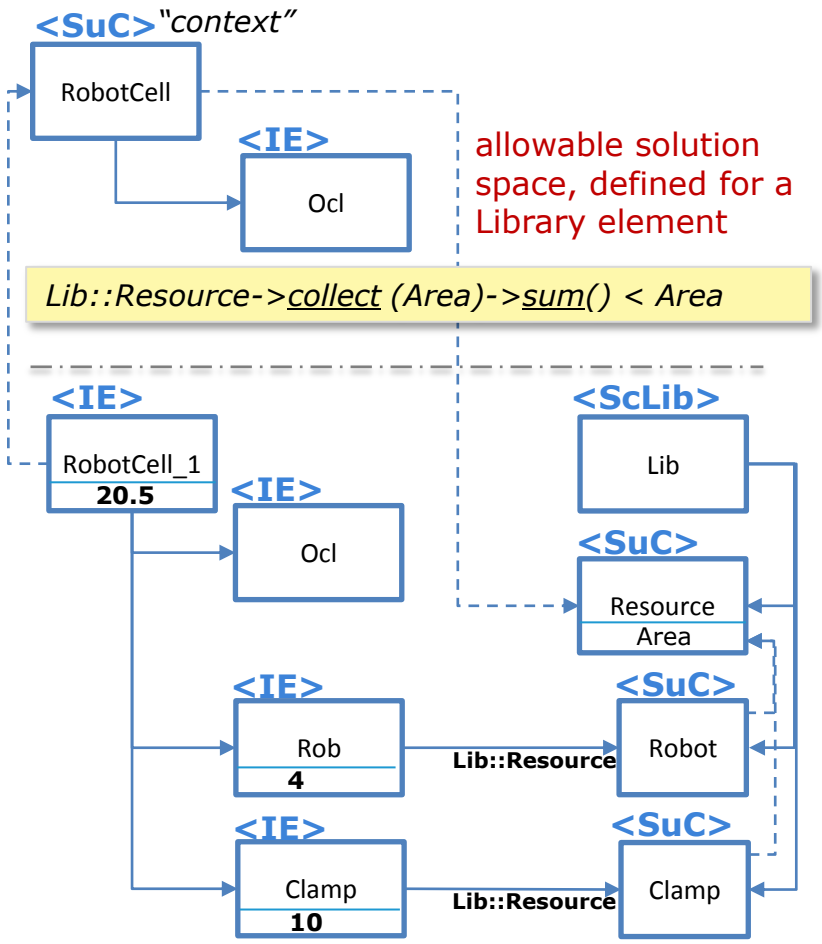
Some Examples

- **Configuration Constraint**
  - '*notEmpty*' requires at least one Instance of the associated class.
  - '*implies*' is a logical predicate.
  - ‚Robot' and 'Controller' define associated classes in the library ‚RLib'.
  - If a design of the 'RobotCell' includes a Robot, it is only feasible when it also includes a Controller.
  - The 'implies' Predicate can be used, to define configuration variants. Variants can be modeled with constraints, there is no need for Model - Templates for each possible configuration.



Some Examples

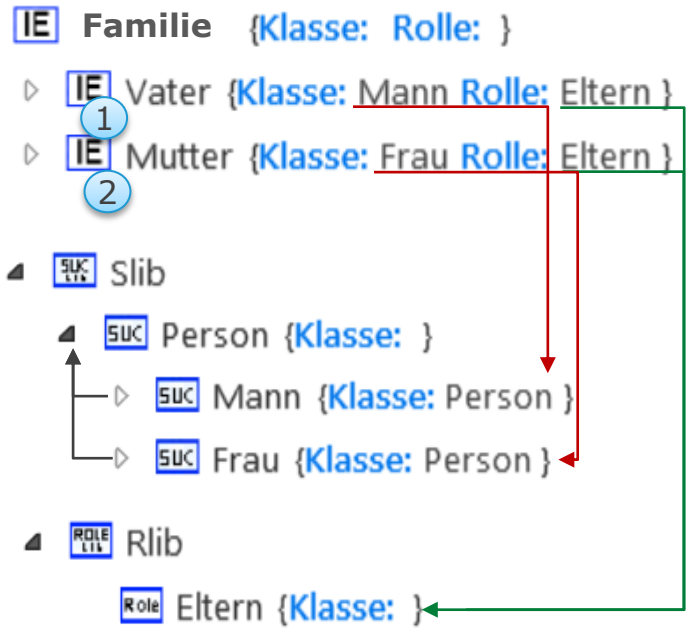
- **Functional Constraint**
  - '*collect*' returns a set of elements, defined by the parameter expression.
  - '*sum*' is a function.
  - ‚Area' is an Attribute of the Class 'Resource' in the library 'lib'. 'RobotCell', 'Robot' and 'Clamp' are derived classes.
  - For all Resources in a 'RobotCell' - design, the sum of the 'Area' of them should not exceed the Area of the Cell itself.
  - OCL-Constraints can be defined on derived classes. They may be also associated with classes.



feasible design

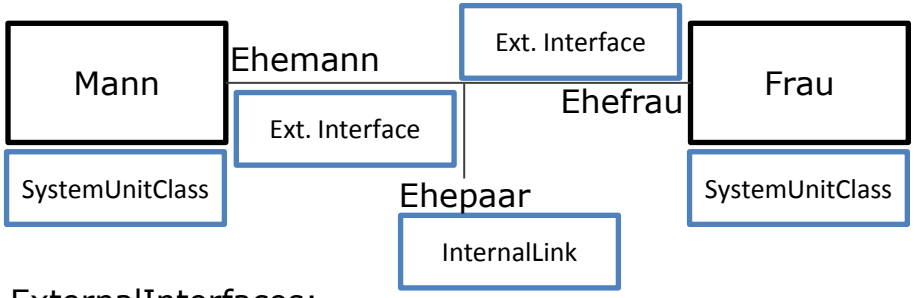
OCL AutomationML - Binding - Classifiers

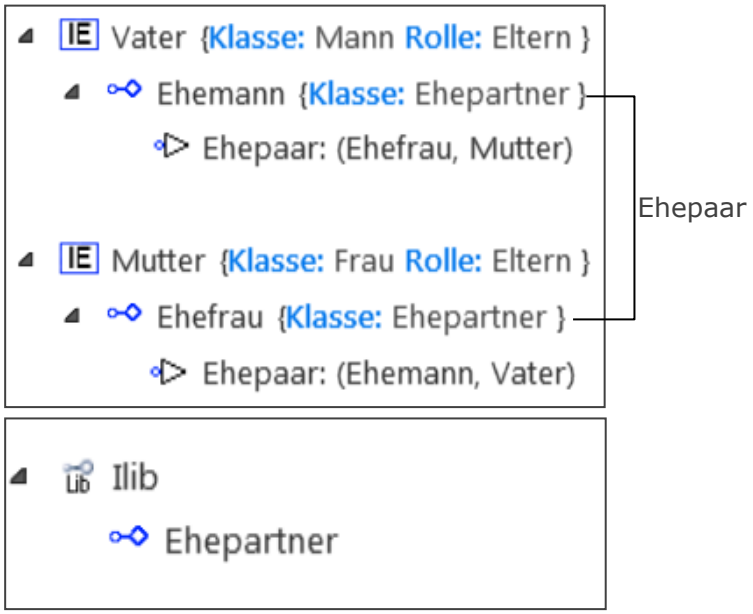
```
context Familie
//Element ‚Vater` is Instance of „Person", „Eltern" and „Mann"
inv c1: Vater.oclIsKindOf(Slib::Person)
inv c2: Vater.oclIsKindOf(Rlib::Eltern)
inv c3: Vater.oclIsKindOf(Slib::Mann)
// ‚Vater' is the first Instance of „Person"
inv c4: Slib::Person->at(1)=Vater
// ‚Mutter' is the second Instance of „Person" but
// the first Instance of „Frau"
inv c5: Slib::Person->at(2)= Mutter
inv c6: Slib::Frau->at(1)= Mutter
// there are 2 Instaces of „Eltern"
inv c7: Rlib::Eltern->size()=2
```



- **oclIsKindOf:** can be used to evaluate the source classifier, inheritance relations are evaluated
- ParentChild - Relations are considered as ordered sets

OCL AutomationML – Binding for Internal Links





ExternalInterfaces:
Association Endnames: "Ehemann, Ehefrau"

InternalLink:
Association Name: "Ehepaar"

```
context Mann
inv : self.Ehefrau
```

: this could be ambiguous

OCL Rule: An association end name may be qualified with its association name or its source classifier name to resolve an ambiguity.

```
context Mann
inv : self.Slib::Frau::Ehefrau
inv : self.Ehepaar::Ehefrau
```

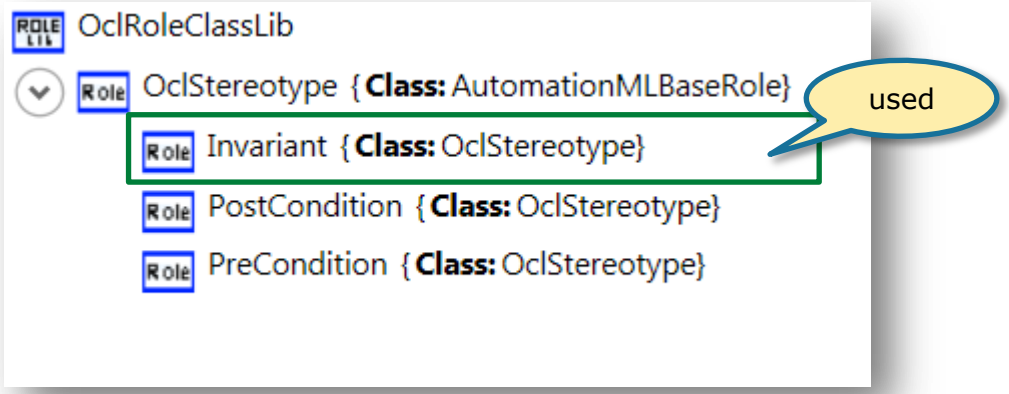
// source classifier name = Slib::Frau

// association name = Ehepaar

Evaluates to a set of all Instances of "Frau" which are related to a "Mann" via the Interface "Ehefrau"

OCL – AutomationML Integration

- Identification of OCL-Constraints in a Design Model via RoleClasses
- OCL-Expression is stored in an Attribute which references the Version of the used OCL-Standard
- An AutomationML Object may have multiple associated OCL-Constraints
- The Use of InternalElements to represent OCL-Constraints allows the construction of OCL-Libraries



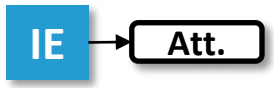

```
<RoleClass
  Name="OclStereotype"
  RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole">
  <Attribute
    Name="oclExpression"
    AttributeDataType="xs:string">
    <Description>an Ocl Expression according to the re
Semantik</Description>
    <RefSemantic
      CorrespondingAttributePath="http://www.omg.org/spec/OCL/2.3.1/" />
  </Attribute>
  <RoleClass
```

supported version

Features of the AutomationML OCL Extension

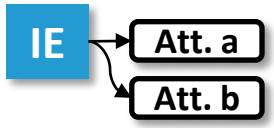
- **Invariants (Integrity of Design Data)**
  - Relations between attributes of different Objects
  - Conditions about mutual dependencies between Objects
  - Conditions about Properties of Objects in Collections

Attribute: (AutomationML)



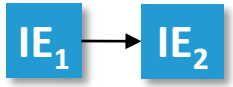
"The value of the Attribute must be a double between 100 and 500"

Tuple: (AutomationML **- OCL**)



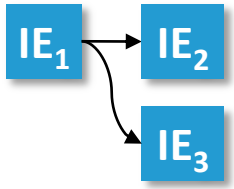
"The value of Attribute a must be lower than the value of b"

Entity: (AutomationML)



"The child IE2 of Entity IE1 must be a …"

Inter-Entity: (AutomationML **- OCL**)

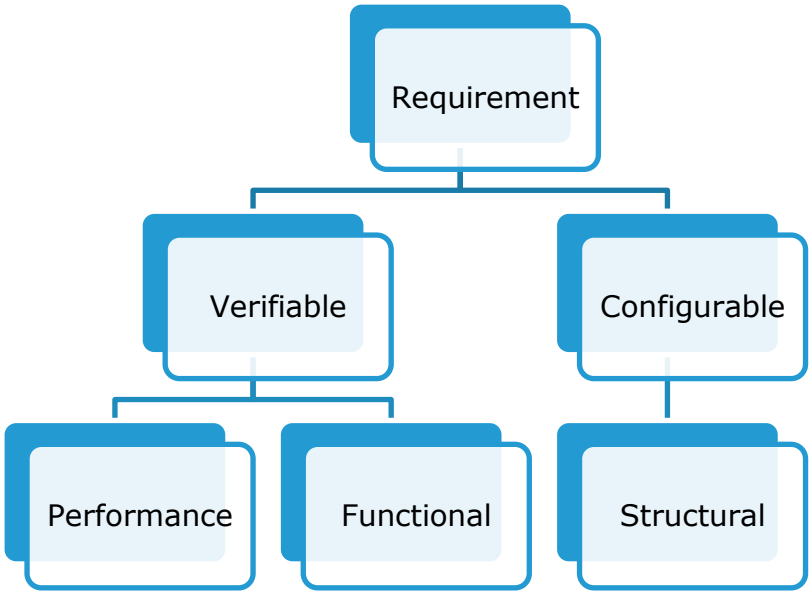


"If the child IE2 of Entity IE1 is a … than it should have a sibling IE3 which is a …"

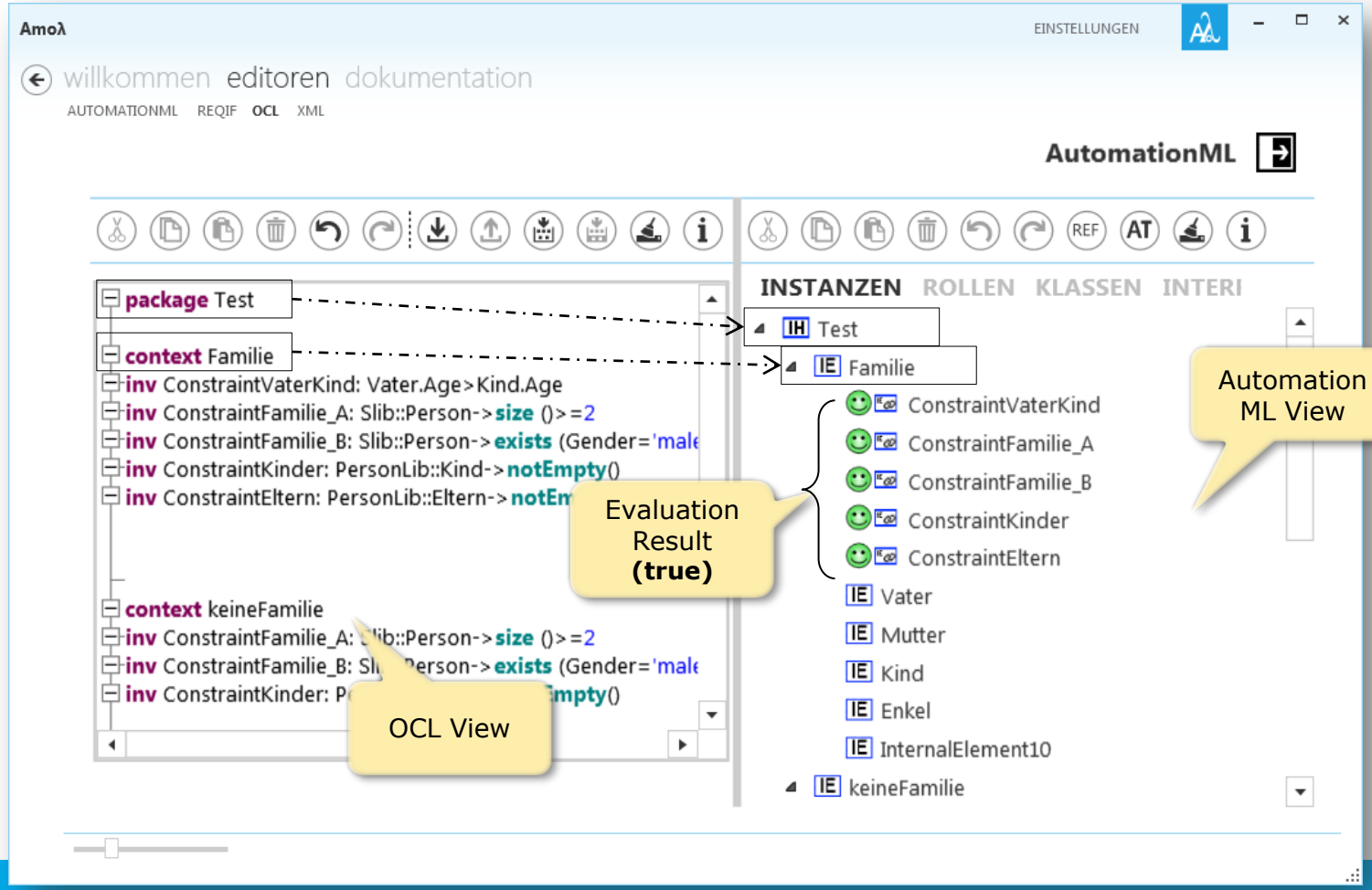
Integrity in AutomationML DesignModels

Context of this Work in the Research Project EfA

- In Model Driven Requirements Engineering it should be possible to link Requirement Models with Design Models.
- Some Requirements, such as performance requirements, like speed, timing, volume or throughput are verifiable requirements. The Verification can only be done on the design model.
- Structural Requirements may result in alternative configuration variants. For automatic construction of those variants, the requirements and the design elements have to be linked.



A Model Integration and Constraint Evaluation Prototype Amoλ

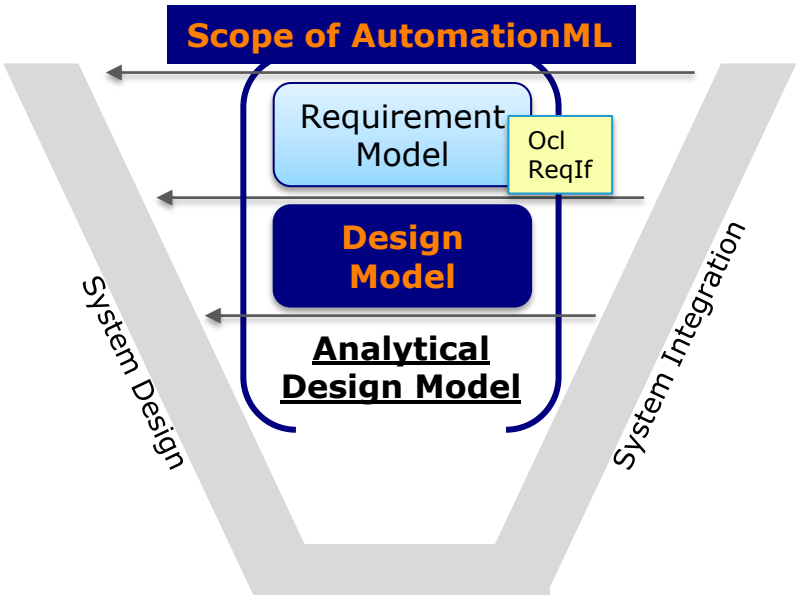


Prototype Features

- **AutomationML OCL Component**
  - OCL Editor
  - OCL Parser
  - OCL - AutomationML Evaluator
  - OCL - AutomationML Association
- **AutomationML ReqIF Component**
  - ReqIf: Requirements Interchange Format (used to structure and for textual description of requirements)
  - ReqIF - AutomationML Association

Advantages

- **The Analytical Design Model**
  - Combination of the Requirement Model with the Design Model
  - Automatic Verification of Design Models
    - Leads to checkable Requirements (Design Guides, Regulations)
    - May Ensure compatibility of Components in the design phase
  - Formal definition of the allowed solution space
    - Avoids infeasible solutions in the design phase
  - Automatic generation of design alternatives

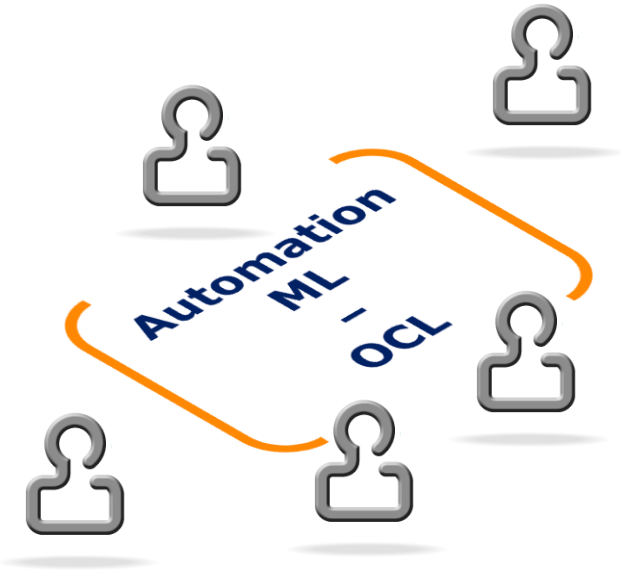


in VDI2206: Design methodology for mechatronic Systems

Further on

- **Cooperative Design**
  - Integrity of distributed Design Models
  - Integrity of the Integrated Design Model

Integrity Checking is possible, even if loosely coupled tools are used in distributed engineering workflows.



Closing Statement

With AutomationML - OCL Association

AutomationML - Models get ready for

Requirements Verification and

Data Integrity Checking of distributed Design Models

Thank you for your kind attention

Please ask questions or look to the software prototype

inpro
Innovationsgesellschaft für fortgeschrittene
Produktionssysteme in der Fahrzeugindustrie mbH

Steinplatz 2

D-10623 Berlin

www.inpro.de

Josef Prinz, T.: 030 39997161 E.: prinz@inpro.de