Consistent Merging of AutomationML Documents in Multiple Sources Scenarios*

Josef Prinz

inpro

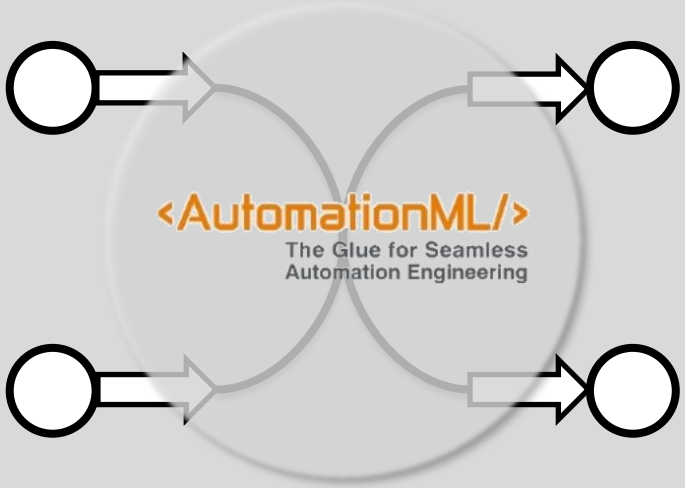
4th AutomationML user conference

18/10/2016

* The presented content is a result of an inpro joint project with Daimler, Volkswagen AG, Siemens, thyssenkrupp and Sabic

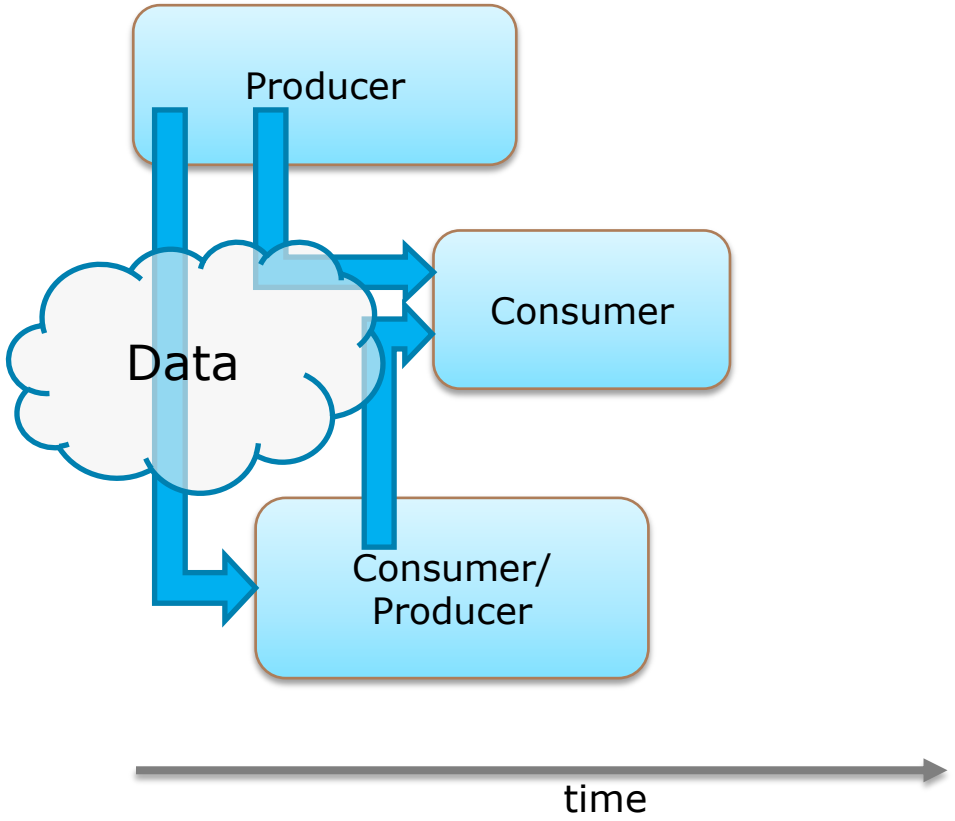
Overview

1. **Motivation and Intentions**
2. Use cases of multiple sources scenarios
3. Usage of AutomationML in these scenarios
4. Merge Workflow
5. Solution proposals
6. A first compare and merge prototype tool



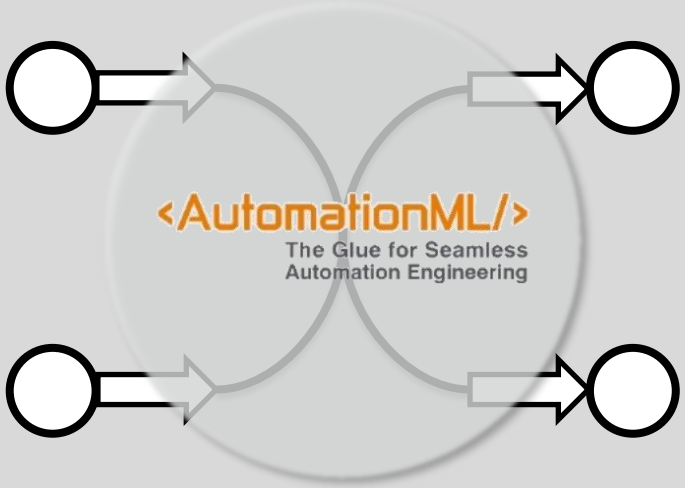
Motivation / Intention

- concurrent design phases with overlapping content are standards in today's engineering processes.
- consistency of shared design artefacts is difficult to achieve for design tools using file-export and file-import to exchange design data.
- AutomationML is a solution for different data exchange scenarios in engineering but could it be used to establish tight couplings of design tools, sharing data?
- What are the workflows (exchange scenarios), that should be supported and how can an "integration tool set" be implemented?



Overview

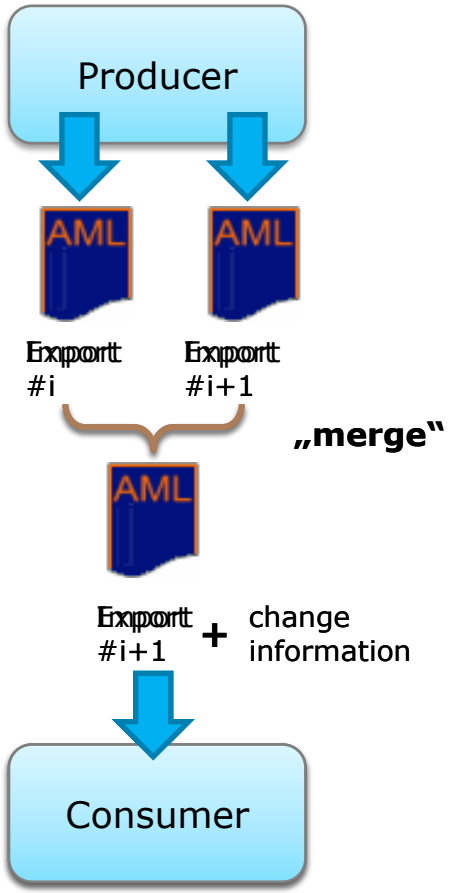
1. Motivation and Intentions
2. **Use cases of multiple sources scenarios**
3. Usage of AutomationML in these scenarios
4. Merge Workflow
5. Solution proposals
6. A first compare and merge prototype tool



Use cases of multiple sources scenarios (Scenario I)

Utilization of the versioning capabilities of AutomationML

- engineering is an iterative design process
- data consumers should be aware of changes made to previously imported data
- data producers are able to inform their customers about changes
- Change information should be related to objects
- AutomationML supports identification of design artefacts on an object oriented level
- AutomationML contains mechanisms to provide change information to consumers
- Integration of change information can be considered as the basic merge scenario

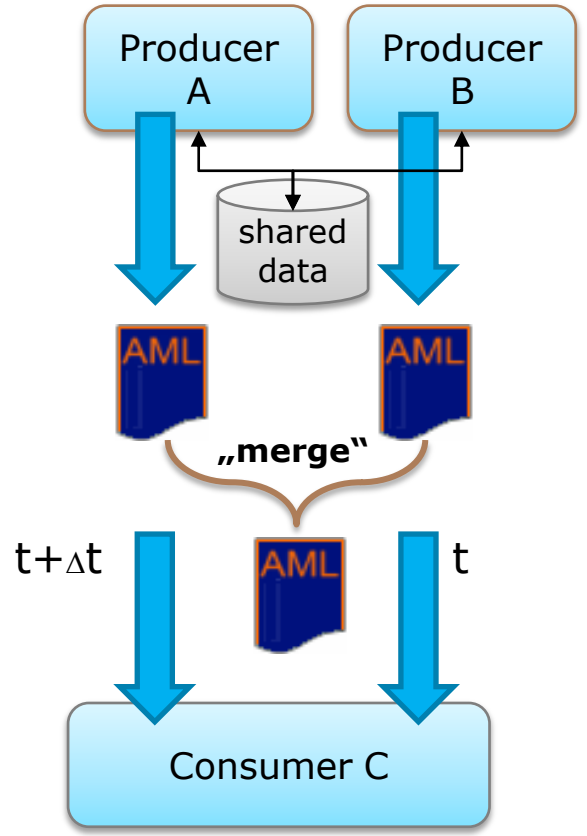


allows skipping of versions

Use cases of multiple sources scenarios (Scenario II)

Dataflow in coupled systems

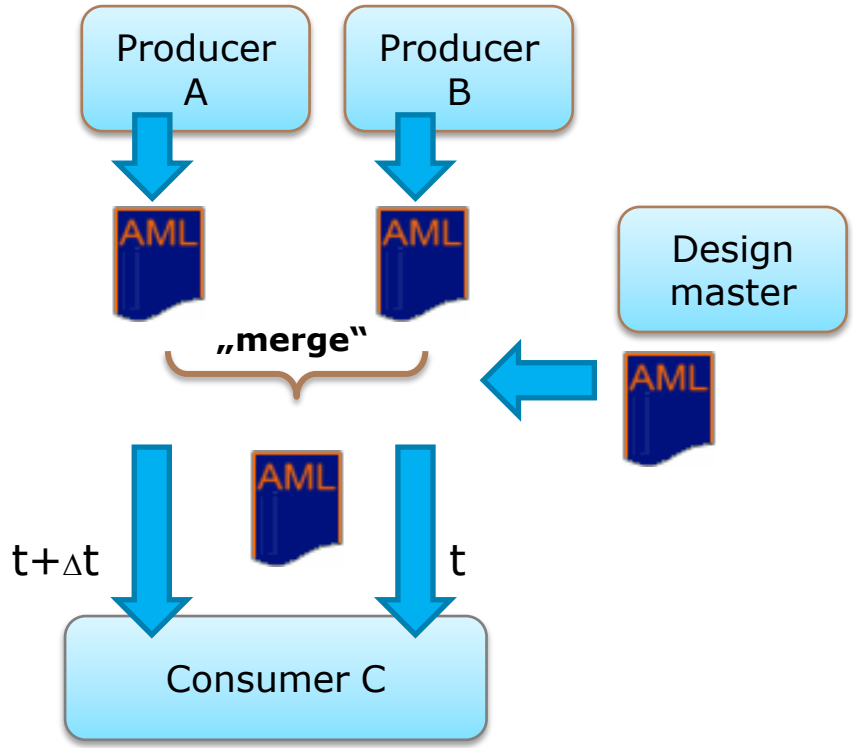
- objects are “globally” identifiable, exporters should be able to generate globally unique identifiers
  - design data is created using shared library objects
  - design data is saved in a shared repository
  - tools are synchronized via a common database
- Consumers should be able to compare identical objects using the AutomationML unique identifier
- Consumers should be aware that conflicts, contradictions and redundancies may exist



Use cases of multiple sources scenarios (Scenario III)

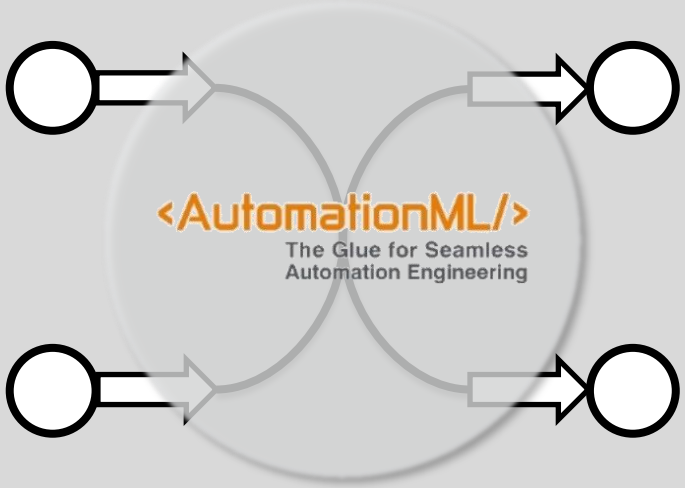
Dataflow in uncoupled systems

- Additional identification strategies are needed, identity of objects may be based on naming conventions or specific attribute values
- design aspects may overlap and could be contradictory
- conflict resolving strategies are needed
- AutomationML offers mechanisms to define a “design guide” via common RoleClasses, SystemUnitClasses or even common topologies (InstanceHierarchies with predefined parent child relations).
- The design guide can be used as the design master in conflict resolving. The design master can exists as a separate AutomationML document.



Overview

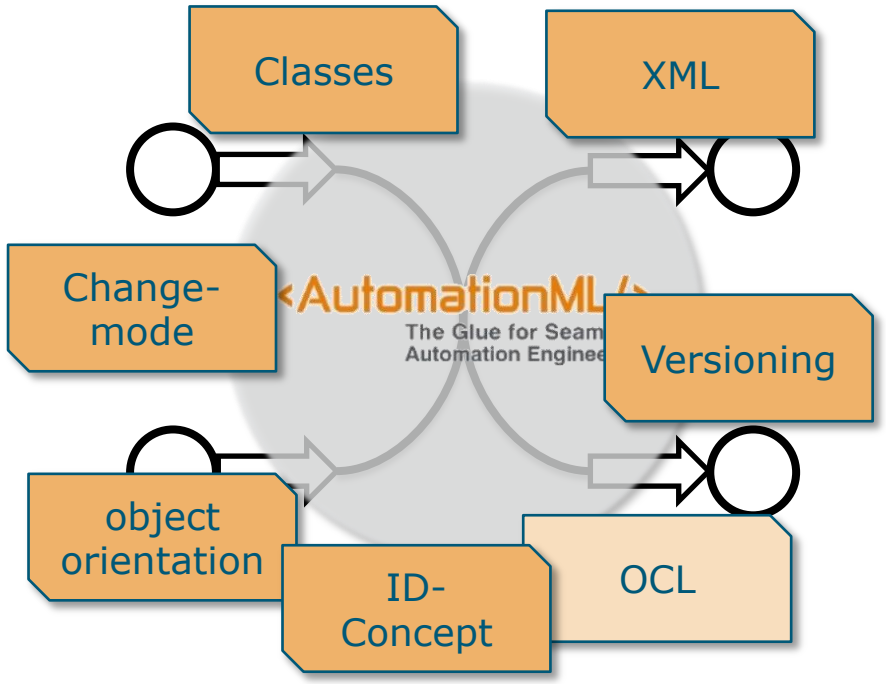
1. Motivation and Intentions
2. Use cases of multiple sources scenarios
3. **Usage of AutomationML in these scenarios**
4. Merge Workflow
5. Solution proposals
6. A first compare and merge prototype tool



Usage of AutomationML for merge scenarios

Key factors of AutomationML supporting merge scenarios

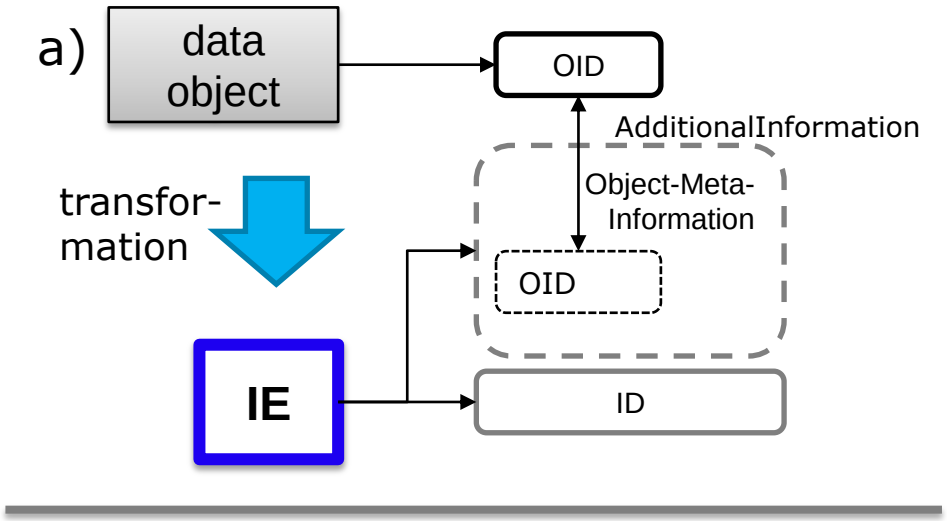
- object oriented modelling
- globally unique object identification
- "typing" of design artefacts using roles and classes
- versioning of objects
- change mode attribute
- XML language
- extensions for constraint modelling using OCL



Usage of AutomationML for merge scenarios

ID – Generation

- Relations between system internal object IDs (OIDs) and AutomationML IDs should be preserved
  - The ID-Relations can be saved in the AutomationML Document using meta information stored in an "AdditionalInformation" – Element
  - ID-Relations can be saved in extra documents (ID-mapping-Tables)



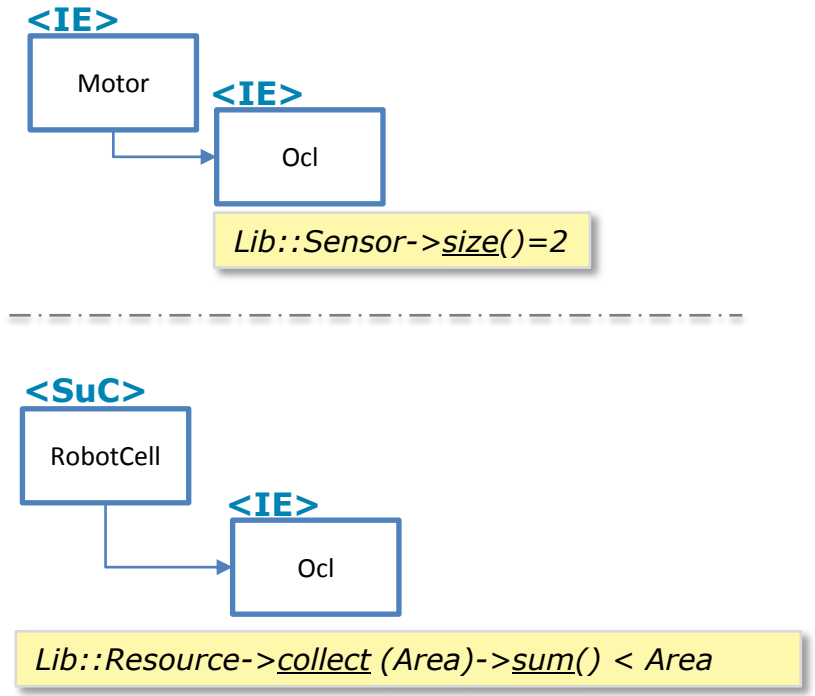
b)

```
<AutomatioMLIDMappingTable>
<IDMappings>
 <AutomationMLIDMap
   AutomationMLID="{435632f3-f7c7-4328-b394-9a746dd17321}"
   DataObjectID="M2">
</AutomationMLIDMap>…..
```

Usage of AutomationML for merge scenarios

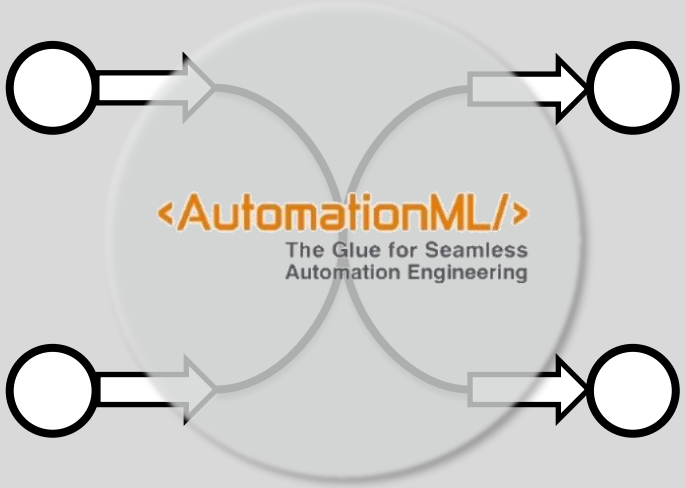
extensions for constraint modelling using OCL

- OCL = Object Constraint Language
- Constraints are assigned to AutomationML classes and objects via InternalElements – representing ocl-expressions.
- A specific AutomationML <-> OCL binding is used, to associate AutomationML/ CAEX elements with variables and functions
- see also "Formalization of object constraints in AutomationML", presentation from the 3d AutomationML user conference

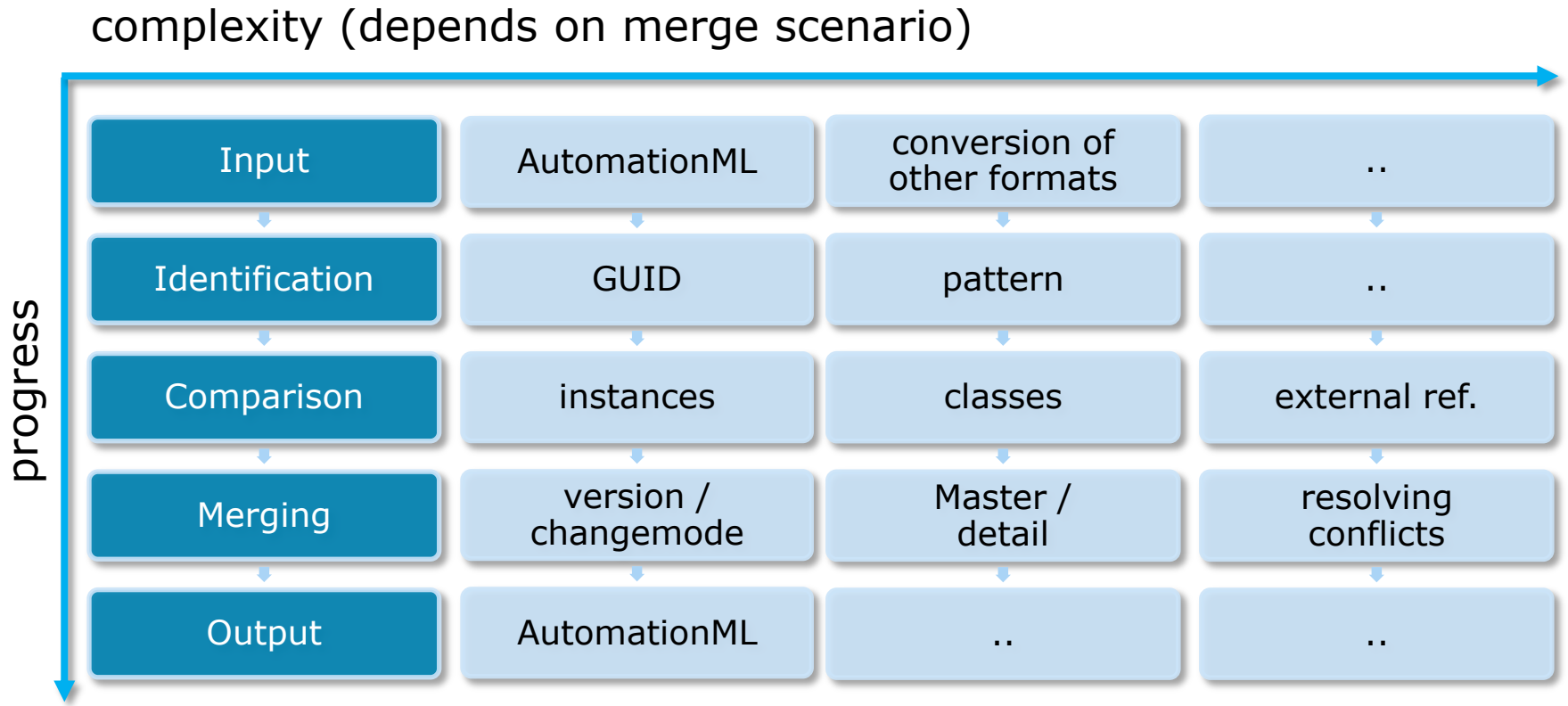


Overview

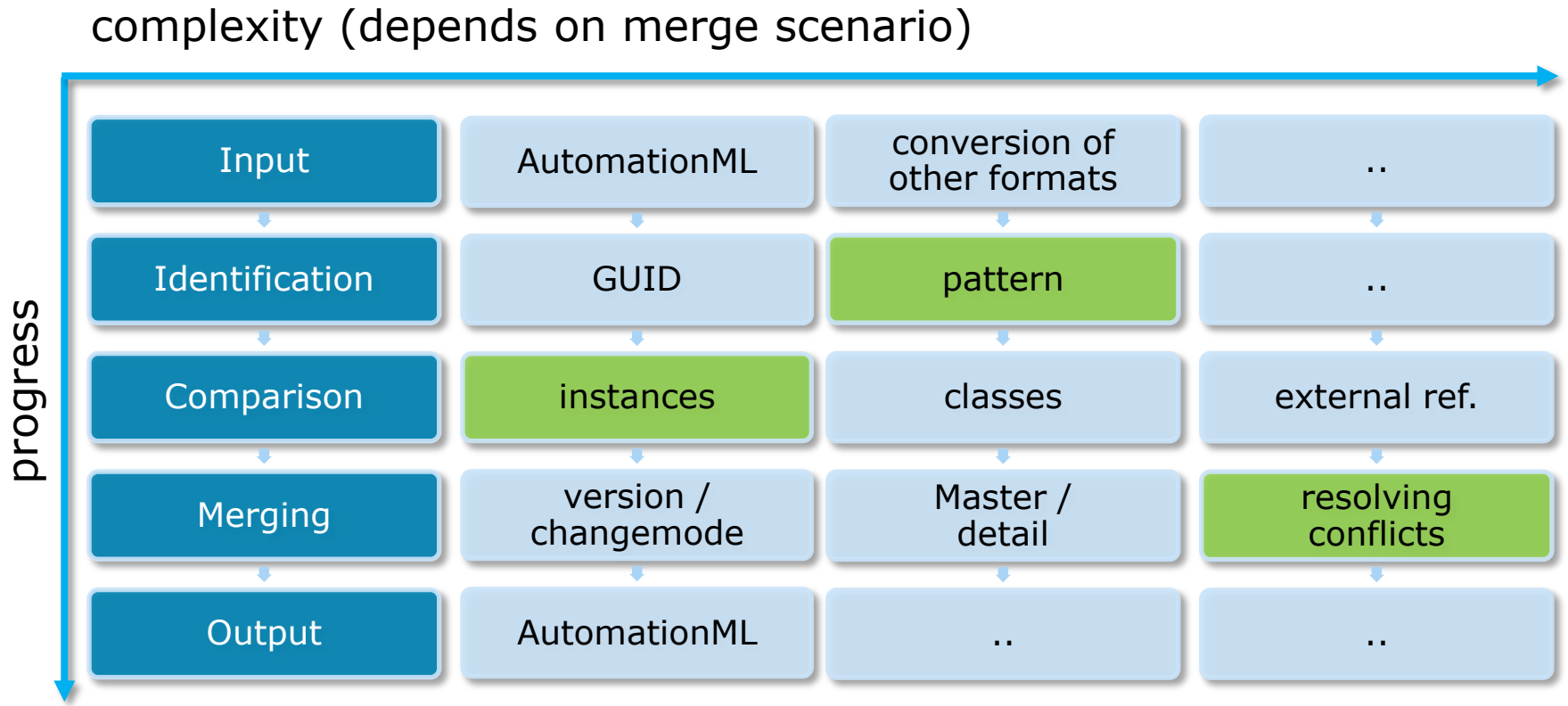
1. Motivation and Intentions
2. Use cases of multiple sources scenarios
3. Usage of AutomationML in these scenarios
4. **Merge Workflow**
5. Solution proposals
6. A first compare and merge prototype tool



Merge Workflow

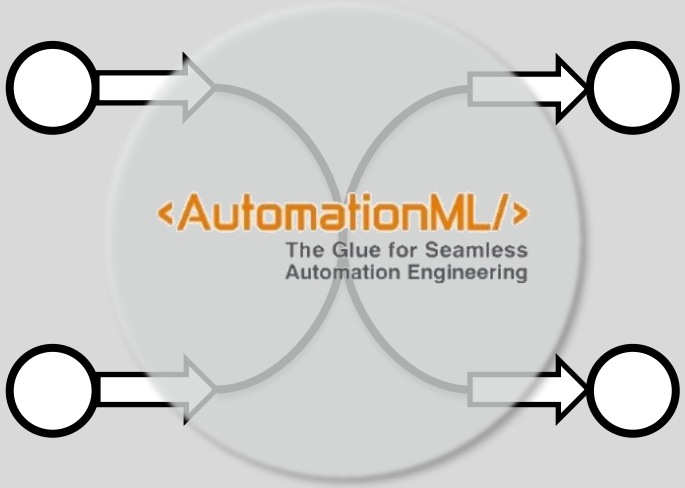


Merge Workflow



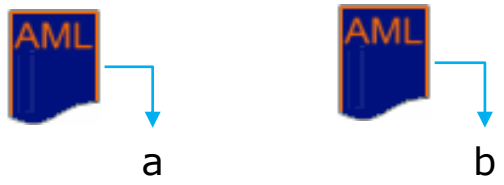
Overview

1. Motivation and Intentions
2. Use cases of multiple sources scenarios
3. Usage of AutomationML in these scenarios
4. Merge Workflow
5. **Solution proposals**
6. A first compare and merge prototype tool



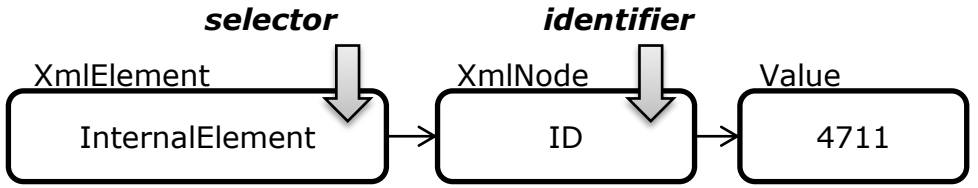
Identification

- using XML tools (XPath) for identification
- identification of equivalent objects
- generic identification concept
  - Standard AutomationML object identification
  - identification using object attributes
  - identification using object relations
- each producer defines its own identification rules
- an identification rule contains two parts
  - Value identifier
  - Object selection



Objekt a ~ Objekt b ⇔ $ident_a$ ~ $ident_a$

ident = {selector, identifier}



Identifier = //InternalElement/@ID
Selector = ./..

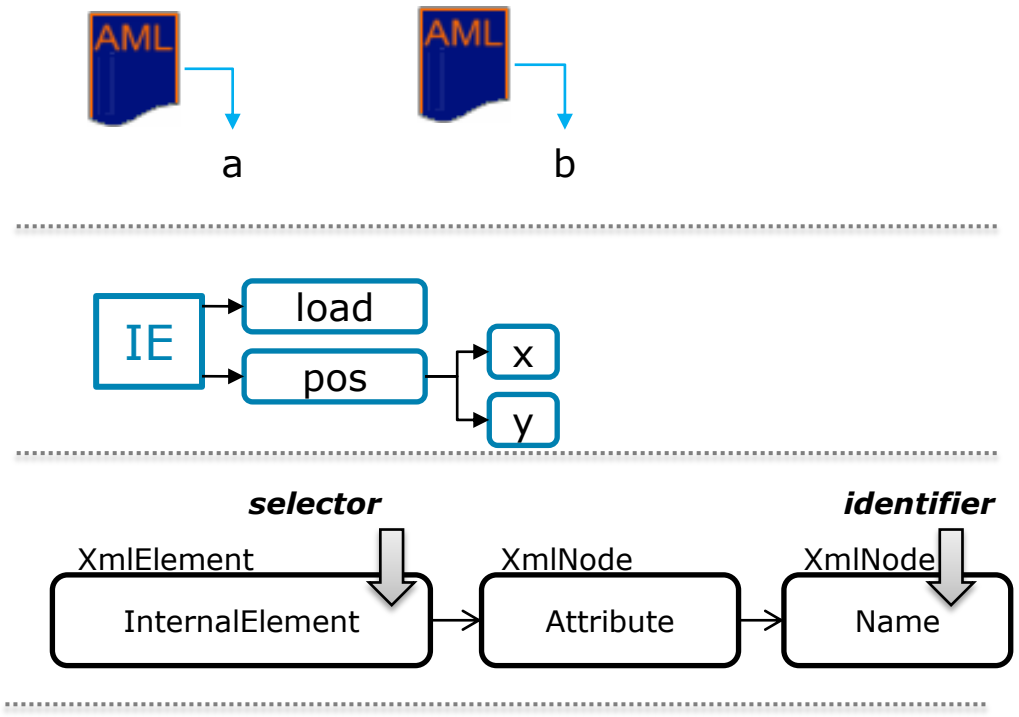
Identification

Identification using Attributes of Elements

a) comparing single attribute values

b) comparing value tuples

c) user defined expressions



$ident_a = ident_a$

a) $ident_a$=$ident_b$=//InternalElement/Attribute[@Name=‘load‘]/Value
b) $ident_a$=$ident_b$=//InternalElement/Attribute/[@Name=‘pos']/
Attribute[@Name=‘x' or @Name=‘y']/Value.

$ident_a \sim_{expr} ident_a$

c) $ident_a$=$ident_b$=//InternalElement/Attribute[@Name=‘load‘]/Value
=> |ident(a)-ident(b)| **<** 10 [a/Unit]

selector = ./ancestor::InternalElement[1]

Identification

Relations used to identify equal objects

• **ID – Equivalence**	**equality of IDs**
• **CAEX-Path – Equivalence**	**equality of CAEX-Paths**

} AutomationML Standard

• Name – Equivalence	equality of names or substrings of names
• Attributevalue – Equivalence	equality or „inequalities" of attribute values
• Hierarchy – Equivalence	equality of parent – child relations
• Semantic – Equivalence	equality of class – instance relations
• Relation – Equivalence	equality of instance – instance relations

Comparison

- objects, which are identified to be equivalent, are compared
- identified objects, which are identified but don’t have an equivalent partner are marked as singletons
- the result of the comparison determines the value of the change mode attribute
  - an equivalent object which is different compared to its partner is marked as changed in the document with the most recent modification date
  - an object is marked as “created”, if it is a singleton and only contained in the document with the most recent modification date
  - an object is marked as “deleted”, if it is a singleton and only contained in the older document

Comparison

- to compare objects, difference functions are used, associated with AutomationML/ CAEX-Elements
  - InternalElement – difference
  - Attribute – difference
  - SystemUnitClass – difference
  - …
- the complete element-trees of each identified object in an equivalence relation are compared (until another identified object is found in a tree)
- the difference functions are classified as “definition-difference”, “set-difference” or “value-difference”

Comparison

Definition Difference

- Name-Difference
  - the Name-Attributes of equivalent objects are different
- ID-Difference
  - the ID-Attribute of equivalent objects are different
- Attribute-Definition-Difference:
  - the AttributeDataType of equivalent attributes are different or
  - the RefSemantic of equivalent attributes are different
  - the Unit of equivalent attributes are different

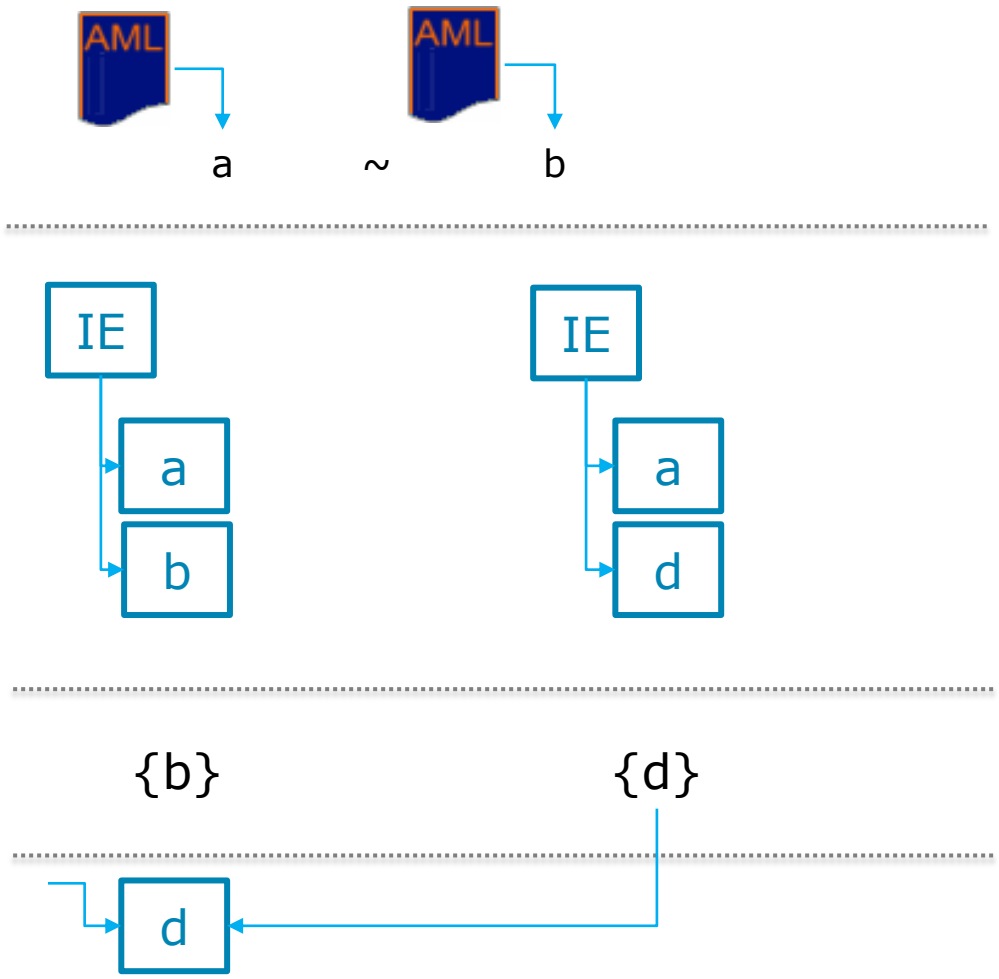
Value-Difference

- Attribute-Value-Difference
  - the Value of equivalent attributes are different (Unit-transformations should be considered)

Comparison

Set Difference

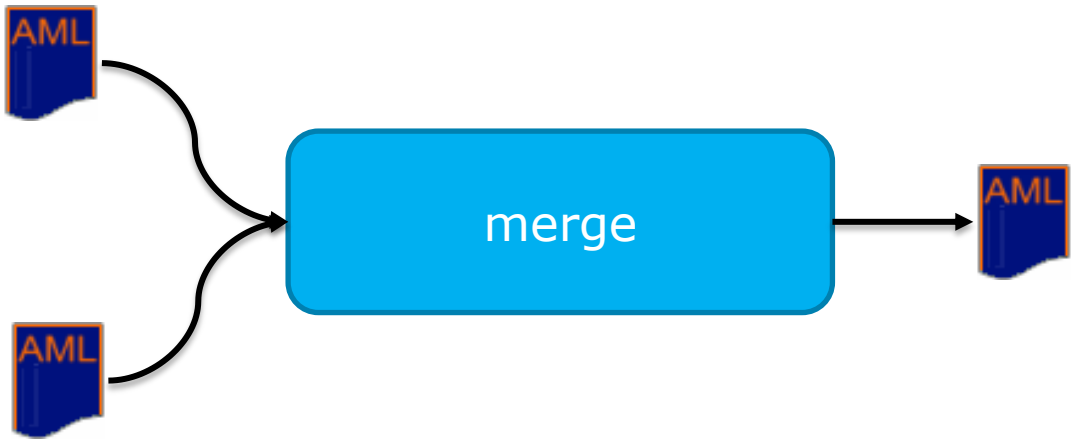
- comparison of two sets of elements (set_a, set_b), where the elements are identified to be equal (sets of InternalElements, sets of Attributes, sets of InternalLinks, sets of SupportedRoles, …)
- The result contains two sets: elements which are in set_a and not in set_b and elements which are in set_b but not in set_a
- <u>Notice:</u> the result of the set difference cannot be used, to set the change mode attribute to either “delete” or “create” (if the element in the result sets is not a singleton, it has been moved to another parent).



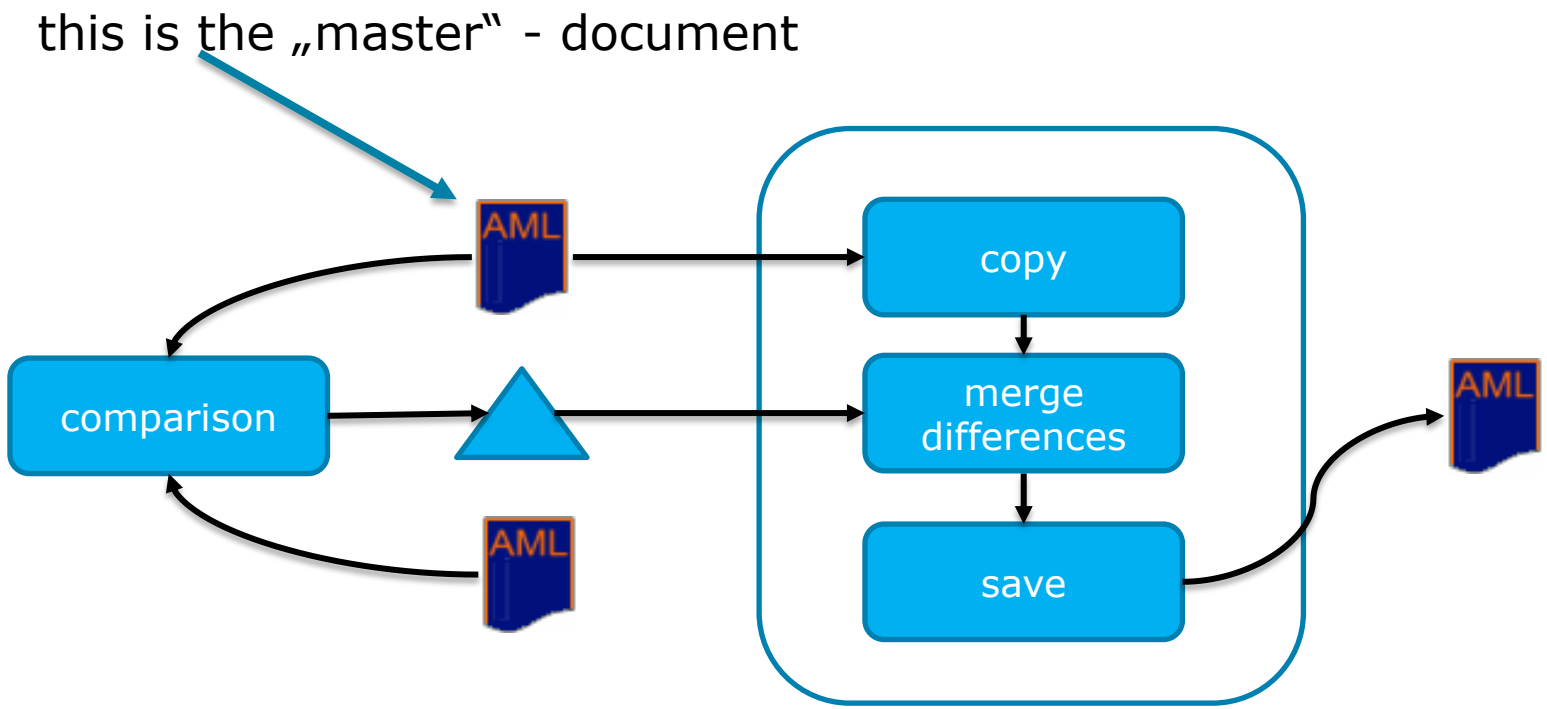
Comparison: Scope of Difference Functions

Difference-Function	Scope						
AttributeValue	Attribute						
AttributeDefinition	Attribute						
AttributeSet	InternalElement	SystemUnitClass	RoleClass	InterfaceClass	External Interface	RoleRequire-ment	Attribute
InterfaceSet	InternalElement	SystemUnitClass	RoleClass	RoleRequirement			
InterfaceDefinition	ExternalInterface						
InternalLinkSet	InternalElement						
InternalElementSet	Internal Element	SystemUnitClass					
InternalElementDefinition	InternalElement						
RoleReferenceSet	InternalElement	SystemUnitClass					
InterfaceNameMappingSet	InternalElement	SupportedRoleClass					
AttributeNameMappingSet	InternalElement	SupportedRoleClass					
SystemUnitClassSet	SystemUnitClass						
SystemUnitClassDefinition	SystemUnitClass						
RoleClassSet	RoleClass						
RoleClassDefinition	RoleClass						
InterfaceClassSet	InterfaceClass						
InterfaceClassDefinition	InterfaceClass						

Merging – the procedure



Merging – the procedure



Merging – Application of rules

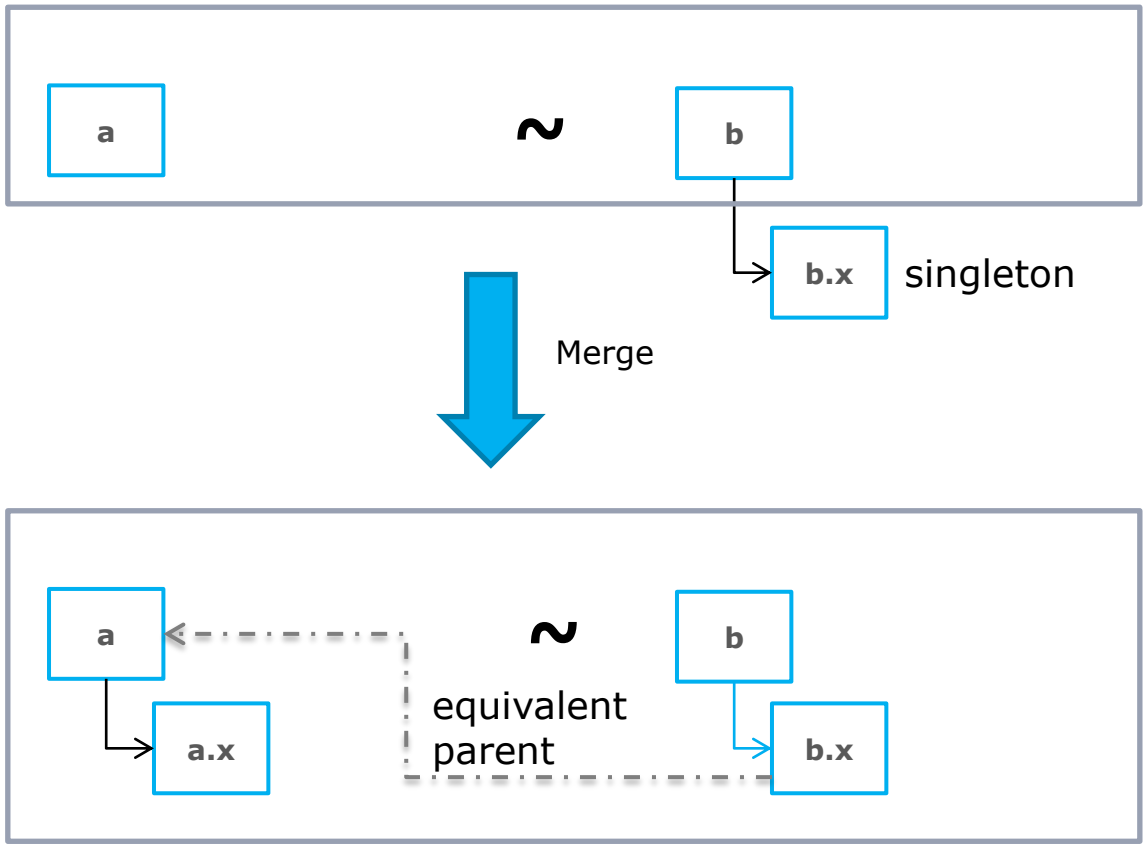
The application of a merge - rule is a supervised procedure

- the supervisor is the user
- a defined constraint (precondition) is checked

The supervision is necessary to resolve conflicts

- If no conflicts exist, no supervision is needed
- a completely automatic merge procedure can be defined

Merging – Examples

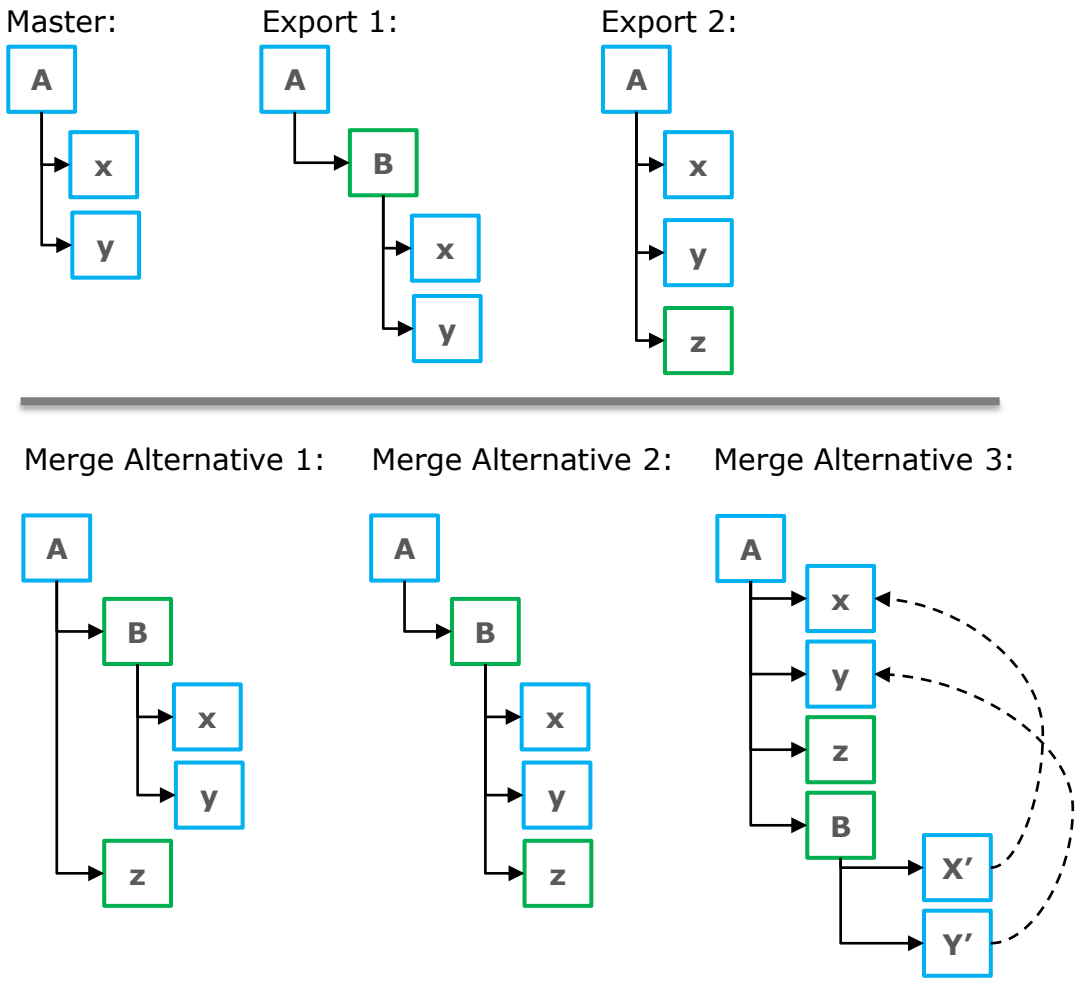


rule:
singletons are added to their equivalent parents

Merging - conflicts

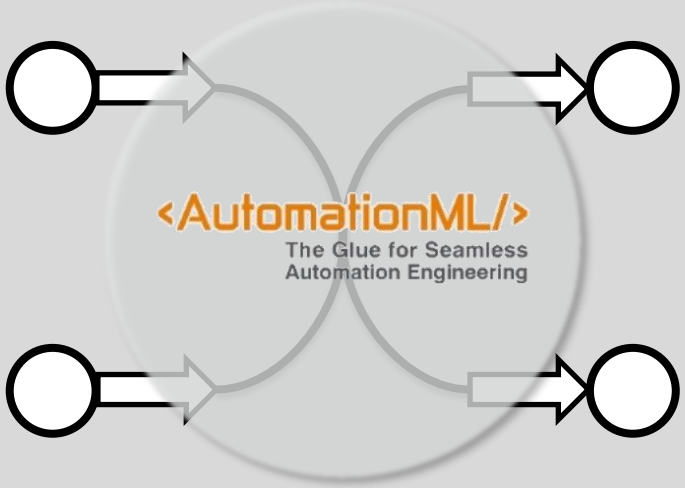
restructure add conflict

- A new hierarchy level ‚B' is added in one document (restructure)
- A new element ‚z' is added to level ‚A' in another document
- **What should be the parent of ‚z' after merging ?**
- A conflict solving rule is necessary:
  - ‚A' should contain elements like ‚z' => Alternative 1 is the correct solution (independent merge)
  - Siblings of ‚z' are elements like ‚x' and ‚y' => Alternative 2 is the correct solution (dependent merge)
  - 'A' should contain elements like 'x', 'y', and 'z' => Alternative 3 is the correct solution, applying the group concept (stable topology)



Overview

1. Motivation and Intentions
2. Use cases of multiple sources scenarios
3. Usage of AutomationML in these scenarios
4. Merge Workflow
5. Solution proposals
6. **A first compare and merge prototype tool**



Toolsupport

provision of a development kit to be used in importers and exporters

- identification tool set
- differencing tool set
- merging tool set

external tools to support specific scenarios

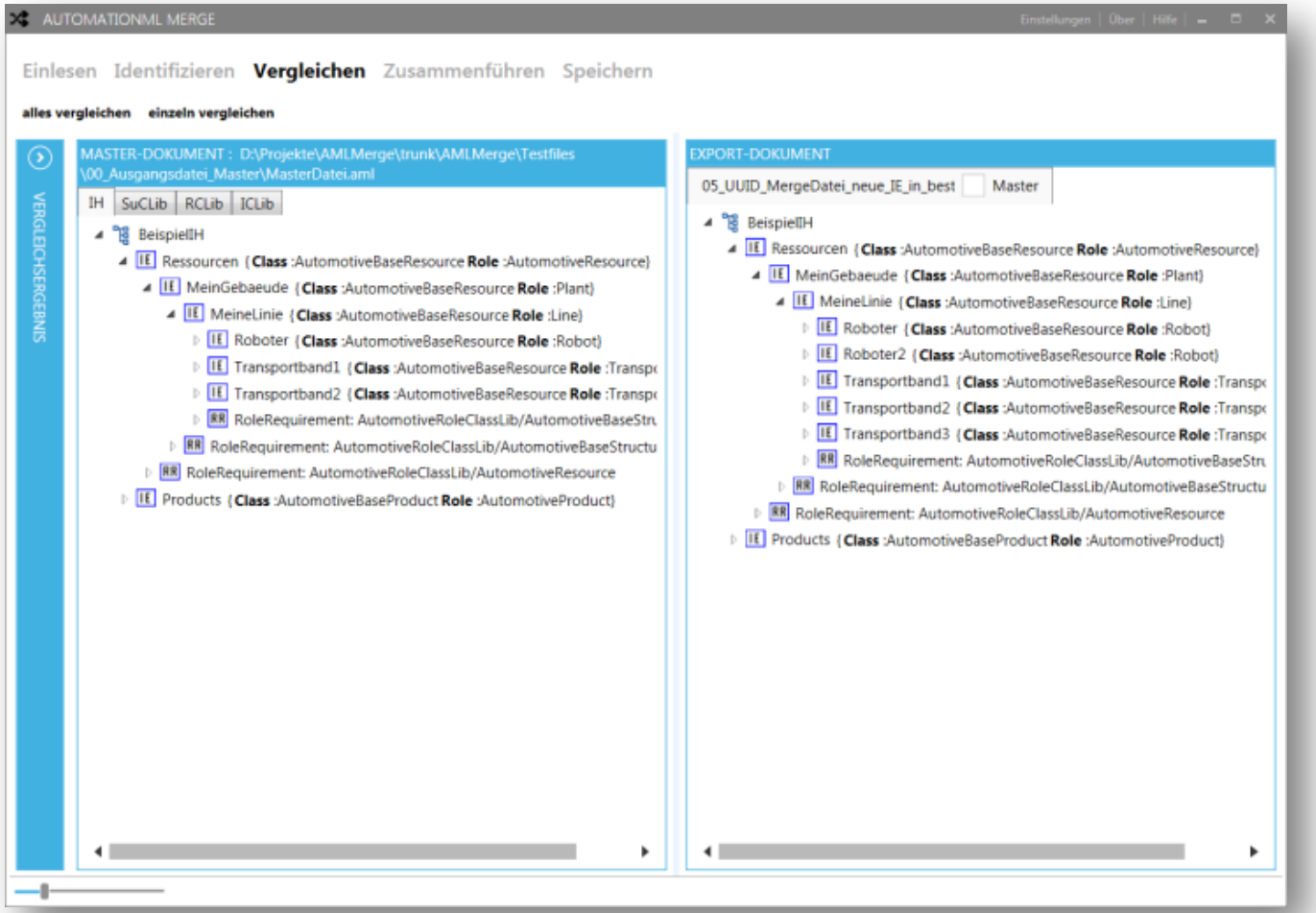
- Versioning tool (adds version – and change mode information)
- Delta extraction (creates an AutomationML document describing the difference)

AutomationML Editor plugins

- Comparer
- Constraint Editor

inpro AMLMerge

- supports the complete workflow
- illustration of different complexity levels
- validation tool for constraint evaluation and constraint solving
- testing of the configurability of the merge workflow and supported methods
- plans for future development and dissemination are currently discussed



Thank you for your kind attention!

Any questions?

inpro
Innovationsgesellschaft für fortgeschrittene
Produktionssysteme in der Fahrzeugindustrie mbH

Steinplatz 2

D-10623 Berlin

www.inpro.de

Josef Prinz T.: +49 30 399 97-161 E.: josef.prinz@inpro.de