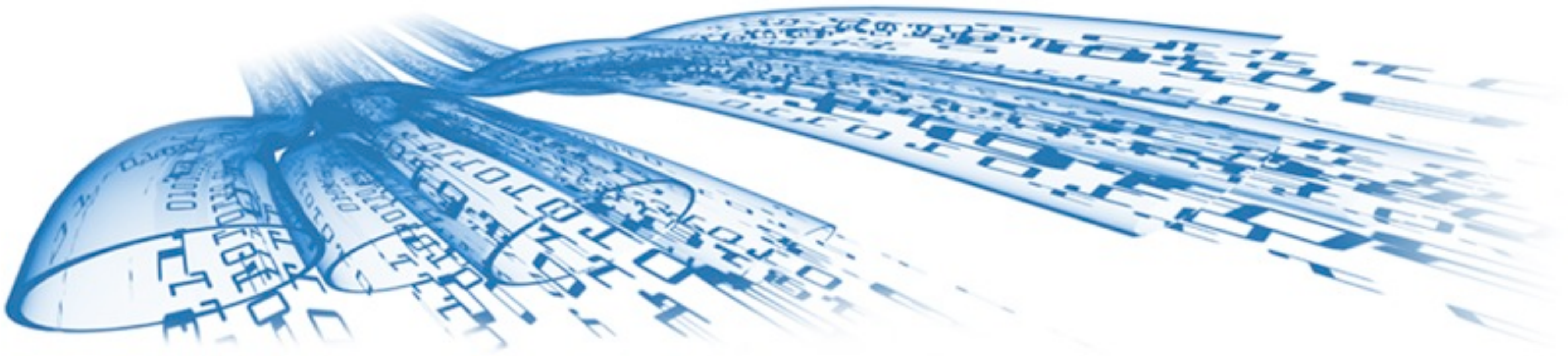


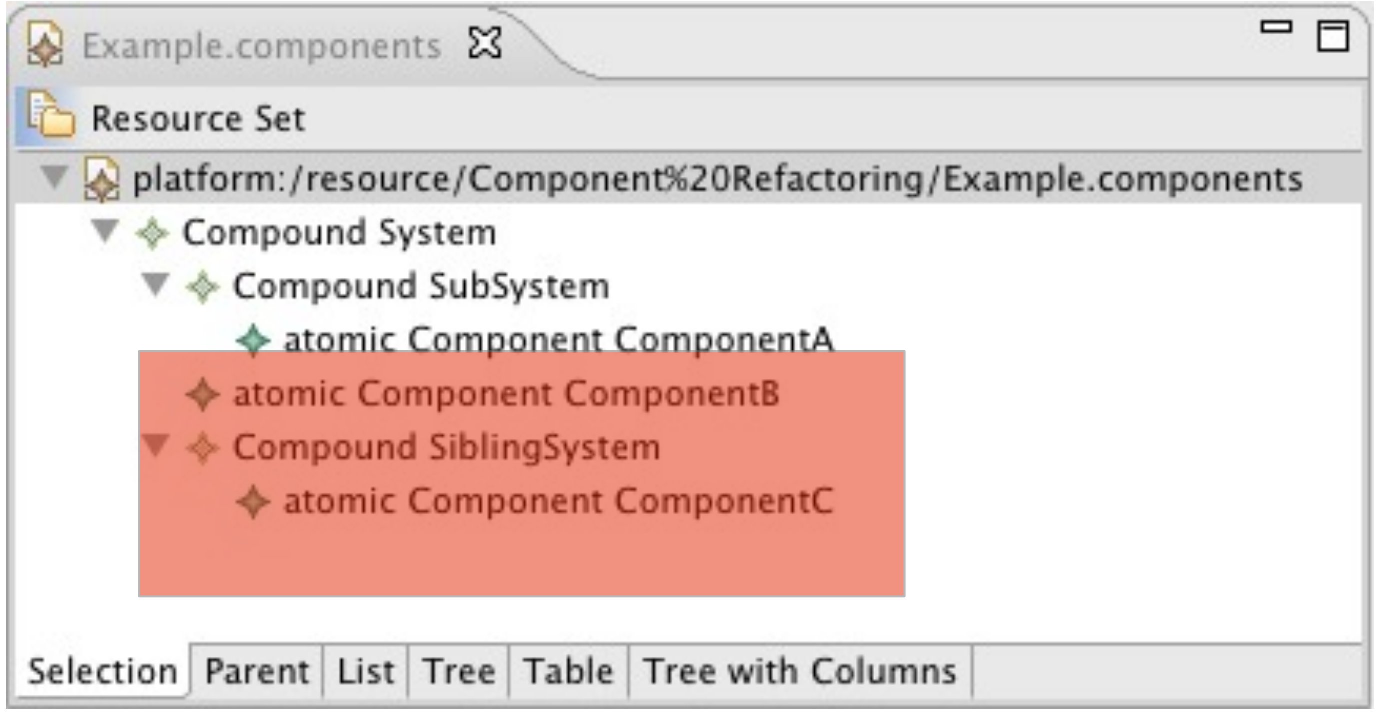
PETE: Prolog EMF Transformation Engine

How to describe model-to-model transformations in a logical fashion

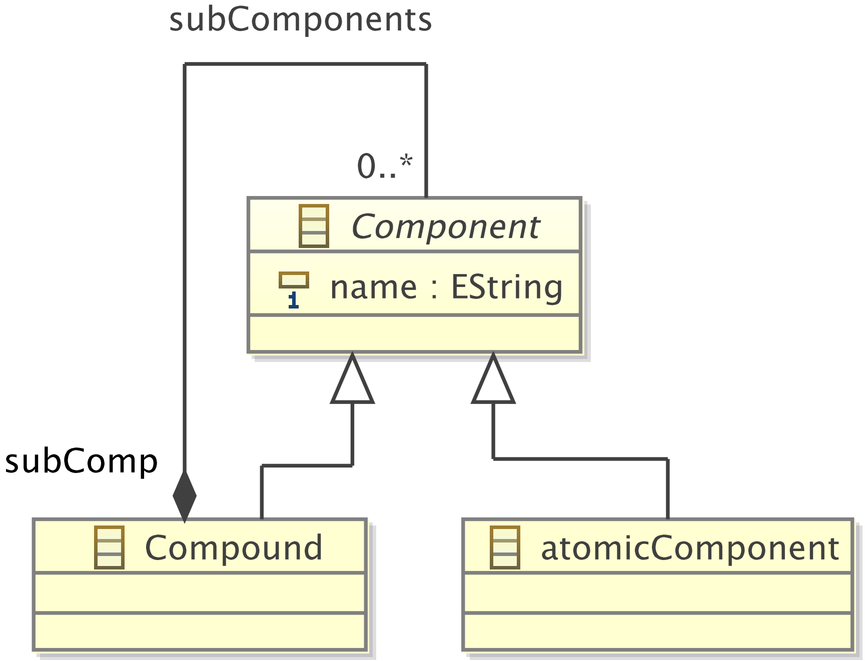
Bernhard Schätz
fortiss gGmbH

23/11/2010, Eclipse Demo Camp

Example: Refactoring



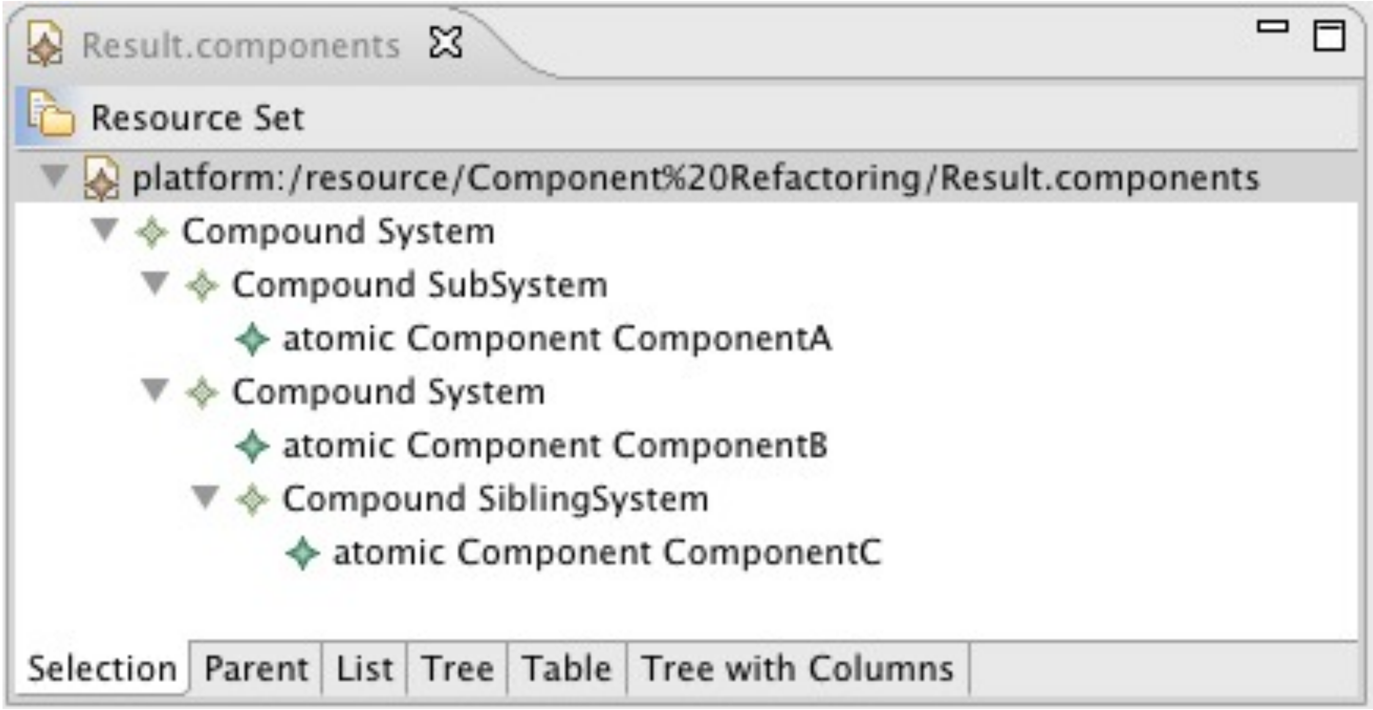
Resource Set
- platform:/resource/Component%20Refactoring/Example.components
  - Compound System
    - Compound SubSystem
      - atomic Component ComponentA
    - atomic Component ComponentB
    - Compound SiblingSystem
      - atomic Component ComponentC



Refactoring: Restructuring of hierarchy

- Components combined to group
- Group clustered in one component

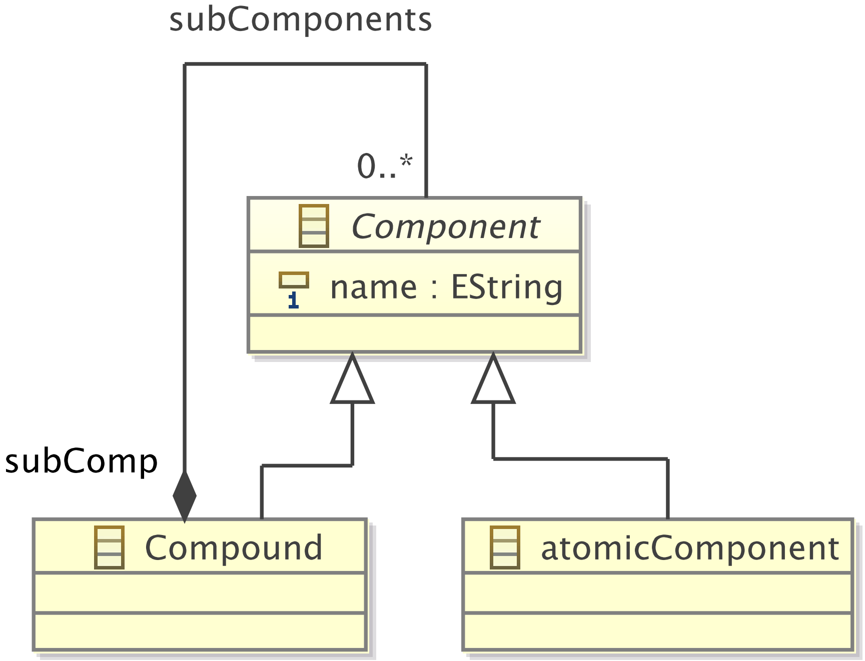
Example: Refactoring



Result.components

Resource Set

```
▼ platform:/resource/Component%20Refactoring/Result.components
  ▼ Compound System
    ▼ Compound SubSystem
        atomic Component ComponentA
    ▼ Compound System
        atomic Component ComponentB
      ▼ Compound SiblingSystem
          atomic Component ComponentC
```



Refactoring: Restructuring of hierarchy

- Components combined to group
- Group clustered in one component

Basics: Set (De)Construction

union(?LeftSet,?RightSet,?UnionSet)

[CompA,CompB] ∪ [SubSystem] = [CompA,SubSystem,CompB]

Set (De)Construction:

- Interpretation: Set UnionSet is the union of LeftSet and RightSet
- Construction: union(+LeftSet,+RightSet,-UnionSet)
- Deconstruction: union(-LeftSet,+RightSet,+UnionSet) and union(+LeftSet,-RightSet,+Unionset) as well as union(-LeftSet,-RightSet,+UnionSet)

Basics: Set (De)Construction

union(?LeftSet,?RightSet,?UnionSet)

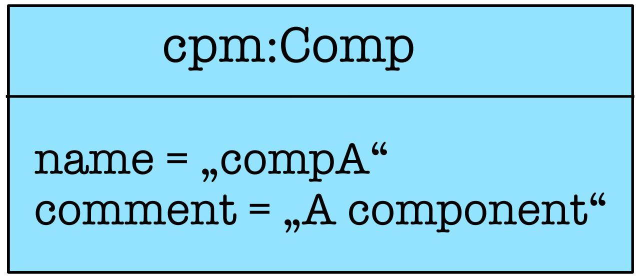
[CompA,CompB] = [SubSystem] / [CompA,SubSystem,CompB]

Set (De)Construction:

- Interpretation: Set UnionSet is the union of LeftSet and RightSet
- Construction: union(+LeftSet,+RightSet,-UnionSet)
- Deconstruction: union(-LeftSet,+RightSet,+UnionSet) and union(+LeftSet,-RightSet,+Unionset) as well as union(-LeftSet,-RightSet,+UnionSet)

Basics: Element (De)Construction

Comp(?Element,?Entity,?Name,?Comment)



cpm:Comp

name = „compA“
comment = „A component“

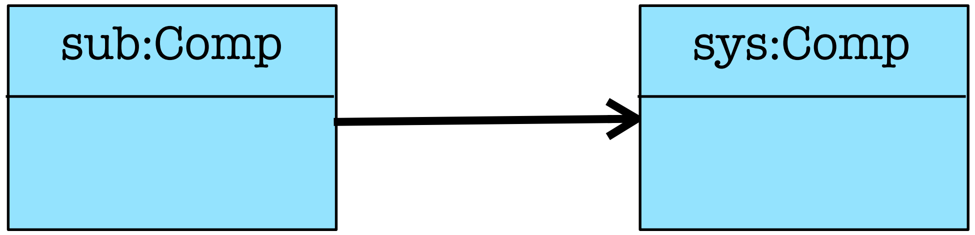
cmp „compA“ „A component“

Element (De)Construction:

- Interpretation: Object Element has reference Entity and attributes Attribut1,…AttributeN
- Deconstruction: class(+Element,-Entity,-Attribute1,…,-AttributeN)
- Update: class(-Element,+Entity,+Attribute1,…,+AttributeN)
- Construction: class(-Element,-Entity,+Attribute1,…,+AttributeN)

Basics: Relation (De)Construction

subComp(?Relation,?Entity1,?Entity2)

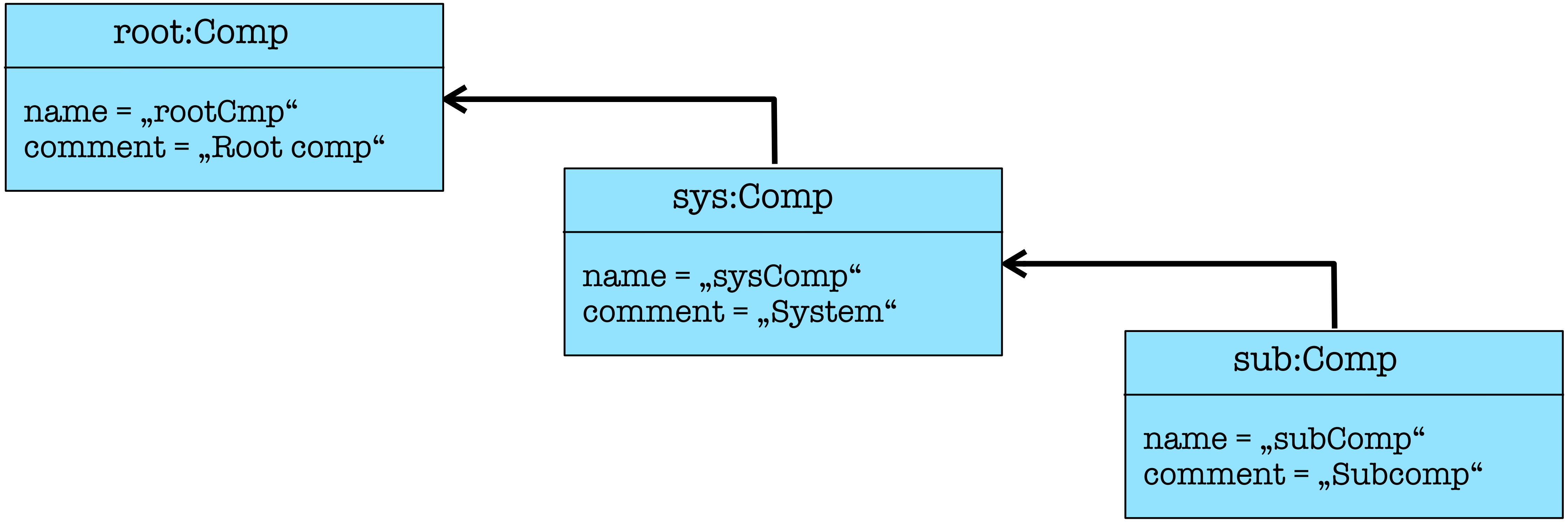


sub sys

Relation (De)Construction:

- Interpretation: Relation Relation links object Entity1 and object Entity2
- Deconstruction: association(+Relation,-Entity1,-Entity2)
- Construction: association(-Element,+Entity1,+ Entity2)

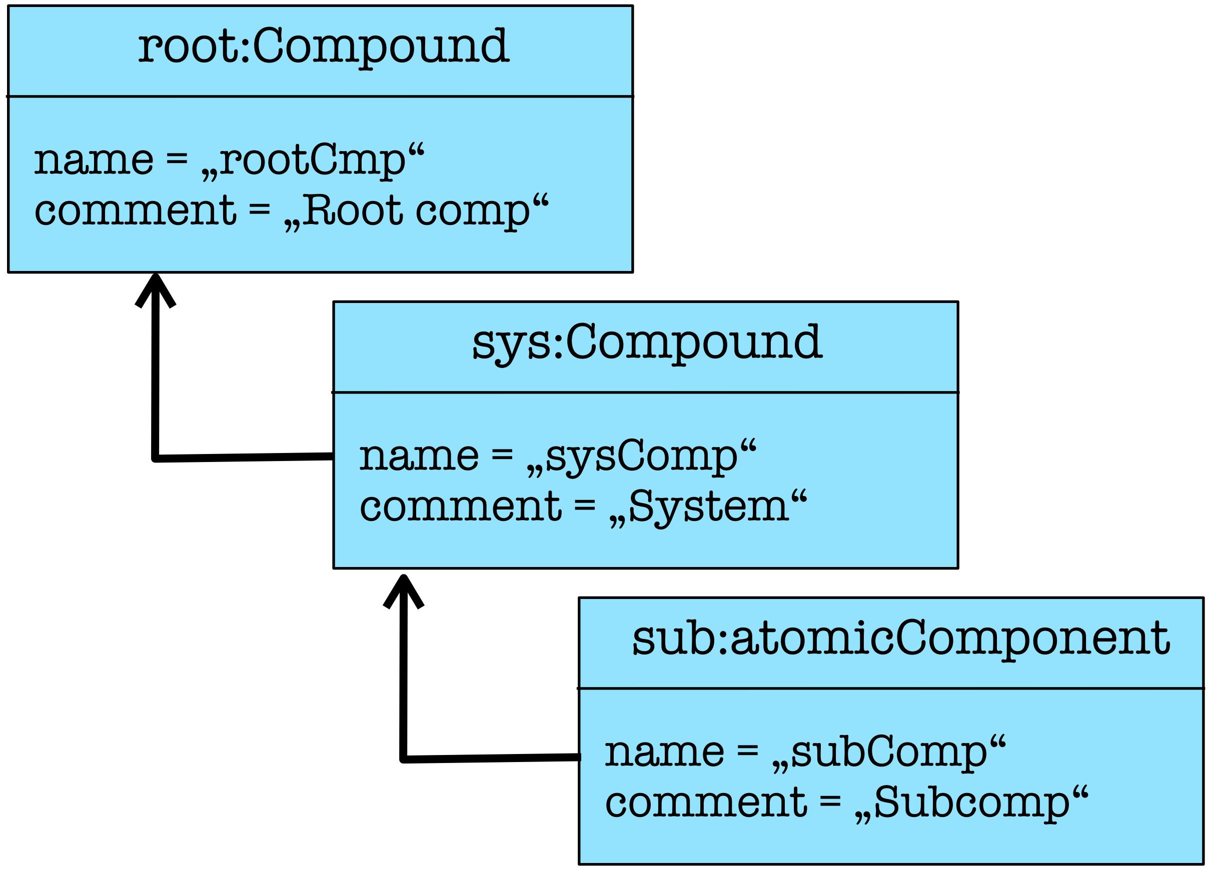
Basics: Classes, Associations



Others (De)Constructions:

- Classes: Class Class has instances Elements
  Example: Comp(Comps,[Root,Sys,Sub])
- Associations: Association Assocation has instances Relations
  Example: subComp(SubComps,[SubSys,SysRoot])

Basics: Structure of the Model



root:Compound

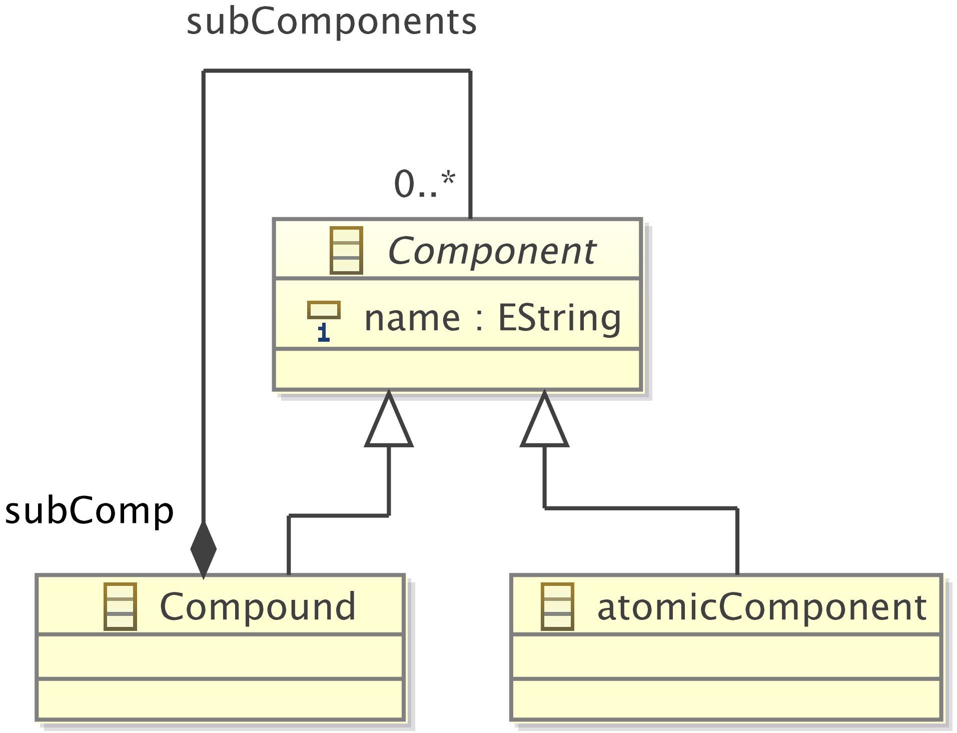
name = „rootCmp“
comment = „Root comp“

sys:Compound

name = „sysComp“
comment = „System“

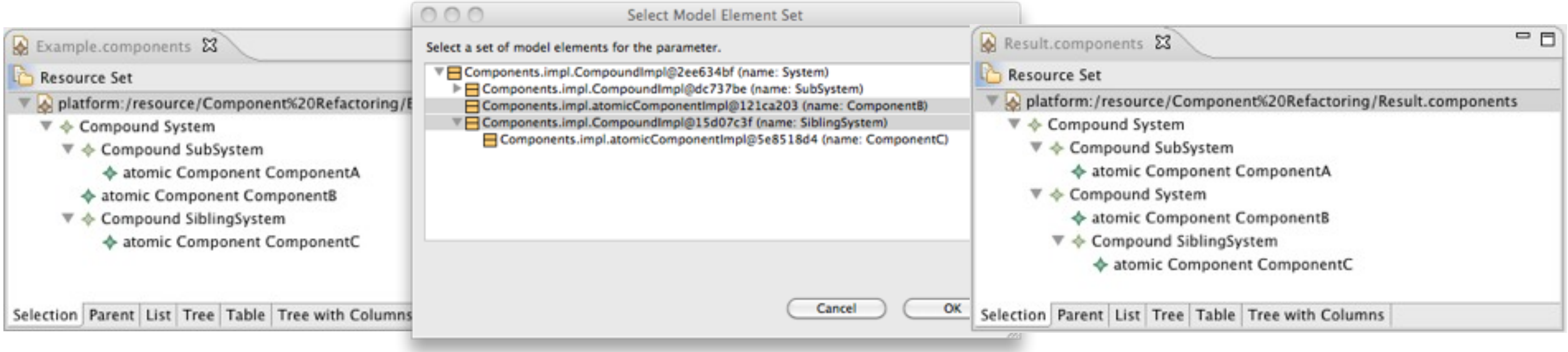
sub:atomicComponent

name = „subComp“
comment = „Subcomp“



- Comps = Compound({ Root, Sys}), Atoms = AtomicComponent({ Sub })
- Root = Compound(root,‘rootCmp‘,‘Root Comp‘),
  Sys = Compound(sys,‘sysComp‘,‘System‘)
- Sub = atomicComponent(sub,‘subComp‘,‘SubComp‘)
- SubComps = subComp({SubSys,SysRoot})
- SubSys = subComp(sub,sys), SysRoot = subComp(sys,root)

Transformation: Relations



```
Resource Set
▼ platform:/resource/Component%20Refactoring/E
  ▼ Compound System
    ▼ Compound SubSystem
        atomic Component ComponentA
      atomic Component ComponentB
    ▼ Compound SiblingSystem
        atomic Component ComponentC
```

Select Model Element Set

Select a set of model elements for the parameter.

```
▼ Components.impl.CompoundImpl@2ee634bf (name: System)
  ▶ Components.impl.CompoundImpl@dc737be (name: SubSystem)
    Components.impl.atomicComponentImpl@121ca203 (name: ComponentB)
  ▼ Components.impl.CompoundImpl@15d07c3f (name: SiblingSystem)
      Components.impl.atomicComponentImpl@5e8518d4 (name: ComponentC)
```

```
Resource Set
▼ platform:/resource/Component%20Refactoring/Result.components
  ▼ Compound System
    ▼ Compound SubSystem
        atomic Component ComponentA
    ▼ Compound System
        atomic Component ComponentB
      ▼ Compound SiblingSystem
          atomic Component ComponentC
```

Model Transformation: (Bi-Directional) Relation

- Pre-Model: Model before transformation
- Parameters: Set of elements to be clustered
- Post-Model: Model after transformation
- Example: cluster(Pre,Group,Post)

Transformation: De-/Construction

```
cluster(Pre,Group,Post) :-
  Architecture(Pre,PreClass,PreAssoc),
  Compound(PreComp,PreComps), OtherClass ∪ [PreCmp] = PreClass,
  subComp(PreSub,PreSubs), Assocs ∪ [PreSub] = PreAssoc,

  link(PreSubs,Group,OldRoot,OutSubs), unlink all Group elements from OldRoot
  Compound(OldCmp,OldRoot,Name),[OldCmp] ∪ Cmps = PreCmps,
  subComp(Sub,OldRoot,NewRoot), [Sub] ∪ OutSubs = InSubs,
  Compound(NewCmp,NewRoot,Name), [OldCmp,NewCmp] ∪ Cmps = PostCmps,
  link(PostSubs,Group,NewRoot,InSubs), link all Group elements to NewRoot

  subComp(PostSub,PostSubs), Assocs ∪ [PostSub] = PostAssoc,
  Compound(PostComp,PostComps), OtherClass ∪ [PostComp] = PostClass,
 Architecture(Post,PostClass,PostAssoc).
```

Transformation: Rules

The OutSubs subComp relation is an extension of the InSub subComp relation by a Group linked under Root iff

- Either: Group is empty and InSub is OutSub
- Or: OutSubs is a corresponding extension of InSub extended by linking some element Sub of Group to Root with the Rest of the Group linked under Root

- link(InSubs,Group,Root,OutSubs) :-
  Group = [], InSubs = OutSubs.
- link(InSubs,Group,Root,OutSubs) :-
  subComp(SubRel,Sub,Root),
  union([Sub],Rest,Group),union([SubRel],Subs,InSubs),
  link(Subs,Rest,Root,OutSubs).

Conclusion: Relation-Based Declarative Model Transformations

- Transformation: Declarative, rule-based, relational
  - Relational: Side-effect free for back-tracking
  - Declarative: Implicit unification for constraint-solving
  - Rule-based: Explicit control-flow for composition

- Application: Transformation in model-based development
  - Medium-sized models: Up to 3000 elements and 5000 relations
  - Complex transformations: Automated deployment, optimizations
  - Verified transformations: Formal verification with theorem prover