

Vorlesung "Software-Engineering"

Prof. Ralf Möller, TUHH, Arbeitsbereich STS

Übung: Miguel Garcia

- **Vorige Vorlesung:**
 - Object Constraint Language (OCL)
- **Heute:**
 - Metamodeling

Diese Vorlesung basiert auf:

Metamodeling

Dr Jon Whittle

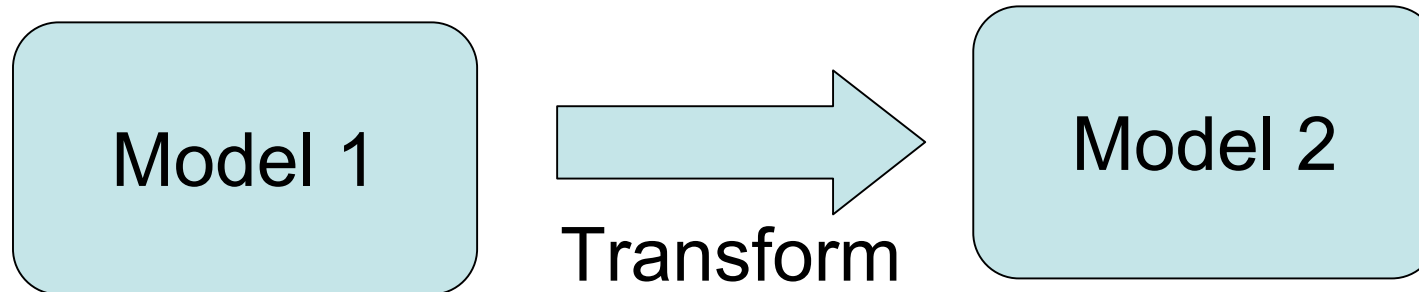
Visiting Prof, TU Braunschweig

Associate Prof, George Mason University

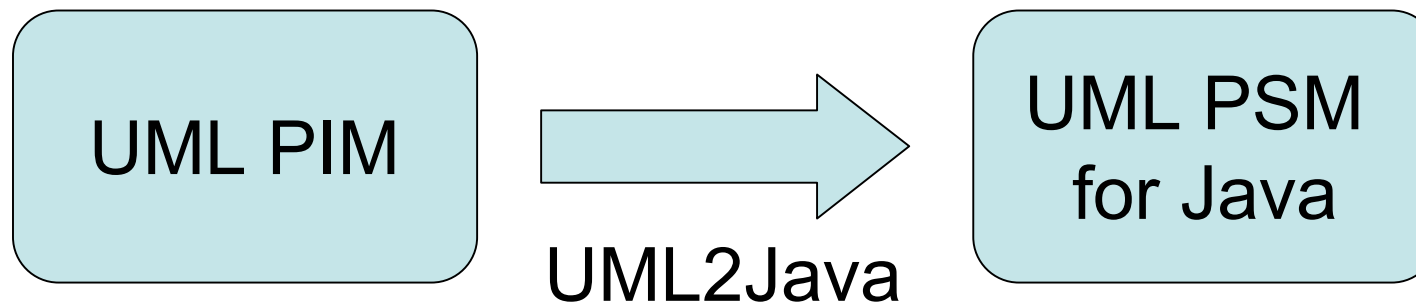
This Week

- Metamodeling:
 - What is it?
 - Why do we need it?
 - How do we do it?
 - Is it really new?
- Reading:
 - “Applied Metamodeling” chs. 1,2,6 (download for free from <http://www.xactium.com> – registration required)
 - “Model Driven Architecture” (David Frankel) ch. 5
 - “Software Factories” ch.8

Recall: MDA/MDD



How do we define Model 1 and Model2?



PIM: Platform-Independent Model, PSM: Platform-Specific Model

Metamodeling

- Language-driven engineering (LDE):
 - Why? *The right abstraction for the right job*
- Metamodel: a model of a language capturing its important properties:
 - Concepts
 - Syntax
 - Semantics
- Metamodels *enable* LDE if there is a common meta-metamodel

Language Features

- All system development languages share these features:
 - Concrete syntax: notation (textual or visual) to represent language concepts
 - Abstract syntax: form/structure of language concepts
 - Semantics: meaning of concepts
 - Mappings
 - Extensibility

UML Metamodel architecture

Layer	Description	Example
meta-metamodel	The infrastructure for a metamodeling architecture. Defines the language for specifying metamodels.	<i>MetaClass, MetaAttribute, MetaOperation</i>
metamodel	An instance of a meta-metamodel. Defines the language for specifying a model.	<i>Class, Attribute, Operation, Component</i>
model	An instance of a metamodel. Defines a language to describe an information domain.	<i>StockShare, askPrice, sellLimitOrder, StockQuoteServer</i>
user objects (user data)	An instance of a model. Defines a specific information domain.	<i><Acme_Software_Share_98789>, 654.56, sell_limit_order, <Stock_Quote_Svr_32123></i>

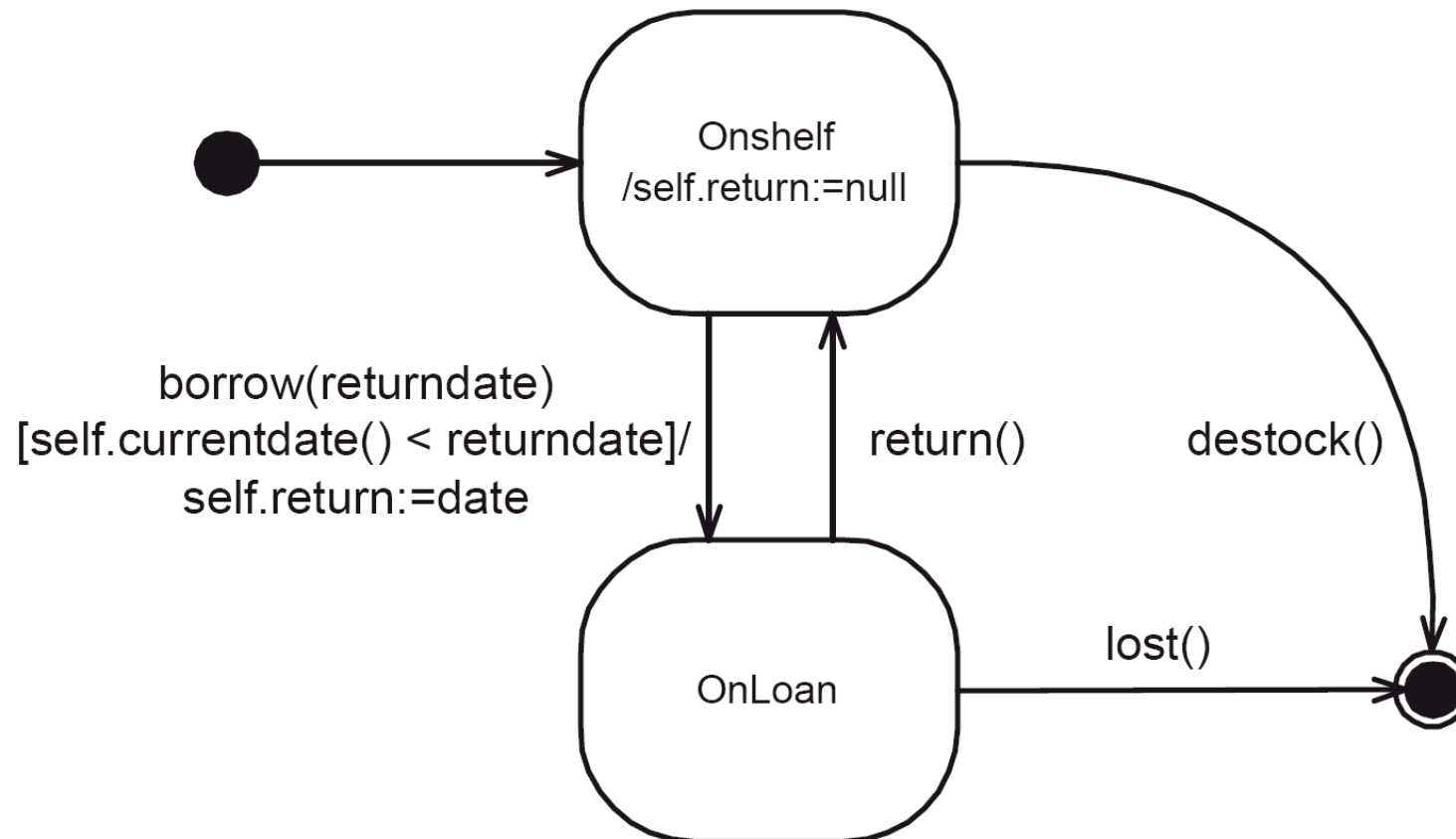
How to write a metamodel

- 1) Define abstract syntax
- 2) Define well-formedness rules and meta-operations
- 3) Define concrete syntax
- 4) Define semantics
- 5) Define mappings to other languages

1. Define Abstract Syntax

- 1) Concept identification
- 2) Concept modeling
- 3) Model architecting
- 4) Model validation
- 5) Model testing

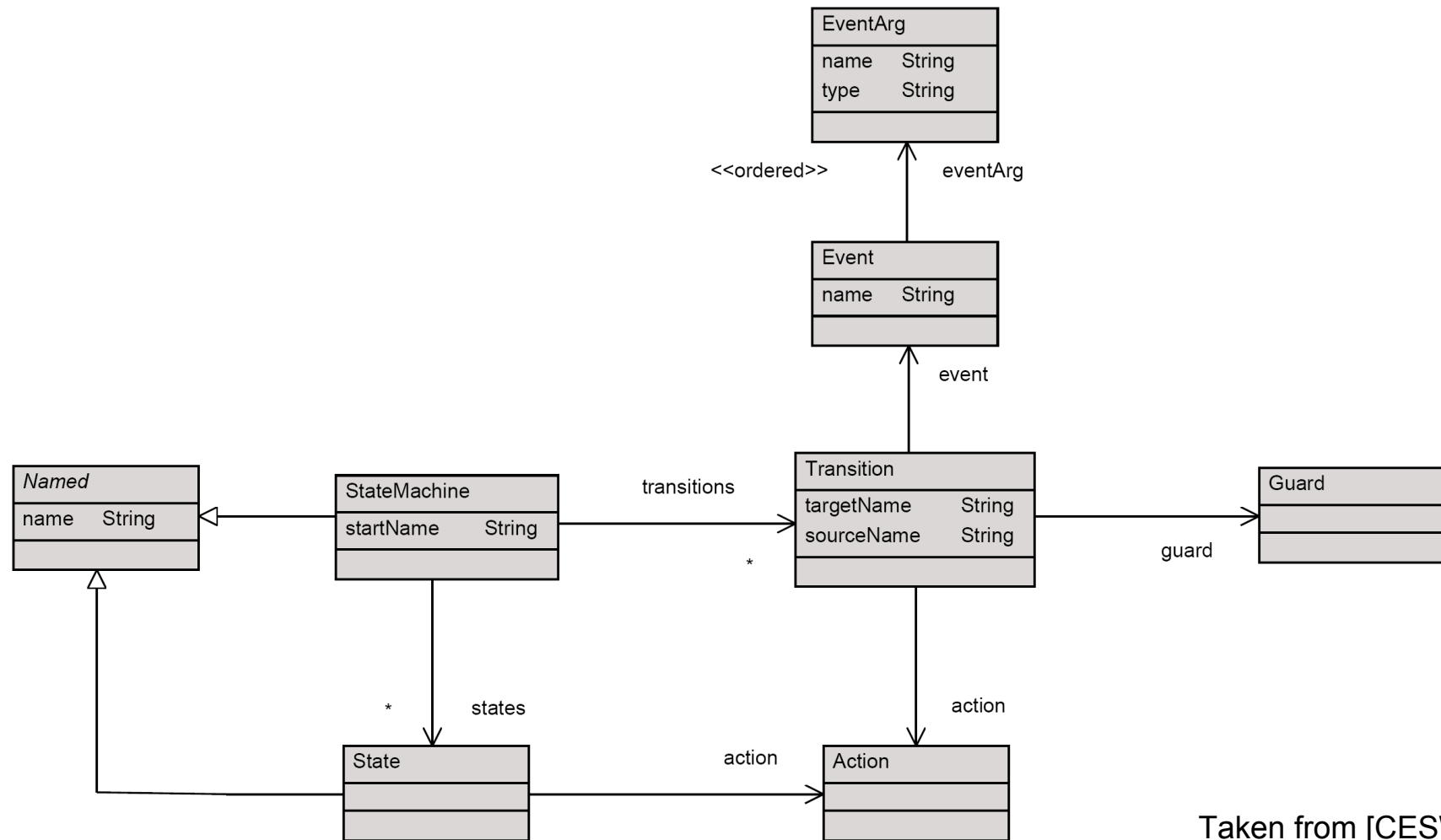
Example: state machines



Concept Identification

- State
- Initial state
- Final state
- Transition
- Event
- Action
- Guard
- Etc.

Concept Modeling

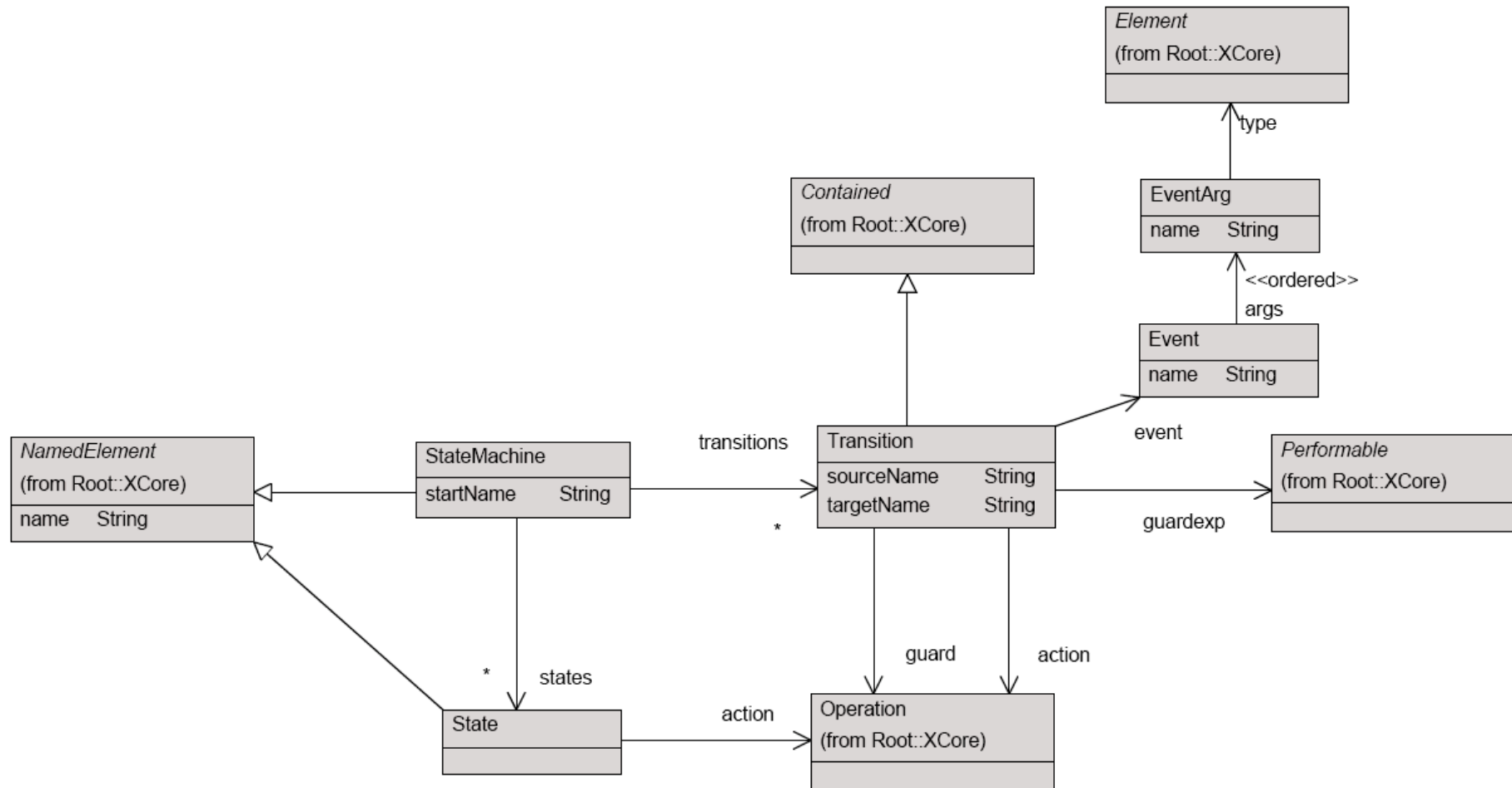


Taken from [CESW04]

Model Architecting

- Named elements
- Contained elements
- Expression elements

Model Architecting (new)



Taken from [CESW04]

Well-formedness rules

```
context StateMachine
  @Constraint StatesHaveUniqueNames
  states->forall(s1 |
    states->forall(s2 |
      s1.name = s2.name implies s1 = s2))
end
```

```
context StateMachine
  @Constraint StatesIncludeInitialState
  states->exists(s | s.name = startName)
```

Language: XMF, a slight variation of OCL

Operations

```
context StateMachine
  @Operation addState(state:StateMachines::State)
    if not self.states->exists(s | s.name = state.name) then
      self.states := states->including(state);
      self.state.owner := self
    else
      self.error("Cannot add a state that already exists")
    end
end
```

Queries

```
context StateMachine
  @Operation startingState()
    if states->exists(s | s.name = startName) then
      states->select(s | s.name = startName)->sel
    else
      self.error("Cannot find starting state: " + startName)
    end
  end
end
```


Defining Semantics

- How?
 - In terms of concepts that already have a well-defined meaning
 - By describing the properties of a concept
 - As a specialization of another concept

Informal Semantics...

- ...is not enough because a language with informally defined semantics:
 - Can be misinterpreted/misused
 - Cannot be understood by tools
 - Cannot be analyzed (e.g., for consistency)
 - Makes defining language extensions difficult

Approaches to Defining Semantics

- **Translational:** translate to concepts with a well-defined semantics in another language
- **Extensional:** extend semantics of existing concepts
- **Operational:** model the operational behavior of concepts
- **Denotational:** model the mapping to semantic domain concepts

Examples

- **Translational:**
 - map a state machine metamodel to a MOF-based metamodel
 - Map a state machine metamodel to a Java metamodel
- **Operational:**
 - Write an interpreter for a state machine
- **Extensional:**
 - Make StateMachine inherit from Class (both are instantiable) and hence can have multiple instances
- **Denotational:**
 - Denotation of a class is set of all objects that may be an instance of it
 - Denotation of an action is set of all state changes that can occur as a result of its invocation

Examples for Defining Languages

The UML Approach

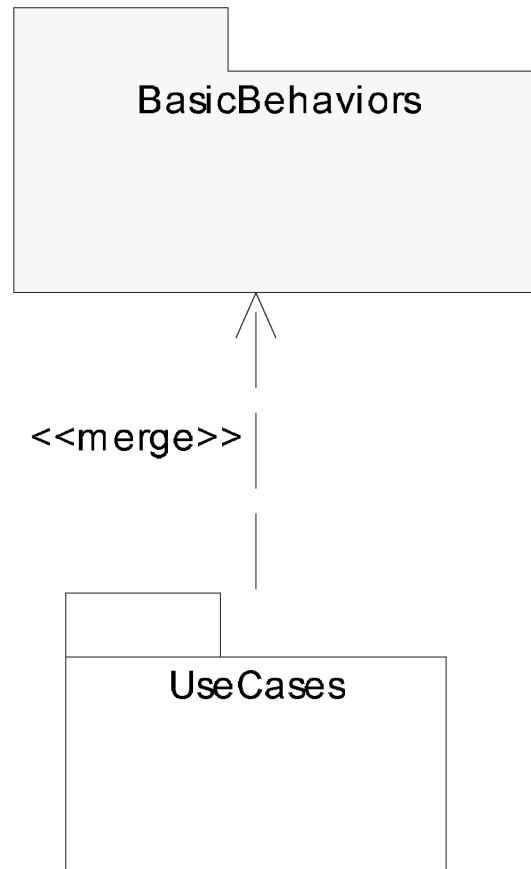
- Abstract syntax in MOF (Meta-Object Facility):
http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF
- Well-formedness constraints in OCL
- Semantics given informally (in English)
 - *Semantic variation points* (e.g., order of firing transitions in orthogonal regions)
- Concrete Syntax: notation tables

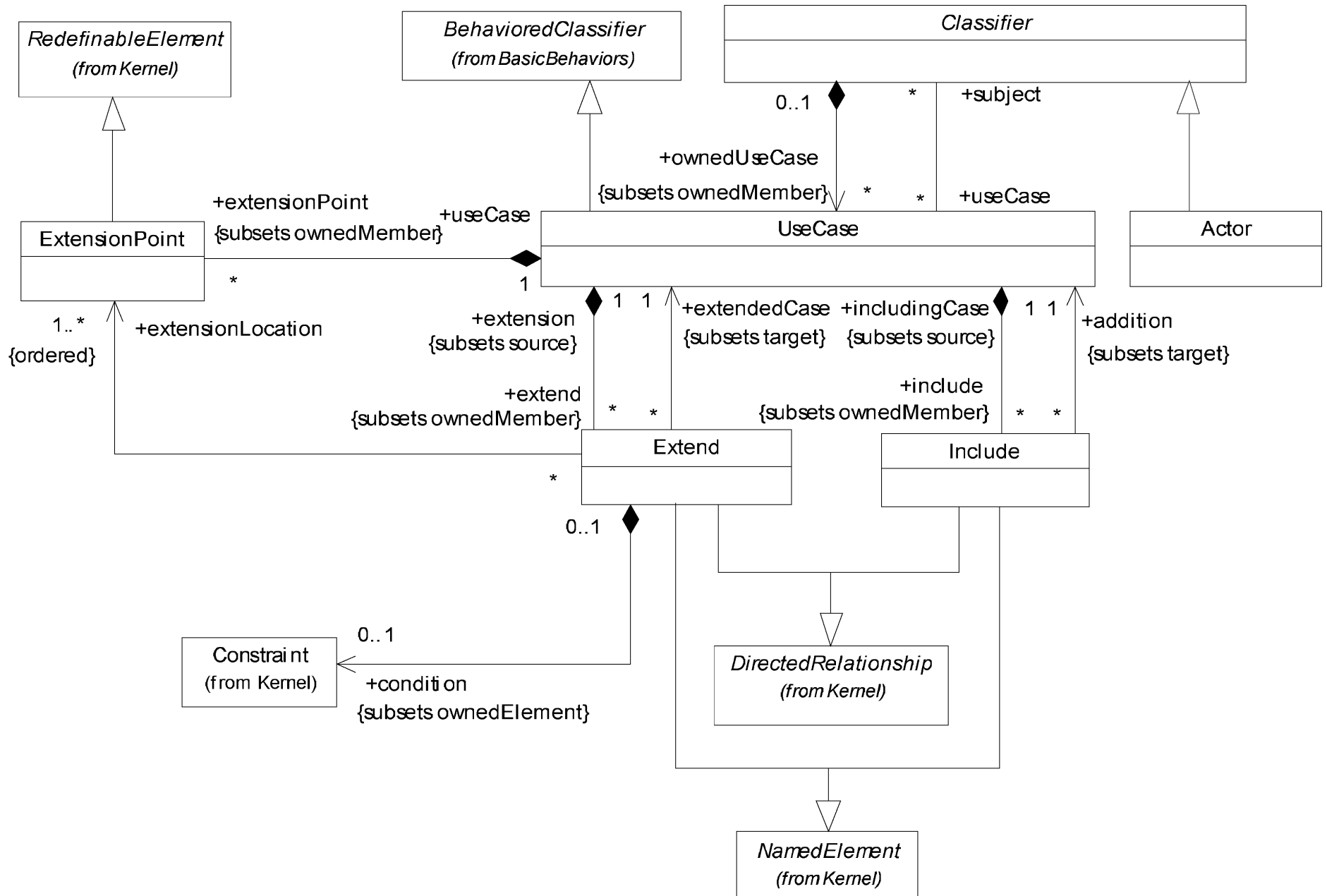
MOF

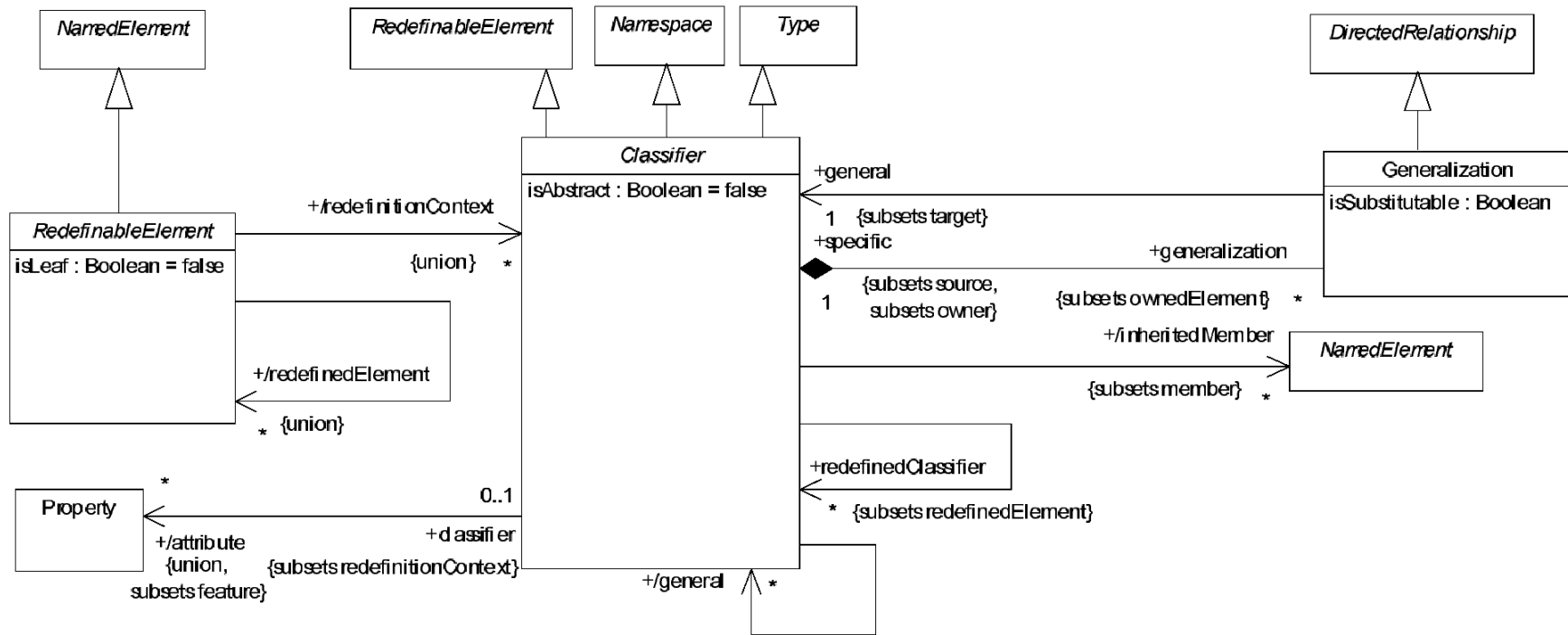
- (essentially) a subset of UML
- Has been used to define metamodels for a variety of languages: UML, CWM, CCM
- Simpler than UML:
 - No association classes
 - No qualified associations
 - No dependency relationships
 - No n-ary associations

CWM: Common Warehouse Metamodel, CCM: Corba Component Model

Example: Use Cases







Classifier = a set of instances with features in common

UML Use Case Description

- Class: Actor
- Description:
- Attributes: none further
- Associations: none further
- Constraints:
 - 1) An actor can only have associations to use cases, components and classes. These associations must be binary
 - 2) An actor must have a name
- Semantics: ...multiplicity at use case end...
- Notation:.....
- Presentation options: ...

Constraints (formalized)

1)

```
self.ownedAttribute->forAll ( a |  
  (a.association->notEmpty()) implies  
    ((a.association.memberEnd.size() = 2) and  
     (a.opposite.class.ocllsKindOf(UseCase) or  
      (a.opposite.class.ocllsKindOf(Class) and not a.opposite.class.ocllsKindOf(Behavior))))
```

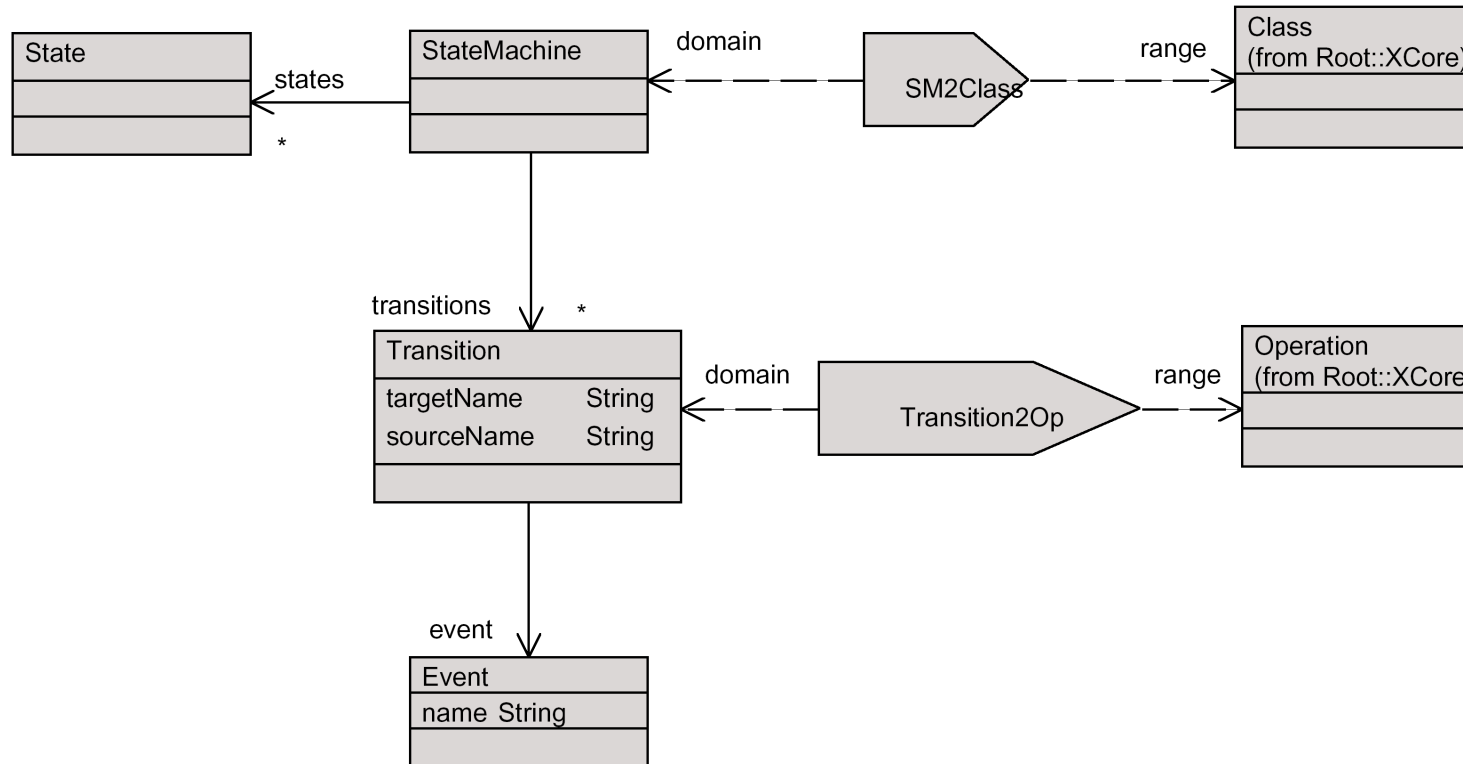
2)

```
name->notEmpty()
```

Translational Semantics

- “translate to something we know”
(cf. compilation)
- Commonly used to reduce a language to a core set of concepts: e.g., flatten hierarchical state machines

Example: State Machines



Example: State Machines

- State Machine -> Class called *state* whose values are the enumerated state values of the machine
- The Class will inherit all the attributes/operations of the State Machine's owning class
- Transition -> Operation

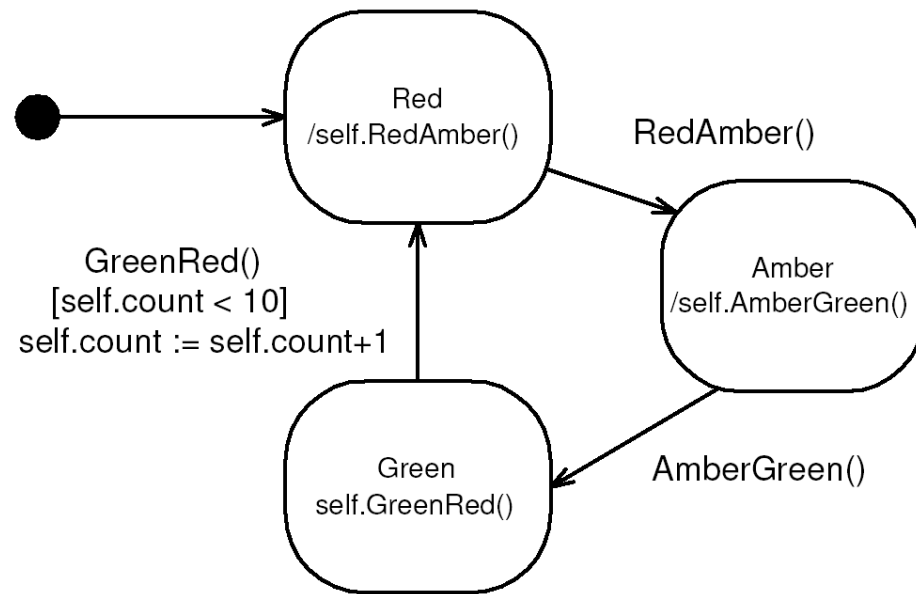


Figure 6.2: The traffic light example

```

@Operation GreenRed()
  if g() then
    a();
    self.state := "Red"
  end
end
end
  
```

Taken from [CESW04]

Operational Semantics

- Operational semantics = write an interpreter
- Expressed in terms of the language itself not another language
- 3 elements (typically):
 - Define operations on language concepts that define their operational semantics (e.g., run() for state machine)
 - Take environment parameter (e.g., variable bindings)
 - Operations return the result of the evaluation

Example: State Machines

```
@Operation run(element)
  let state = self.startingState() in
    @While state <> null do
      let result = state.activate(element) then
        transitions = self.transitionsFrom(state) then
          enabledTransitions = transitions->select(t |
            t.isEnabled(element,result) and
            if t.event <> null then
              messages->head().name = t.event.name
            else
              true
            end) in
            if enabledTransitions->isEmpty then
              state := null
            else
              let transition = enabledTransitions->sel in
                transition.activate(element,result + self.messages->
                  head().args.value);
                state := transition.target();
                if transition.event <> null then
                  self.messages := self.messages->tail()
                end
              end
            end
          end
        end
      end
    end
  end
end
```

Taken from [CESW04]

Extensional Semantics

- Use existing elements and specialization
- E.g., make a use case an activity
- Be careful: you also inherit the semantics

Denotational Semantics

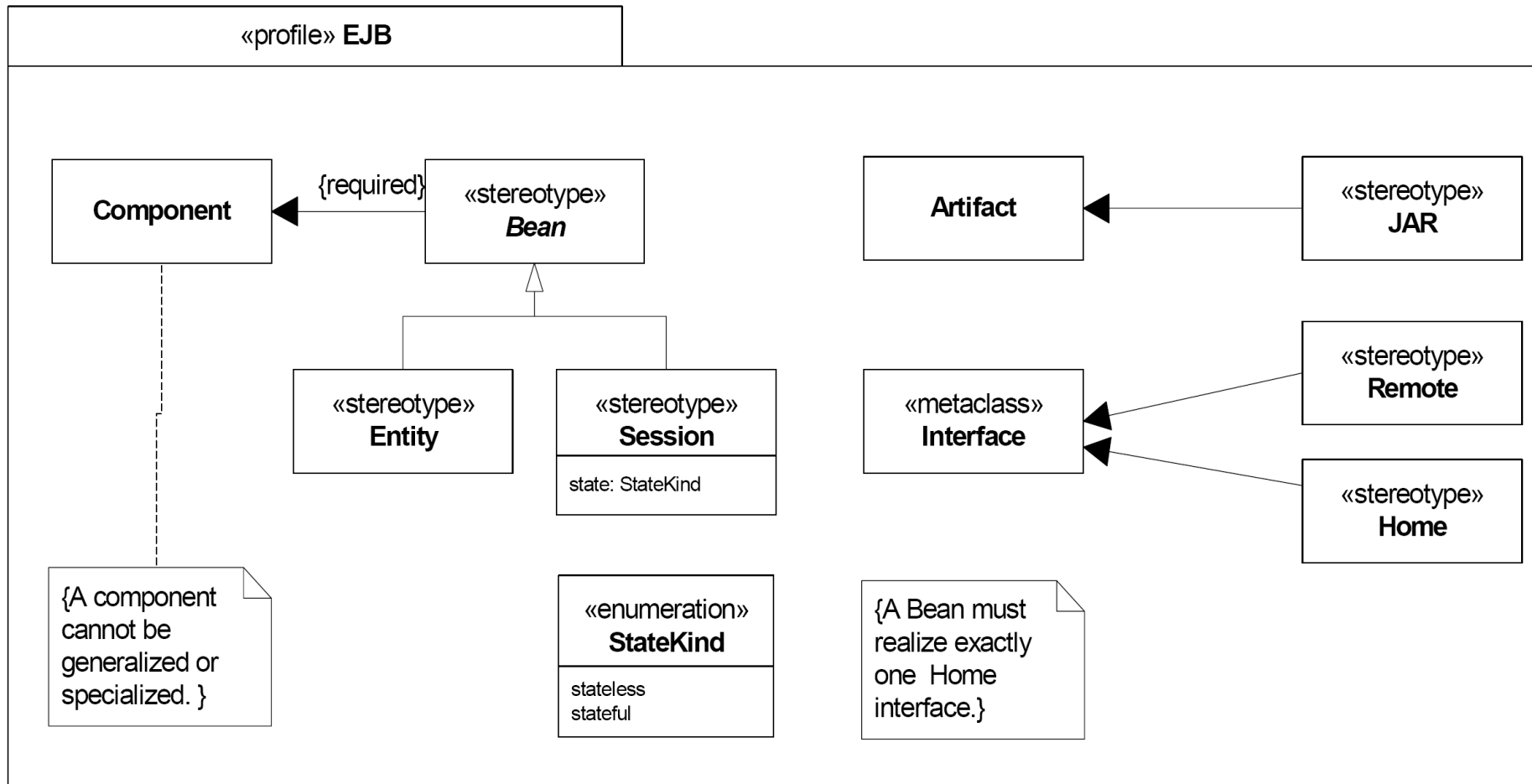
- Declarative description of semantics
- Does not commit to a particular operational semantics implementation
- Example: sequence diagrams

UML Profiles

- Lightweight approach to tailoring UML
 - E.g. for platforms (.NET, EJB)
 - E.g., for domains (telecomms, real-time)
- Adapt an existing metamodel
 - Cannot change existing metaclasses
- In particular, can only add new constraints
 - Cannot replace existing constraints

Extensions





Metamodels vs CFGs

(what follows is from “Software Factories” pps. 286-)

- OSL: simple language containing classes and protocol state machines
- Class: attributes + operations
- Operations: assignment only
- State machines: protocols only

BNF abstract syntax

Model ::= (ModelElement)*

ModelElement ::= Class

Class ::= ClassName (Features)* (StateMachine)?

ClassName ::= Identifier

Feature ::= Attribute | Method

Attribute ::= AttributeName TypeRef

AttributeName ::= Identifier

TypeRef ::= Identifier

Method ::= MethodName (Argument)* Statement

MethodName ::= Identifier

Argument ::= ArgumentName TypeRef

ArgumentName ::= Identifier

BNF (contd)

Statement ::= (Statement)* | AssignmentStatement

AssignmentStatement ::= LHS RHS

LHS ::= AttributeRef

AttributeRef ::= Identifier

RHS ::= AttributeRef | ArgumentRef

ArgumentRef ::= Identifier

StateMachine ::= State

State ::= StateName (StartState)? (State)*
(Transition)*

StateName ::= Identifier

StartState ::= StateRef

Transition ::= MethodRef StateRef

MethodRef ::= Identifier

StateRef ::= Identifier

From BNF to metamodel

- ::= _ composite aggregation
- Non-terminal _ class
- Terminal _ datatype
- References _ attributes or associations
(cf. Transition)
- *,+,? _ association end multiplicities
- Choice _ generalization
- X ::= Y productions _ generalization or aggregation
- Scope rules _ OCL constraints
- Type checking _ OCL constraints

Comparison (up for discussion)

- Metamodels are more readable
 - References (e.g., StateRef) don't capture intention easily
 - Levels of indirection in CFGs (cf. StartState)
 - $X ::= Y$ rules _ intention not easily captured in CFG
- Metamodels are graphs not trees
- Metamodels define self-contained namespaces (cf. ClassName, AttributeName etc.)
- OCL is a more declarative way of capturing necessary contextual information

References

- [CESW04] "Applied Metamodelling: a foundation for language driven development," Tony Clark, Andy Evans, Paul Sammut, James Willans. Xactium.com
- [GS05] "Software Factories," Jack Greenfield and Keith Short, Wiley