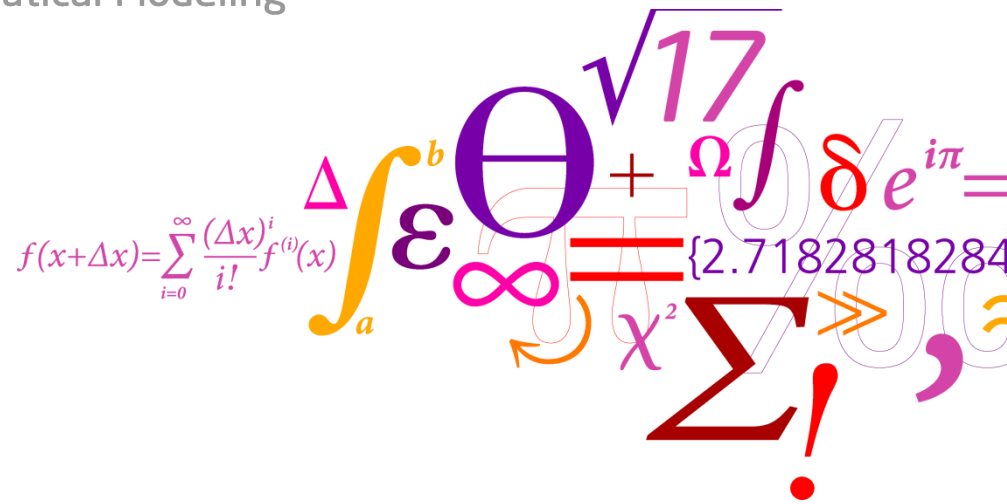
Advanced Topics in Software Engineering (02265)

Ekkart Kindler

DTU Informatics
Department of Informatics and Mathematical Modeling



IV. (Meta-) Modelling Frameworks

1. Motivation & Overview

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler

- With MOF, we have:
  - Modelling notation (EMOF / CMOF)
  - Java API (JMI, EMF, etc)
  - Instance model / reflective interface
  - Transfer syntax for models and instances (XMI)

- How do we
  - create meta-models, models, and instances?
  - use and change model instances?
  - build editors / viewers for models (and instances)?
  - keep posted on instance changes?
  - control / serialise accesses to a model?
  - formulate and guarantee some additional consistency requirements?

Motivation and Overview

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler

Modelling frameworks provide this (and more) functionality:

- Editors
- Editor framworks
- Notification mechanisms
- Adapter and delegation mechanisms
- Commands
- Transactions
- Validation

In the tutorials, we use EMF / EMFT as a concrete example.

And a bit more on GMF in tutorial today (project 1)!

This lecture will provide just an overview of some of the underlying ideas and concepts.

2. Architecture

Actually, this is more on concepts and patterns within the architecture.

- Model View Controller
- Observer pattern
- Factory
- Delegation
- Command framework

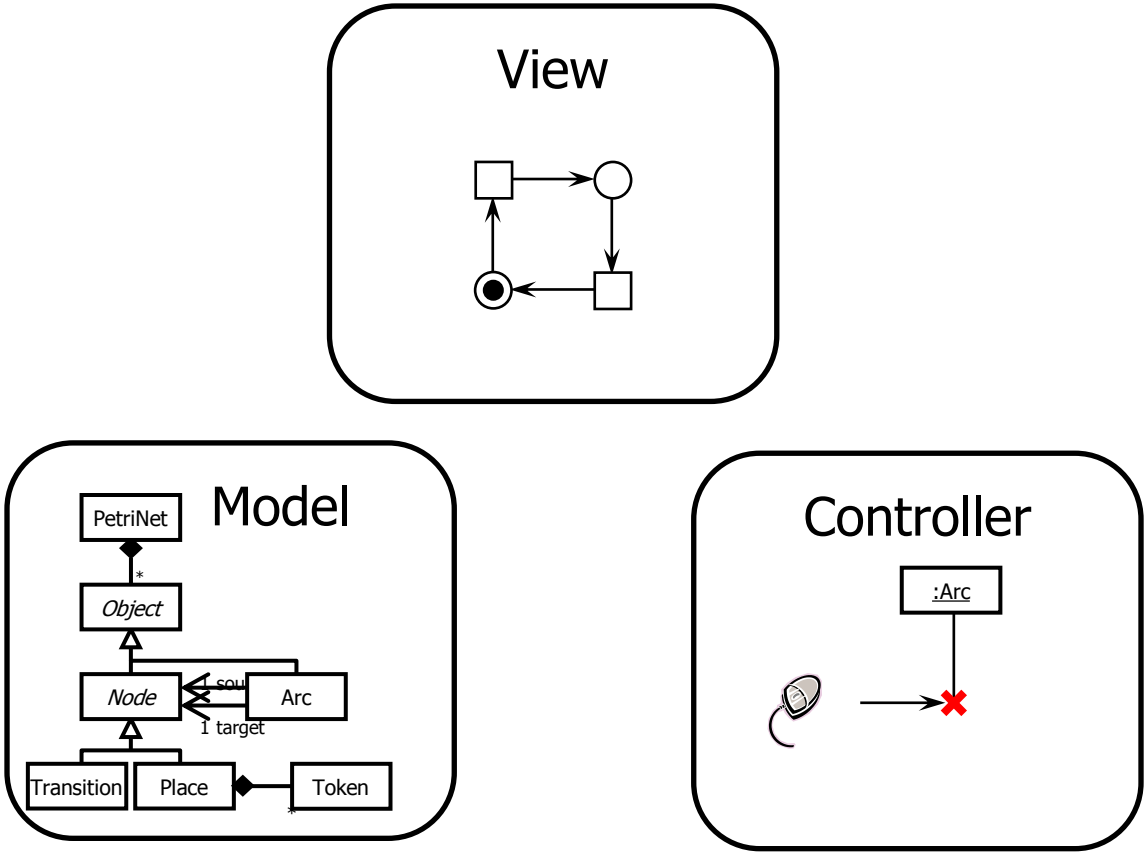
just a glimps (reminder)

- Transactions

Design patterns would be a course (seminar) in its own right!

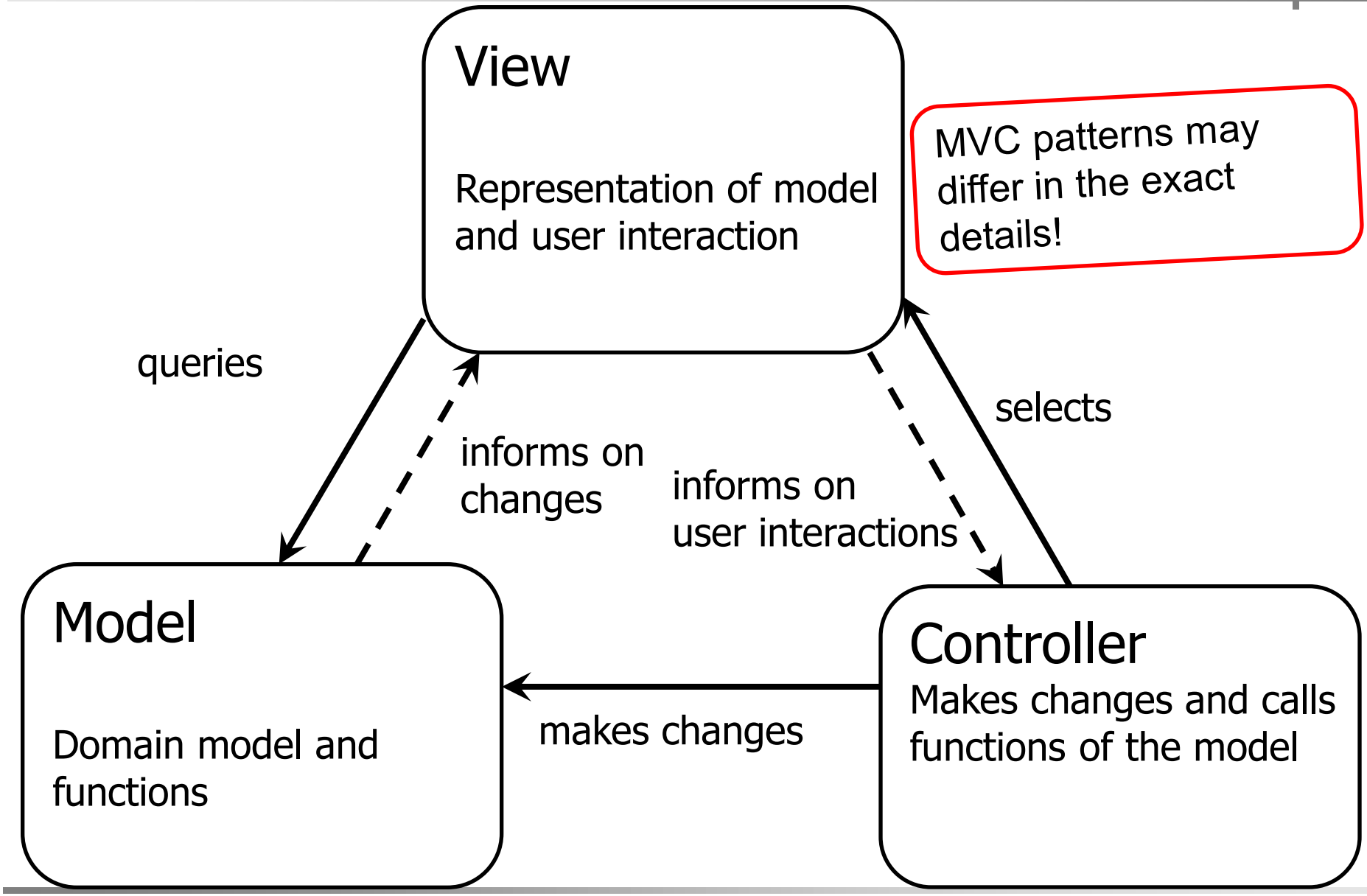
Model View Controller (MVC)

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler



MVC

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler



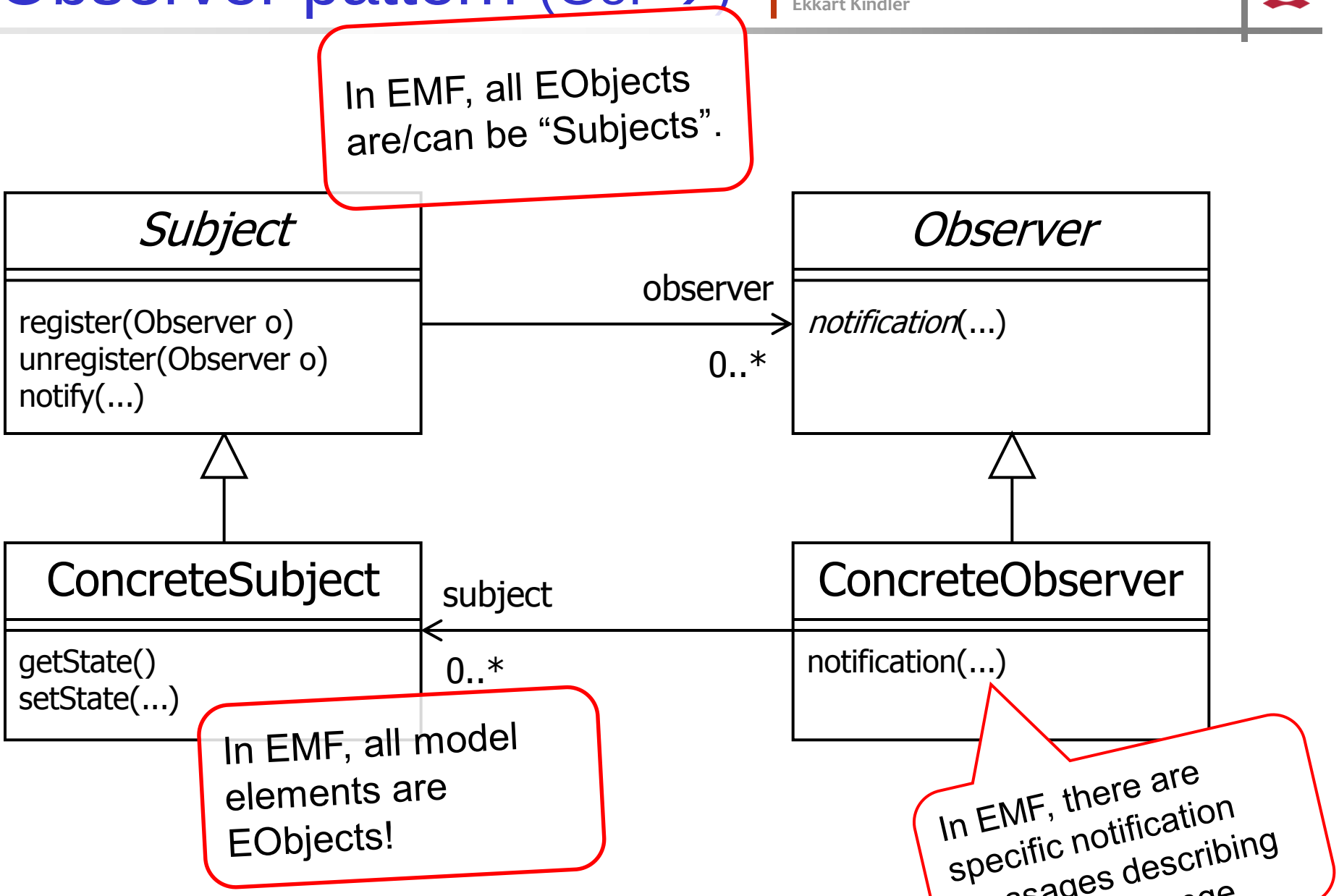
Notification

- When different parts of an application can change a model, the different parts need to know about their changes
- The model itself can best take care of notifying the others by implementing the observation pattern (as subject → next slide)

The ECNO engine makes extensive use of that. Changes of the underlying model, will automatically update the possible interactions at the GUI.

Observer pattern (GoF→)

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler



GoF = ”Gang of Four”

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler

Design Patterns in Software Engineering (*) were introduced by

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides:
Design Patterns, Addison Wesley 1995

These four are affectionaly called ”The Gang of Four” (GoF).

(*) The idea of using “Design Patterns” came from architecture: Christopher Alexander, 1977.

Factory

Create objects of a type

- even if type is an interface
- where the concrete class for object is decided at runtime
- when helpers are needed to decide the right class or to properly instantiate an the object (in some context)

Examples in EMF were briefly discussed in the tutorials:
e.g. see PetriNetsFactory.java (or PNVisFactory)

⇒ separation of interfaces from implementation

⇒ creation of objects can be delegated to the right instance

Command Framework

This should not be confused with the GoF Command pattern.

Problems:

- Make changes on a model / instance so that they can be easily undone.
- Define the changes belonging to each other.

We need that for different reasons:

- Undo/redo in editors or even a simulation
- In the context of additional constraints:
  it is difficult to forsee, whether a model will be consistent after some changes are made;
  with an undo feature, we can make the change, then check if the result is consistent, and, if it is not, undo the change

→ see also “Validation”

Command Framework

The editing domain serves many other purposes.

- For each model / instance, there is an editing domain that maintains a command stack
- There are factories for creating commands for all the types of operations:
  - create
  - delete
  - add elements
  - set attributes
  - ...

  and for building more complexe commands.

See Petri net example from the first tutorial for a simple example.

- The editing domain can also be used as a factory for creating commands (which will be **delegated** to the right factory).

In EMF, e.g. the ItemProviders.

Programming commands

- Programming commands is tedious (not difficult thorugh);
- a lot of syntax for almost nothing
- This is nothing human programmers would like to do (see Tutorial 1)
- In some situtations (when using GMF), you can use Recording commands – basically programming as usual except for the set up (tutorial 2).
- When changes are made in a GMF editor, and react to these notifications, you can program this behaviour since the notifications run inside a recording command (tutorial 4) already.

Recording commands

- In some situtations (when using GMF), you can use Recording commands – basically programming as usual except for the set up (tutorial 2).
- When changes are made in a GMF editor, and react to these notifications, you can program this behaviour since the notifications run inside a recording command (tutorial 4) already.

- The secret behind is a special kind of EditionDomain: a transactional editing domain.

3. OCL (and Validation)

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler

- Many of the requirements of a model cannot be expressed in EMOF or CMOF (or even UML) alone!
- Validity checks could be implemented as methods, but this would be dependent on the programming language

and, remember, “we don’t like programming” ;-)

⇒ OCL (Object Constraint Language) is independent from a concrete programing language and tuned to formulate typical constraints on MOF or UML models

⇒ OCL also allows us to formulate pre and post conditions of methods

See option “body” later

⇒ or even to ”implement” some methods or derived attributes

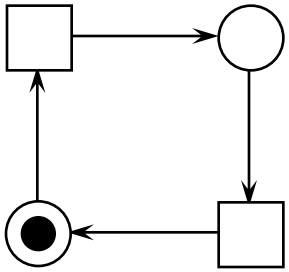
⇒ can be used for validation (at different points of the lifecycle of a model)

Example (cf. L01)

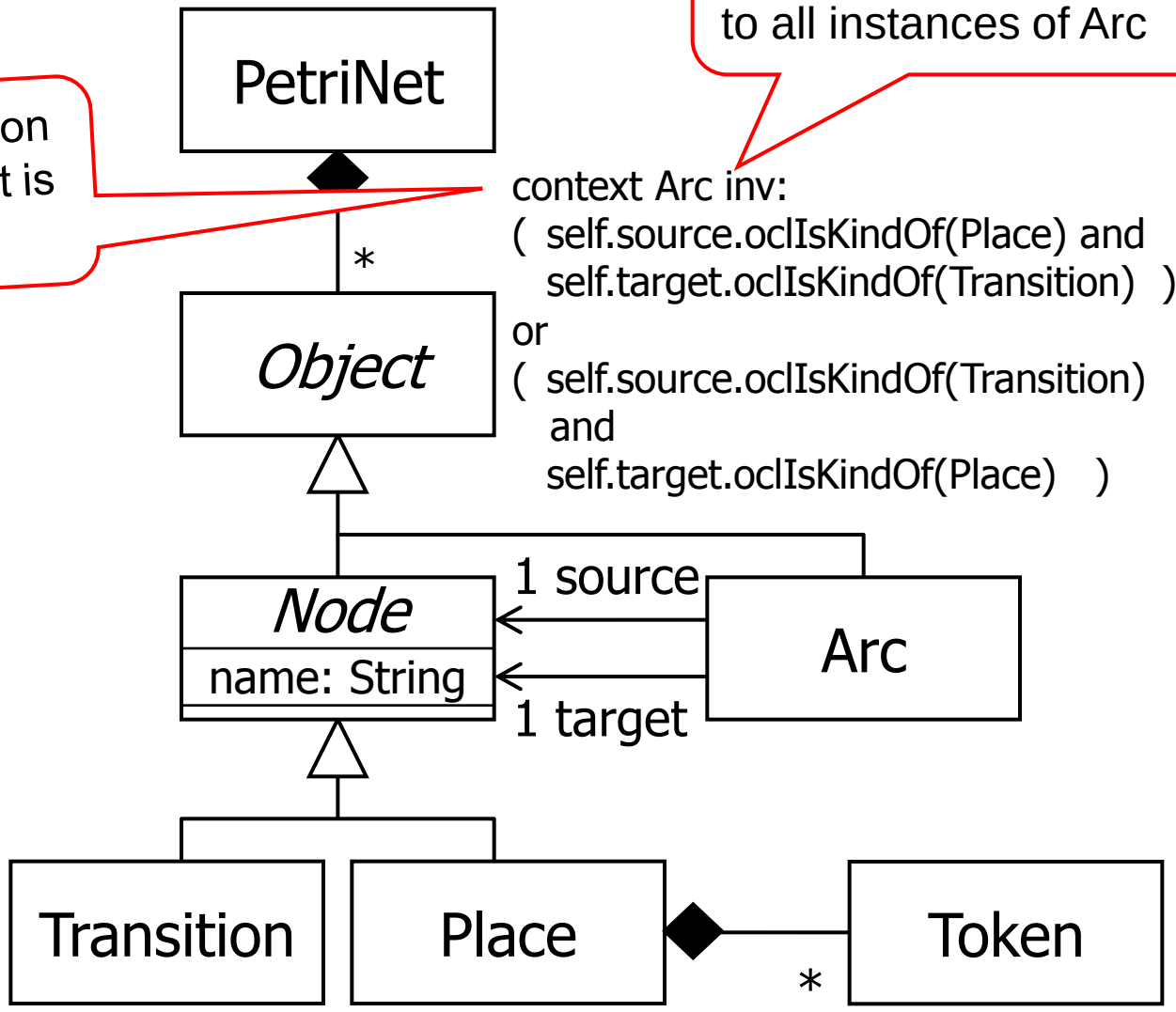
DTU Informatics
Department of Info
Ekkart Kindler

Stands for <<invariant>>: applies to all instances of Arc

Defines the context on which this constraint is applied



Petri net model



Meta-model for Petri nets

OCL basics

OCL expressions

- start from the context object
  (in OCL called **self**!)
- may access attributes and methods
  (by dot-notation and resp. names)
- may navigate along associations
- may call operations and built-in functions

OCL contex

- The context can be a class
  (self refers to an instance of that class)
  - option inv

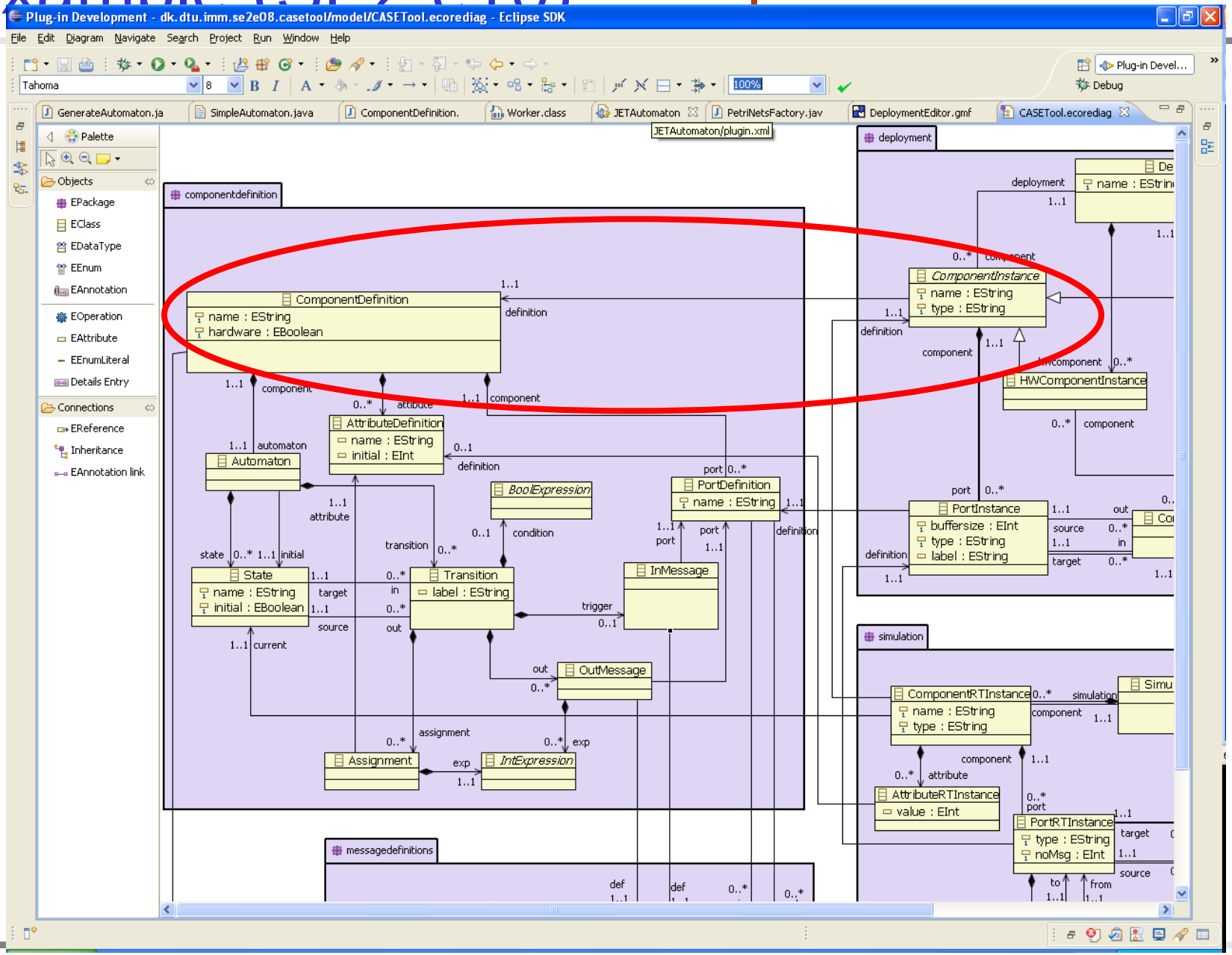
Today, it depends on tool / framework in which way an expression is attached to a UML model and a context.

- The context can be an operation
  (self refers to the object on which the operation is called)
  - options pre, post and body:
    pre and post define a pre- and post-condition
    body defines the result.

In post, we can refer to values before execution by the @pre tag

- The context can be an attribute or association
  (self refers to the object to which it belongs)
  - options init and derived

Example (SE2 e10)



Example (SE2)

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler

```
context HWComponentInstance inv:
    self.definition->size() > 0 and
    self.definition.hardware
```

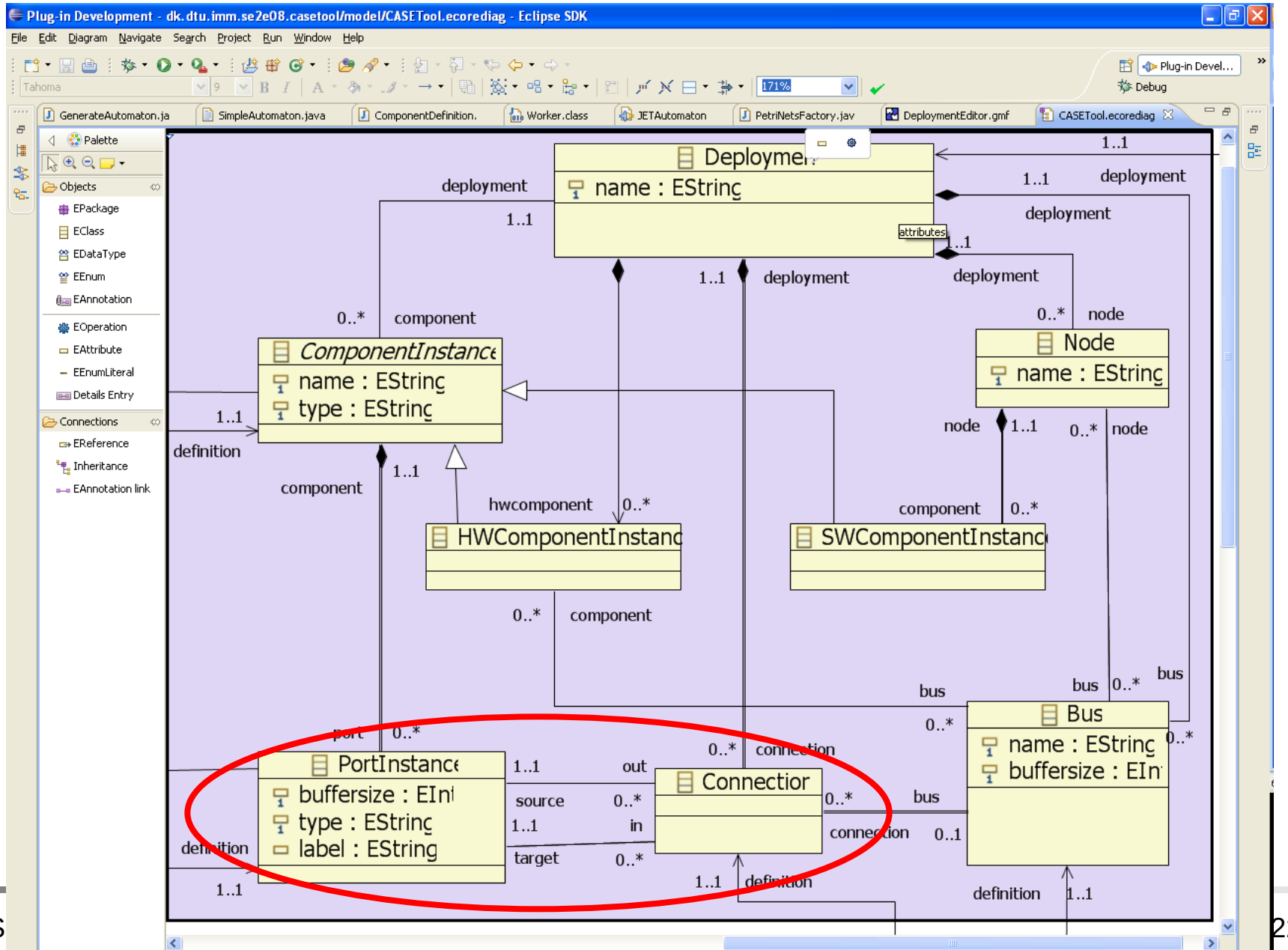
size() > 0 is one way of checking for non-null value.

"->" converts value to a set

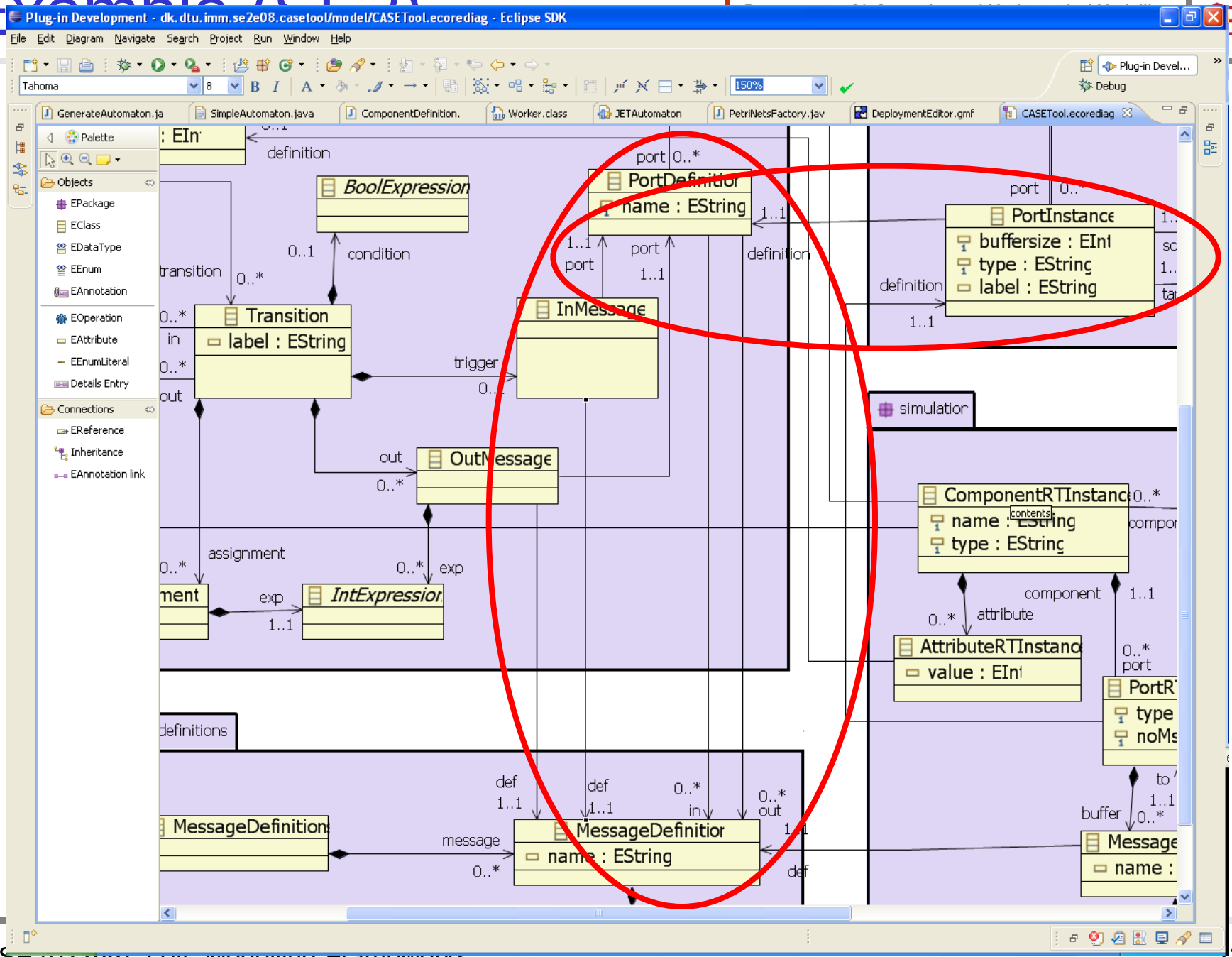
We can also check for *null* with
… .isTypeOf(OclVoid)

Example (SE2 e10)

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler



Example (SE2)


Plug-in Development - dk.dtu.imm.se2e08.casetool/model/CASETool.ecorediag - Eclipse SDK
File Edit Diagram Navigate Search Project Run Window Help
GenerateAutomaton.ja
SimpleAutomaton.java
ComponentDefinition.
Worker.class
JETAutomaton
PetriNetsFactory.jav
DeploymentEditor.gmf
CASETool.ecorediag
Palette
Objects
EPackage
EClass
EDataType
EEnum
EAnnotation
EOperation
EAttribute
EEnumLiteral
Details Entry
Connections
EReference
Inheritance
EAnnotation link
definition
BoolExpression
condition
transition
Transition
label : EString
in
out
trigger
InMessage
OutMessage
assignment
exp
IntExpressior
port
PortDefinition
name : EString
PortInstance
buffersize : EInt
type : EString
label : EString
simulation
ComponentRTInstanc
name : EString
type : EString
contents
component
attribute
AttributeRTInstance
value : EInt
definitions
MessageDefinitions
MessageDefinitior
name : EString
message
def
Message
name :
buffer
to

Example (SE2 e10)

DTU Informatics
Department of Informatics and Mathematical Modelling
Ekkart Kindler

```
context Connection inv:
   self.source.definition.out->forAll( m1 |
     self.target.definition._in->exists(m2 | m1=m2 ))
  and
   self.target.definition.out->forAll( m1 |
     self.source.definition._in->exists(m2 | m1=m2 ))
```

"in" is a keyword of OCL; therefore the attribute "in" needs to be accessed via "_in"

OCL more details

See http://www.omg.org/spec/OCL/2.0/PDF/ for more details on OCL.

- If an attribute or association has cardinality less or equal 1 the reference to that attribute or association always returns a single value of the respecitive type (null, if it does not exist)
- If the cardinality is greater 1, the reference to it returns a set (collection) of the respective type
- There are operations to select elements from sets and to quantify on sets.

->iterate(x; res = init | exp(x,res))

- These set operations are accessed via ->

Validation frameworks

- OCL has a precisely defined meaning (independently from a specific programming language or implementation of the model)

- The way to ”hook in” OCL constraints depends on the used technology
(e.g. EMF Validation Framework)

For an example, see Sect. 4.5.1.4 of the ePNK manual.

OCL Opinion

- OCL looks and feels much like programming with a flavour of logic
- Programmers are not so used to it, and often get OCL wrong

- In most modelling frameworks, it is possible to formulate constraints in your favourite programming language

We will see an example in tutorial 4