SINTEF

F3
9. Feb. 2006

Applied Metamodelling

MDA Distilled

SINTEF

Applied Metamodelling

- *Language-Driven Development*
- *Metamodelling*
- *A Metamodelling Facility*
- *Abstract Syntax*

- Concrete Syntax
- Semantics
- Executable Metamodelling
- Mappings

- Reuse
- Cases 1 (Aspects), 2 (Telecom), 3 (XAction)

3

Different related approaches

- Language Engineering (Xactium, XMF)
- MDA – Model Driven Architecture (OMG)
- DSL – Domain Specific Languages (Microsoft) *(på slutten av kurset)*

4

Language Engineering

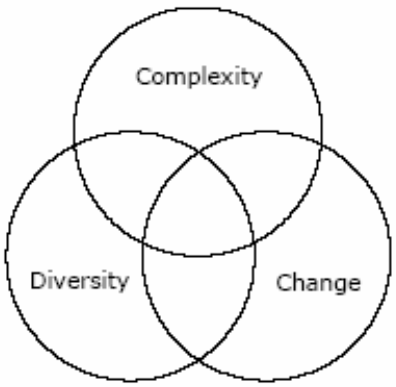
SINTEF

Language-driven development is fundamentally based on the ability to rapidly design new languages and tools in a unified and interoperable manner. We argue that existing technologies do not provide this capability, but a language engineering approach based on *metamodelling* can. The detailed study of metamodelling and how it can realise the Language-Driven Development vision will form the focus for the remainder of this book.

5

Challenges Facing Developers

SINTEF



6

Language-Driven Development – Providing the Solution

SINTEF

- *Execution:* allows the model or program to be tested, run and deployed;
- *Analysis:* provides information of the properties of models and programs;
- *Testing:* support for both generating test cases and validating them must be provided;
- *Visualisation:* many languages have a graphical syntax, and support must be provided for this via the user interface to the language;
- *Parsing:* if a language has a textual syntax, a means must be provided for reading in expressions written in the language;
- *Translation:* languages don't exist in isolation. They are typically connected together whether it is done informally or automatically through code generation or compilation;
- *Integration:* it is often useful to be able to integrate features from one model or program into another, e.g. through the use of configuration management.

7

From Model-Driven to Language-Driven Development

SINTEF

- the term *model* suggests a focus on high-level abstractions and *modelling* languages, with other artefacts seen as of lesser value. We feel that languages itself are the truly central abstractions, and that modelling languages form an undoubtedly useful yet partial subset of the spectrum of useful languages in system development. Consequently, all language artefacts, not just models, have a crucial role to play in the process;
- the prominent model-driven approach, MDA, is limited in its scope of application, compared to the full potential of Language-Driven Development (see section 1.3.2.

8

Language Engineering vs MDA

SINTEF

- whilst the MDA vision is grand, the technology for implementing it is very vaguely specified. So weak in fact that any modelling tool which has some simple code generation facility can (and in most cases does) claim to implement MDA. MDA is more useful as a marketing tool than anything else;
- MDA is too fixed on the notion of *platform*. What constitutes a *platform* is unclear at best - the transition from the most abstract model of a system to the most refined model may include several stages of models, each which could considered Platform Specific when compared to the previous stage, or Platform Independent when compared to the following stage. In any case, PIM to PSM mappings are just one of a whole spectrum of potential applications of Language-Driven Development;
- MDA is built on a weak inflexible architecture. This will be discussed in the context of metamodelling in section 2.8.

9

Language Engineering and Metamodelling

SINTEF

In order to be able to engineer languages, we need a language for capturing, describing and manipulating all aspects of languages in a unified and semantically rich way. This language is called a metamodelling language. Metamodels (models of languages) are the primary means by which language engineering artefacts are expressed, and are therefore the foundation for Language-Driven Development. While we have motivated Language-Driven Development in this chapter, the rest of the book will explore how metamodelling (the process of creating metamodels) can realise the Language-Driven Development vision.

10

Metamodelling

- A controversial topic, and one that is currently critical within the UML/OMG/MDA community.
- A metamodel is just another model (e.g., written in UML).
- Metamodels are examples of *domain-specific models.*
  - Other example domains: real-time systems, safety critical systems, e-business.
- The domain of metamodelling is *language definition*.
- Thus, a metamodel is a *model of some part of a language*.
  - Which part depends on how the metamodel is to be used.
  - Parts: syntax, semantics, views/diagrams, ...

11

Uses for a Metamodel

- For defining the syntax and semantics of a language.
- To explain the language.
- To compare languages rigorously.
- To specify requirements for a tool for the language.
- To specify a language to be used in a meta-tool (e.g., XMF).
- To enable interchange between tools.

12

Language Design

- How would you go about designing a programming language?
  1. What sort of programs do you want to allow programmers to create? (ie., user requirements).
  2. Define a syntax (eg., EBNF).
  3. Define semantics using structural induction over the constructs of the language, e.g., what do *while, if, ;, etc* all mean?
  4. Implement a compiler and libraries.
  5. Implement supporting tools.
  6. Build your killer app.
- In doing so, you would follow well-known principles of programming language design.

13

Programming Language Design

- The primary purpose of a programming language (PL) is to help a programmer to write programs.
  - ie., language design is *not* an exercise in and of itself.
  - if the language gets in the way, then it's not a good one.
- Other requirements, e.g., portability, stability, existing popularity, sponsorship by powerful organizations, should not be dominant factors.

14

User Requirements for a PL

1. A PL should give assistance in expressing what a program should accomplish and how it should execute.
2. A PL will encourage and assist in producing self-documenting code.
   - To find out what a program does, you (ideally) will be able to look in one place.
3. A PL will give assistance in finding errors.

15

Principles of Programming Language Design

- Simplicity is absolutely necessary.
  - Otherwise how will the designer know the consequences of their design decisions?
  - Pursue this to the extreme!
- Security.
- Fast generation of efficient code.
- Readability.
- Clear syntax to enable identification of syntax errors.
- Suitable structures for solving relevant problems.
- Proof rules for features of the language.
- Use patterns from other languages.
- Uniqueness.

16

Features of Languages

- Concrete syntax
- Abstract syntax
- Semantics

17

Concrete syntax - textual

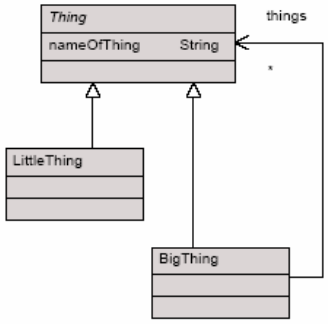
A textual syntax enables models or programs to be described in a structured textual form. A textual syntax can take many forms, but typically consists of a mixture of declarations, which declare specific objects and variables to be available, and expressions, which state properties relating to the declared objects and variables. The following Java code illustrates a textual syntax that includes a class with a local attribute declaration and a method with a return expression:

```
public abstract class Thing
  {
  private String nameOfThing;
  public String getName()
    {return nameOfThing;}
  }
```

18

Concrete syntax - visual

A visual syntax presents a model or program in a diagrammatical form. A visual syntax consists of a number of graphical icons that represent views on an underlying model. A good example of a visual syntax is a class diagram, which provides graphical icons for class models. As shown in Figure 2.1 it is particularly good at presenting an overview of the relationships and concepts in a model:



19

Abstract syntax

The *abstract syntax* of a language describes the vocabulary of concepts provided by the language and how they may be combined to create models. It consists of a definition of the concepts, the relationships that exist between concepts and well-formedness rules

20

Semantics

An abstract syntax conveys little information about what the concepts in a language actually mean. Therefore, additional information is needed in order to capture the semantics of a language. Defining a semantics for a language is important in order to be clear about what the language represents and means. Otherwise, assumptions may be made about the language that lead to its incorrect use. For instance, although we may have an intuitive understanding of what is meant by a state machine, it is likely that the detailed semantics of the language will be open to misinterpretation if they are not defined precisely. What exactly is a state? What does it mean for transition to occur? What happens if two transitions leave the same state. Which will be chosen? All these questions should be captured by the semantics of the language.

21

Modelling Languages vs. Programming Languages

A strong distinction has traditionally been made between modelling languages and programming languages (a fact reflected by the two distinct modelling and programming communities!). One reason for this is that modelling languages have been traditionally viewed as having an informal and abstract semantics whereas programming languages are significantly more concrete due to their need to be executable.

This is not the case in this book. Here, we view modelling languages and programming languages as being one and the same. Both have a concrete syntax, abstract syntax and semantics. If there is a difference, it is the level of abstraction that the languages are targeted at. For instance, UML tends to focus on specification whilst Java emphasises implementation. However, even this distinction is blurred: Java has been widely extended with declarative features, such as assertions, whilst significant inroads have been made towards developing executable versions of UML.

22

What is a Metamodel?

In its broadest sense, a metamodel is a model of a modelling language. The term "meta" means transcending or above, emphasising the fact that a metamodel describes a modelling language at a higher level of abstraction than the modelling language itself.

In order to understand what a metamodel is, it is useful to understand the difference between a metamodel and a model. Whilst a metamodel is also a model (as defined in chapter 1), a metamodel has two main distinguishing characteristics. Firstly, it must capture the essential features and properties of the language that is being modelled. Thus, a metamodel should be capable of describing a language's concrete syntax, abstract syntax and semantics. Note, how we do this is the major topic of the rest of this book!

23

Why Metamodel?

- System development is fundamentally based on the use of languages to capture and relate different aspects of the problem domain.
- The benefit of metamodelling is its ability to describe these languages in a unified way. This means that the languages can be uniformly managed and manipulated thus tackling the problem of language diversity. For instance, mappings can be constructed between any number of languages provided that they are described in the same metamodelling language.
- Another benefit is the ability to define semantically rich languages that abstract from implementation specific technologies and focus on the problem domain at hand. Using metamodels, many different abstractions can be defined and combined to create new languages that are specifically tailored for a particular application domain. Productivity is greatly improved as a result.

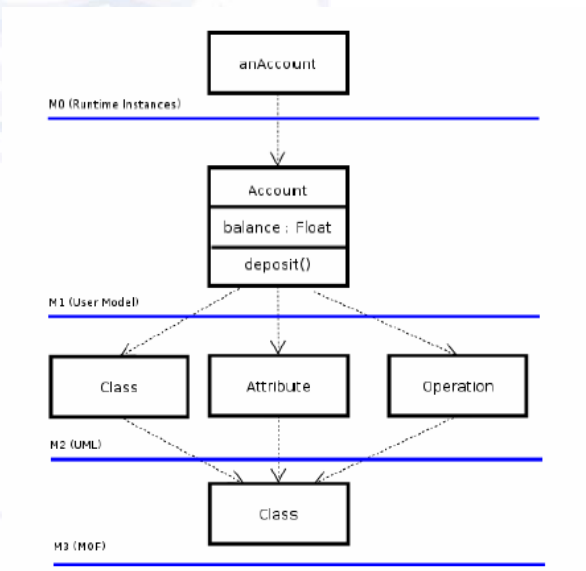
24

Traditional Metamodel Architecture

- The traditional metamodel architecture, proposed by the original OMG MOF 1.X standards is based on 4 distinct meta-levels. These are as follows:
- **M0** contains the data of the application (for example, the instances populating an objectoriented system at run time, or rows in relational database tables).
- **M1** contains the application: the classes of an object-oriented system, or the table definitions of a relational database. This is the level at which application modeling takes place (the type or model level).
- **M2** contains the metamodel that captures the language: for example, UML elements such as Class, Attribute, and Operation. This is the level at which tools operate (the metamodel or architectural level).
- **M3** The meta-metamodel that describes the properties of all metamodels can exhibit. This is the level at which modeling languages and operate, providing for interchange between tools.

25

Traditional Metamodel Architecture



26

Golden Braid Metamodel Architecture

- Although the 4-layer metamodel is widely cited, its use of numbering can be confusing. An alterative architecture is the golden braid architecture [Hof79]. This architecture emphasises the fact that metamodels, models and instances are all relative concepts based on the fundamental property of instantiation.
- The idea was first developed in LOOPS (the early Lisp Object Oriented Programming System, and then became a feature of both ObjVLisp [Coi87] and also CLOS (the Common Lisp Object System). – *First in Smalltalk (1977 ?)* Underpinning the golden braid architecture is the relationship between a Class and an Object. A Class can be instantiated to create an Object. An Object is said to be an instance of a Class. This fact can be determined through a distinct operation, of(), that returns the Class that the Object was created from. In addition, a Class is also a subclass of Object. This means that a Class can also be instantiated by another Class: its meta Class. This relationship is key to th metaarchitecture, as it enables an arbitrary number of meta-levels to be described through the instantiation relationship.

27

The Metamodelling Process

- However, there is a clearly defined process to constructing metamodels, which does at least make the task a well-defined, if iterative, process. The process has the following basic steps:
- defining abstract syntax
- defining well-formedness rules and meta-operations
- defining concrete syntax
- defining semantics
- constructing mappings to other languages

28

Metamodel quality: Five levels of Metamodelling

- **Level 1** This is the lowest level. A simple abstract syntax model must be defined, which has not been checked in a tool. The semantics of the language it defines will be informal and incomplete and there will be few, if any, well-formed rules.
- **Level 2** At this level, the abstract syntax model will be relatively complete. A significant number of well-formedness rules will have been defined, and some or all of the model will have been checked in a tool. Snapshots of the abstract syntax model will have been constructed and used to validate its correctness. The semantics will still be informally defined. However,

29

Metamodel quality: Five levels of Metamodelling

- **Level 3** The abstract syntax model will be completely tried and tested. Concrete syntax will have been defined for the language, but will only have been partially formalised. Typically, the concrete syntax will be described in terms of informal examples of the concrete syntax, as opposed to a precise concrete syntax model. Some consideration will have been given to the extensibility of the language architecture, but it will not be formalised or tested.
- **Level 4** At level 4, the concrete syntax of the language will have been formalised and tested. Users will be able to create models either visually and textually and check that they result in a valid instance of the abstract syntax model. The language architecture will have been refactored to facilitate reuse and extensibility. Models of semantics will have begun to appear.

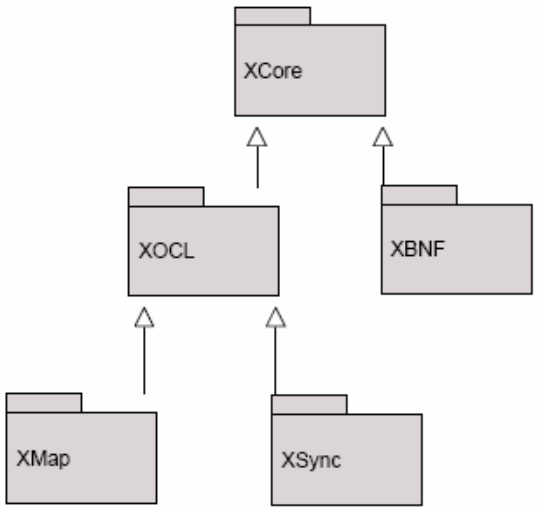
30

A Metamodelling Facility

- XMF (eXecutable Metamodelling Facility)
- XMF:Mosaic tool
- www.xactium.com

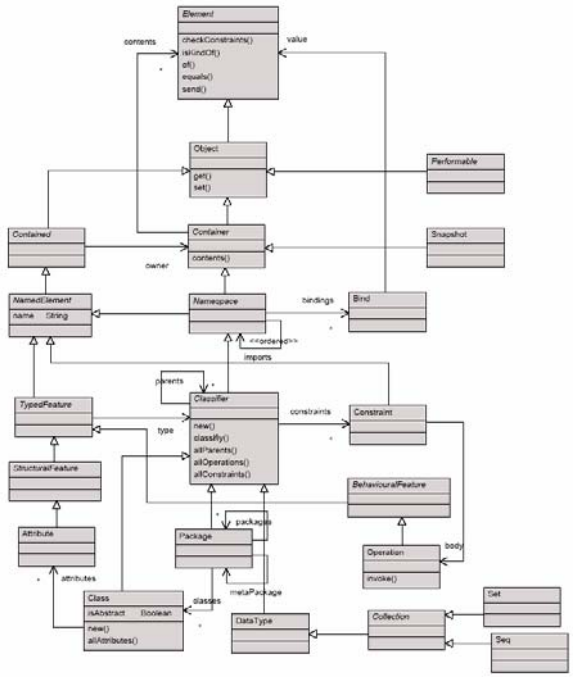
31

Overview of XMF architecture

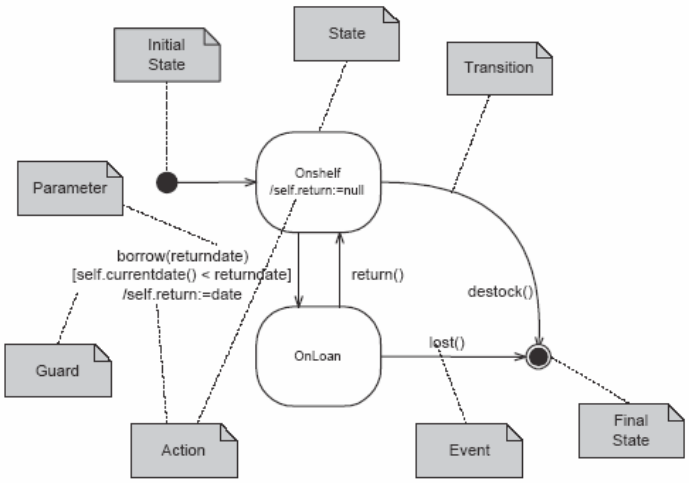


32

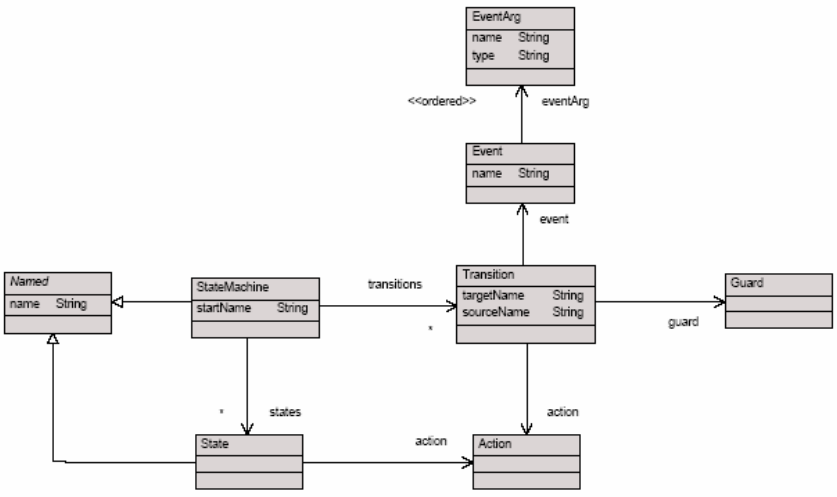
XCore Metamodel



State Machine example



State Machine abstract syntax metamodel



35

Well formedness rules

Firstly, it must be the case that all states have a unique names:

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

36

Operation addState

```
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
```

37

XMF Lexical concrete syntax

- 1 @Package StateMachines
- 2 @Class isAbstract Named
- 3 @Attribute name : String end
- 4 end
- 5 @Class StateMachine extends Named
- 6 @Attribute startName **:** String **end**
- 7 @Attribute states **:** Set(State) **end**
- 8 @Attribute transitions **:** Set(Transition) **end**
- 9 **end**
- 10 @Class State extends Named
- 11 **end**
- 12 @Class Transition
- 13 @Attribute sourceName **:** String **end**
- 14 @Attribute targetName **:** String **end**
- 15 **end**
- 16 **end**

38

XOCL & Object Constraint Language - OCL

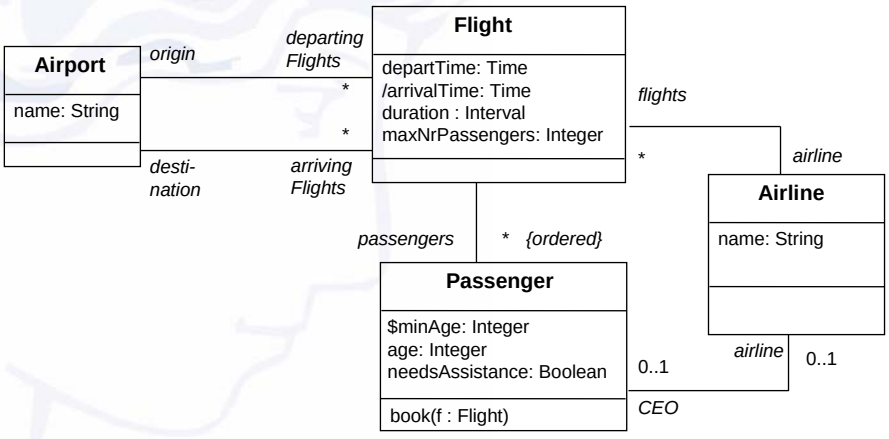
- OCL er del av UML
- *XOCL utvider OCL med action primitiver*
- Tekstlig språk for å beskrive beskrankninger
- Predikatlogikk gjort folkelig (ingen ∀∃∧∨⇒∴)
- Constraints
  - begrensninger på modellene
  - multiplisitet, etc er begrensinger!
    - ønsker ytterligere begrensninger
- Brukes i definisjonen av UMLs metamodell
  - må jo være presis!
- Formelt
  - entydig
  - ingen side-effekter (XOCL har side-effekter)

referanse materiale

se også XOCL i XMF:Mosaic BlueBook

39

Example model



40

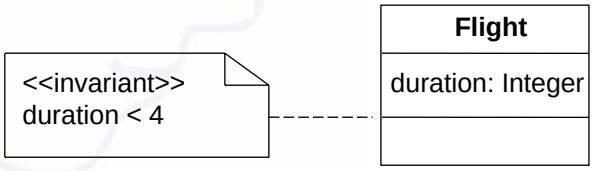
Constraint context and self

- Every OCL expression is bound to a specific context.
- The context may be denoted within the expression using the keyword ‘self’.

41

Notation

- Constraints may be denoted within the UML model or in a separate document.
  - the expression:
    context Flight inv: self.duration < 4
  - is identical to:
    context Flight inv: duration < 4
  - is identical to:

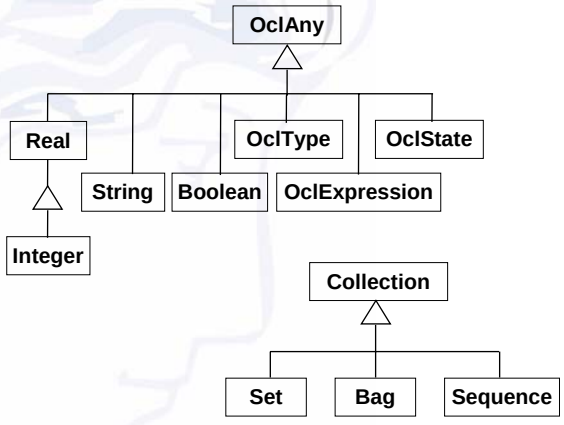


42

Elements of an OCL expression

- In an OCL expression these elements may be used:
  - basic types: String, Boolean, Integer, Real.
  - classifiers from the UML model and their features
    - attributes, and class attributes
    - query operations, and class query operations
  - associations from the UML model

OCL types



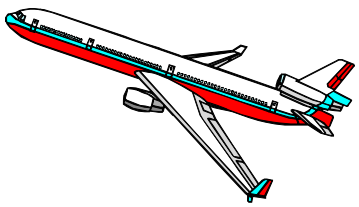
Example: OCL basic types

context Airline inv:
name.toLower = ‘klm’

context Passenger inv:
age >= ((9.6 - 3.5)* 3.1).abs implies
mature = true

Model classes and attributes

- “Normal” attributes
  context Flight inv:
  self.maxNrPassengers <= 1000
- Class attributes
  context Passenger inv:
  age >= Passenger.minAge



Example: query operations

context Flight inv:
self.departTime.difference(self.arrivalTime)
.equals(self.duration)

Time
$midnight: Time month : String day : Integer year : Integer hour : Integer minute : Integer
difference(t:Time):Interval before(t: Time): Boolean plus(d : Interval) : Time

Interval
nrOfDays : Integer nrOfHours : Integer nrOfMinutes : Integer
equals(i:Interval):Boolean $Interval(d, h, m : Integer) : Interval

47

Example: navigations

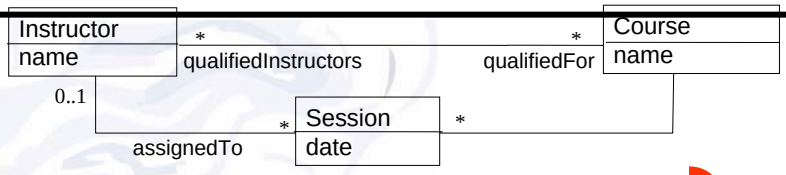
- Navigations

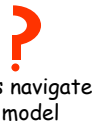
context Flight
inv: origin <> destination
inv: origin.name = ‘Amsterdam’

context Flight
inv: airline.name = ‘KLM’

48

Basic “Navigation” expressions

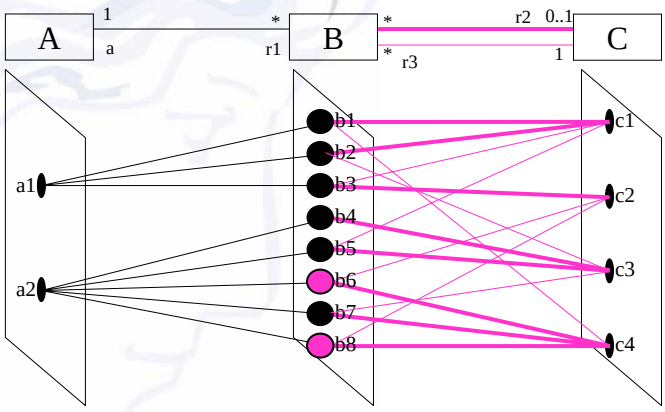




- i: Instructor, c: Course, s: Session
  - The name of the course:
    - c.name
  - The date of the session:
    - s.date
  - The instructor assigned to the session:
    - s.instructor
  - The course of the session:
    - s.course
  - The name of the course of the session:
    - s.course.name
  - The instructors qualified for the session:
    - s.course.qualifiedInstructors

49

Navigation Example

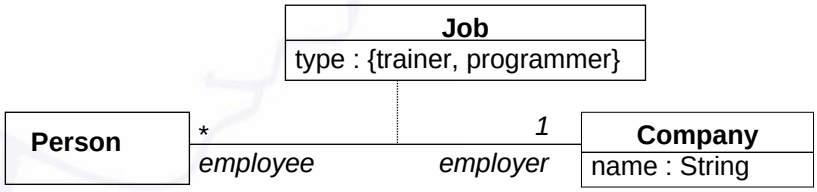


- What does **a1.r1.r2.r3** yield?
- Assuming the B's have a boolean attribute “*black*”; black=false for b6, b8 - what expression refers from **a2** to the set { **b1** }

50

Association classes

```
context Person inv:
if employer.name = 'Klasse Objecten' then
   job.type = #trainer
else
   job.type = #programmer
endif
```



51

Three subtypes to Collection

- Set:
  - arrivingFlights(from the context Airport)
- Bag:
  - arrivingFlights.duration (from the context Airport)
- Sequence:
  - passengers (from the context Flight)

52

Collection operations

- OCL has a great number of predefined operations on the collections types.

- Syntax:

  collection->operation

53

The collect operation

- Syntax:
  collection->collect(elem : T | expr)
  collection->collect(elem | expr)
  collection->collect(expr)
- Shorthand:
  collection.expr

- The *collect* operation results in the collection of the values resulting evaluating *expr* for all elements in the *collection*

54

The select operation

- Syntax:
  collection->select(elem : T | expression)
  collection->select(elem | expression)
  collection->select(expression)
- The *select* operation results in the subset of all elements for which *expression* is true

55

The forAll operation

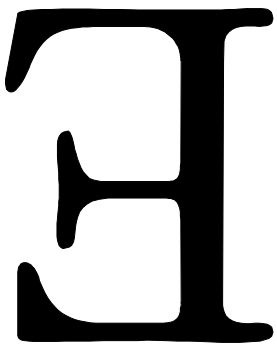
- Syntax:
  collection->forAll(elem : T | expr)
  collection->forAll(elem | expr)
  collection->forAll(expr)
- The *forAll* operation results in true if *expr* is true for all elements of the collection

56

The exists operation

- Syntax:
  collection->exists(elem : T | expr)
  collection->exists(elem | expr)
  collection->exists(expr)

- The *exists* operation results in true if there is at least one element in the collection for which the expression *expr* is true.



57

Example: exists operation

context Airport inv:
self.departingFlights ->
exists(departTime.hour < 6)

58

Other collection operations

- *isEmpty*: true if collection has no elements
- *notEmpty*: true if collection has at least one element
- *size*: number of elements in collection
- *count(elem)*: number of occurences of elem in collection
- *includes(elem)*: true if elem is in collection
- *excludes(elem)*: true if elem is not in collection
- *includesAll(coll)*: true if all elements of coll are in collection

59

Iterate example

- Example iterate:

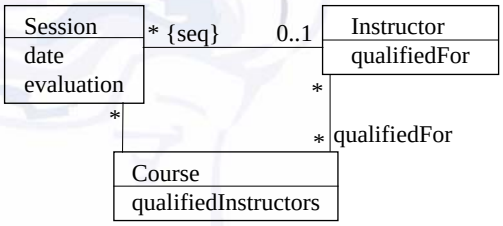
```
context Airline inv:
flights->select(maxNrPassengers > 150)->notEmpty
```

- Is identical to:

```
context Airline inv:
flights->iterate(f : Flight; answer : Set(Flight) = Set{ } |
if f.maxNrPassengers > 150 then
  answer->including(f)
else answer endif )->notEmpty
```

60

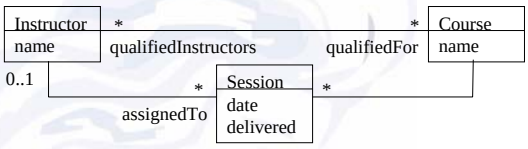
OCL — Navigation Details



- An association end with cardinality maximum > 1 yields a set or sequence
  - anInstructor.Session yields a sequence
  - anInstructor.qualifiedFor yields a set
- An association end with cardinality maximum of 1 yields an object or a set (with zero or one elements)
  - aSession.Instructor yields an object
  - aSession.Instructor->isEmpty yields a Boolean

61

Collections use



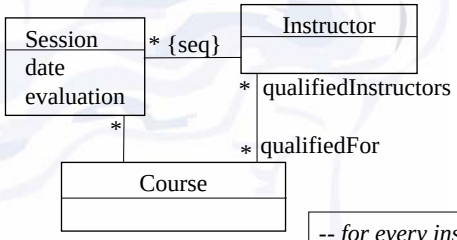
?

Let's navigate on a model

- i: Instructor
- The courses an instructor is qualified to teach
  - Course.allInstances ->**select** (c | c.qualifiedInstructors ->**includes** (i))
- Sessions delivered by an instructor who is no longer qualified to teach it
  - Session.allInstances ->**select** (s | s.delivered **and** s.course.qualifiedInstructors ->**excludes** (s.instructor))
- The last can be simplified significantly with "convenience" attributes
  - Session.allInstances **->select** (s | s.teacherNotQualified)

62

Another Invariant Formalized

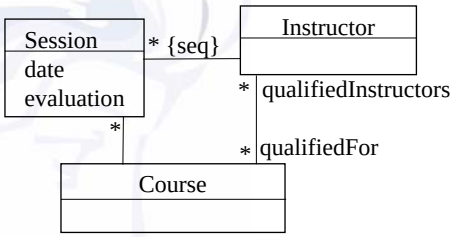



```
-- for every instructor ...
Instructor::invariant
-- for any course
Course.allInstances->forAll ( c |
-- if the evaluation bad
  Session->select(Course=c)->forAll(s |
          s.evaluation = bad
  -- then instructor is disqualified for course
  implies qualifiedFor ->excludes (c)) )
```

Always combine formal and narrative descriptions

63

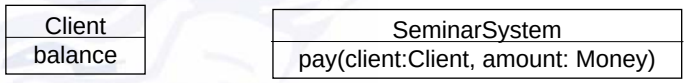
Same Invariant on Course




```
-- for every course ...
Course::invariant
-- for all sessions
Session->forAll ( s |
-- if the evaluation is bad
    s.evaluation = bad implies
-- then the instructor is not a qualified instructor
qualifiedInstructors->excludes(s.Instructor) )
```

64

Operation Specification



operation SeminarSystem::pay (**in** client:Client, **out** amount: Money)
-- *When you pay off an invoice*
pre -- *Provided the payment amount is not negative and does not*
-- *exceed amount owed*
client .balance >= amount **and** amount>=0
post -- *The balance is reduced by the amount of the payment*
client.balance@**pre** = client.balance + amount

65

let, new: Convenient Names, New Objects

Any specification can introduce local names using **let** … **in** …

operation SeminarSystem::scheduleCourse
(client: Client, date: Date, course: Course)
let (availableInstructors =
instructors ->**select** (qualifiedFor(course) **and** availableOn(date)) **)**
in (-- *the name "availableInstructors" can be used in pre or post*
pre availableInstructors ->**notEmpty**
post -- *some instructor from available instructors is assigned …*
)

Actions often result in the creation of a **new** object

let (s = Session.**new) in (** -- *s is a new member of Session type*
s.client = client **and** s.date = date **and** s.course = course
and ….
)

66

OCL — Misc.

- Special words
  - @**pre** designates a value at the start of an operation
    total = total@**pre** + amount
  - **self** designates the object itself
    self.total = **self**.total@**pre** + amount
  - **result** designates the returned object (if any)
    result = total
- Comments
  - -- Two hyphens start a comment that goes through the end of line

67

OCL Tools

- Cybernetics
  - ww.cybernetic.org
- University of Dresden
  - www-st.inf.tu-dresden.de/ocl/
- Boldsoft
  - www.boldsoft.com
- ICON computing
  - www.iconcomp.com
- Royal Dutch Navy
- Others … …

68

Conclusions and Tips

SINTEF

- OCL invariants allow you to
  - model more precisely
  - stay implementation independent
- OCL pre- and postconditions allow you to
  - specify contracts (design by contract)
  - precisely specify interfaces of components
- OCL usage tips
  - keep constraints simple
  - always combine natural language with OCL
  - use a tool to check your OCL

69

UML og OCL

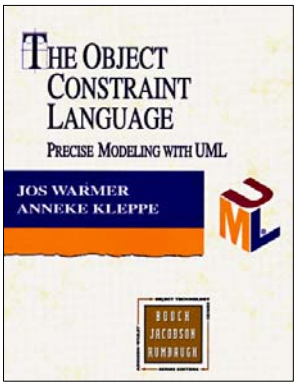
SINTEF

- Skriver OCL som tilleggsdokumentasjon til modeller
- Skriver OCL i Constraints
- (Verktøy)problem: hvordan bruke aktivt
  - forfining
  - konsistens
  - kodegenerering

70

Further resources on OCL

- The Object Constraint Language
  - ISBN 0-201-37940-6
- OCL home page
  - www.klasse.nl/ocl/index.htm



OclAny

x,y:OclAny; T is a OclType	
x = y	x and y are the same object
x < > y	not (x=y)
x.oclIsNew	True if x is a new instance
x.oclType	The type of x
x.isKindOf(T)	True if T is a supertype (transitive) of the type of x
x.isTypeOf(T)	True if T is equal to the type of x
x.asType(T)	Results in x, but of type T.

OclType and operators

T is a OclType	
T.new	Create a new instance of type T
T.allInstances	All of the instances of type T

- Logical operators in Boolean expressions
  - and, or, xor, not, implies

Collection (1)

c,c2 : Collection(T); x,e:T; P:T→ Boolean;
f, f2: T → Object

c->size	Number of elements
c->sum	Sum of elements (elements must support addition)
c->count(e)	Number of times e is in c
c->isEmpty	c->size = 0
c->notEmpty	not c->isEmpty

Collection (2)

c->includes(e)	True if e is in c
c->includesAll(c2)	True if c2 in c
c->excludes(e)	True if e not in c
c->exludesAll(c2)	True if none in c2 is in c
c->exists(P)	True if an e makes P true
c->forAll(P)	True if P true for all e in c
c->isUnique(f)	True if f evaluates to different value for all e in c
c->sortedBy(f)	Sequence sorted by f
c->iterate(x;e=f;f2)	Iterate x over c and apply f2, initialise e to f

75

Collection subtypes (1)

Applies to set and bag
set, bag: Collection; e,x:T; P: T→Boolean;
f, f2: T→Object

set->union(set2)
set->union(bag)
set = set2
set->intersection(set2)
set->intersection(bag)
set – set2
set->including(e)

76

Collection subtypes (2)

set->excluding(e)	
set->symmetricDifference(set2)	The set of elements in set or set2, but not in both
set->select(x|P)	All elements for which P is valid
set->select(P)	Same as set->select(self|P)
set->reject(x|P)	Same as set->select(x|not P)
set->reject(P)	Same as set->select(self|not P)
set->collect(x|f)	The bag of elements which results from applying f to every member of set
set->asSequence	
set->asBag	

77

Sequence

seq->append(e)	seq followed by e
seq->prepend(e)	e followed by seq
seq->subSequence(lower, upper)	Subsequence in range [lower, upper]
seq->at(i)	Element at position i
seq->first	seq->at(1)
seq->last	seq->at(seq->size)

78

Example

- PIM4SOA Metamodel
- Web Services Metamodel
- BDI Agent Metamodel
- Transformation Rules
- Application example

79

Oblig oppgave 1

- a) Bli kjent med metamodellering gjennom bruk av XMF:Mosaic, XMF Walktrhrough i Help-delen (BlueBook) i verktøyet. Domain model in the component modelling domain,
- b) Lag en metamodell for Komponent-modellering, I Eclipse EMF, evt. UML profil.
- c) Bruk ATL for å gjøre en mapping og transformasjon fra komponent modell til Java modell
- d) Bruk MoFScript for å gjøre en mapping og transformasjon fra Java modell til Java kode

80

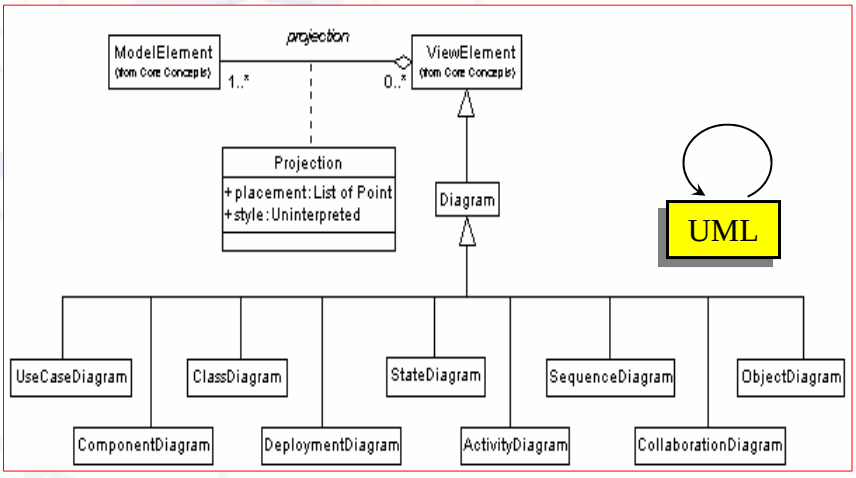
SINTEF

OMG MOF and Metamodeling

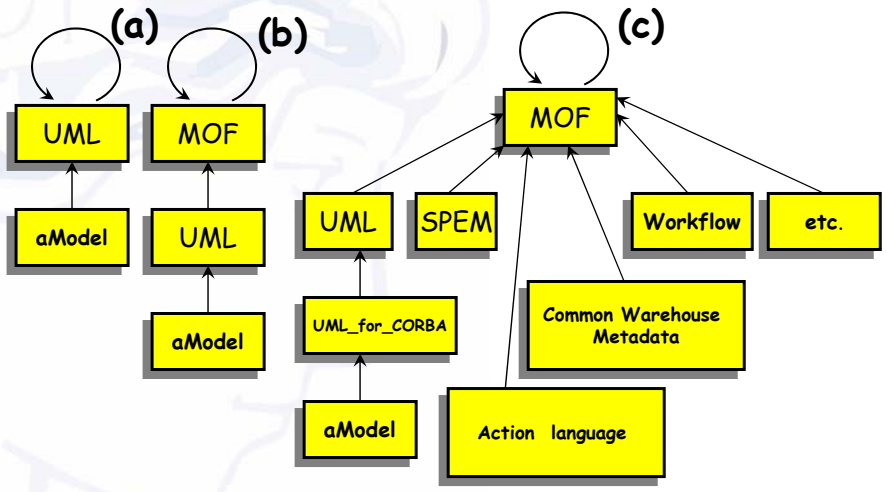
SINTEF

Fragments of a UML meta-model

1.0

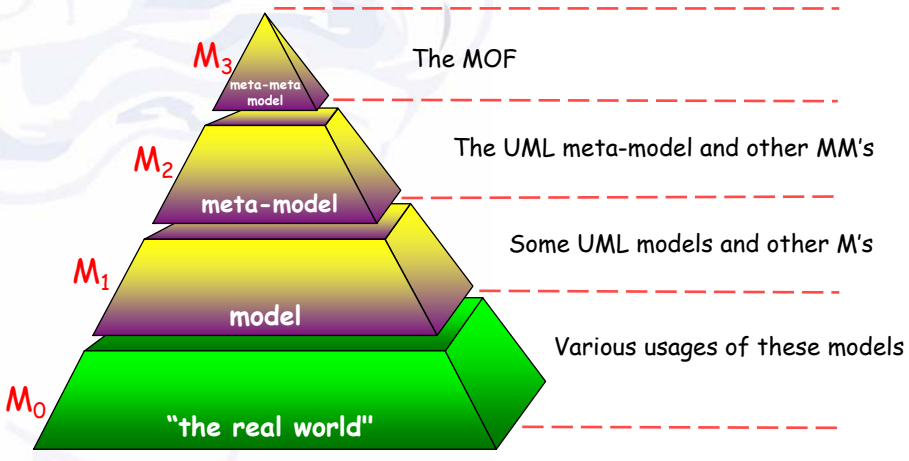


Three stages in the evolution of modeling techniques at the OMG.



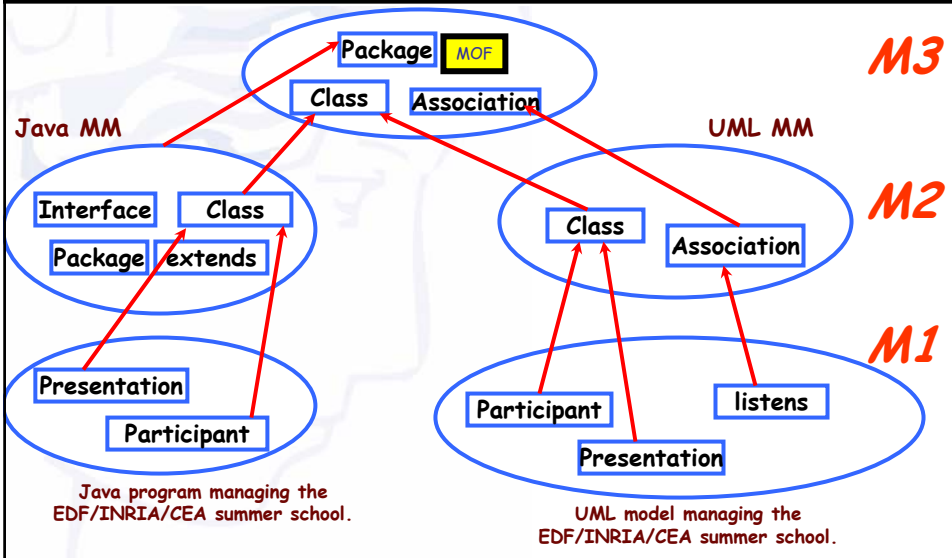
83

Egyptian architecture



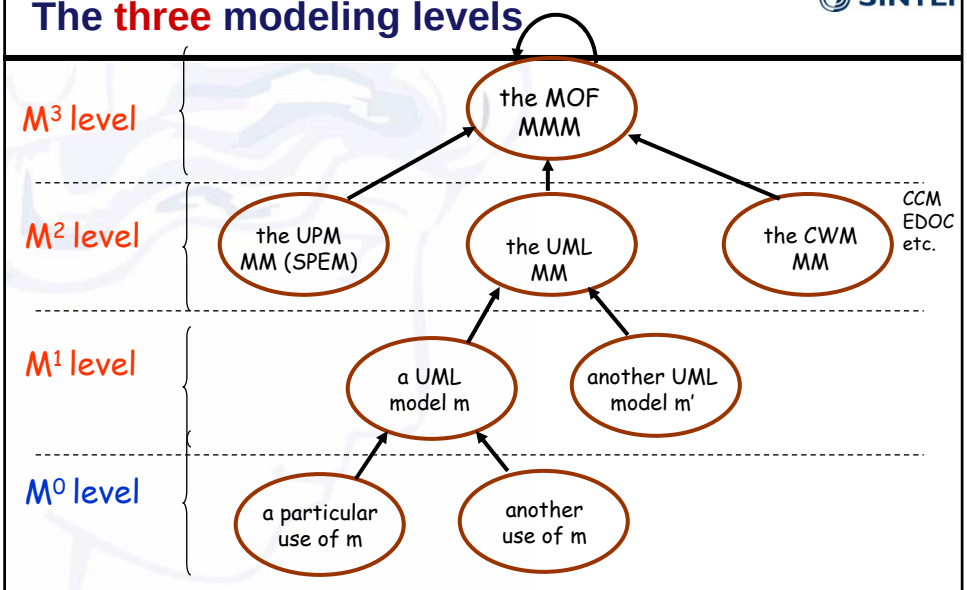
84

Illustration.



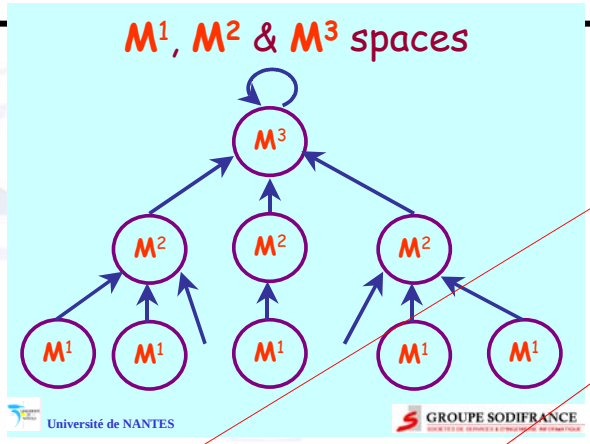
85

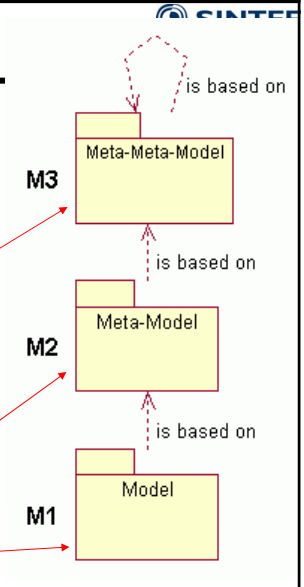
The three modeling levels



86

The MDA meta-model stack

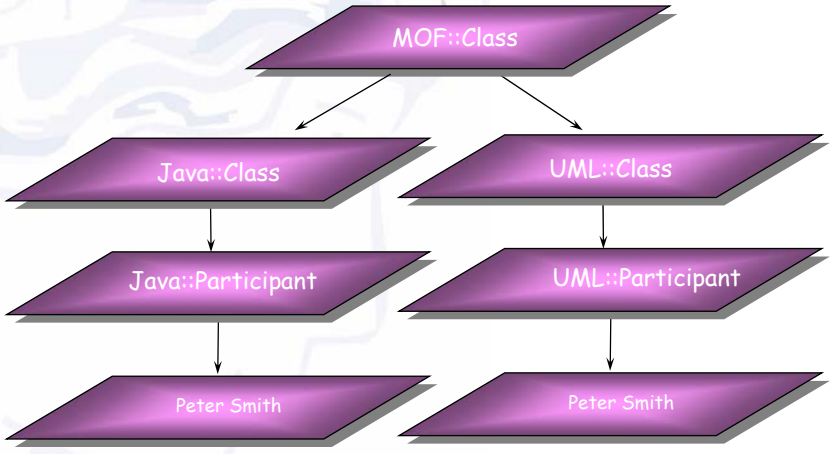




- One unique Meta-Meta-model (the MOF)
- An important library of compatible Meta-models
- Each of the models is defined in the language of its unique meta-model

87

Multiple meta-models

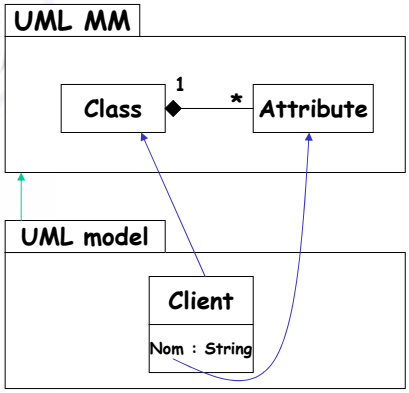


88

Model -> Meta-model

entity → meta-entity relationship

model → meta-model relationship



See Richard Lemesle's thesis document for a MOF presentation:

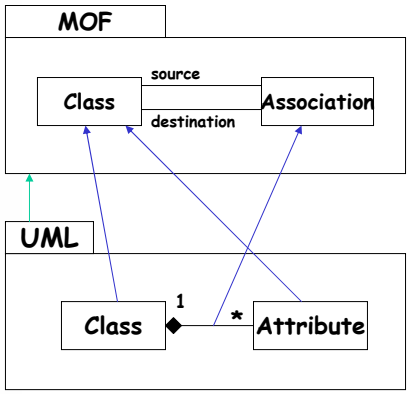
http://www.sciences.univ-nantes.fr/info/lrsg/Pages_perso/RL/Richard_2001.htm

89

Meta-model -> Meta-meta-model

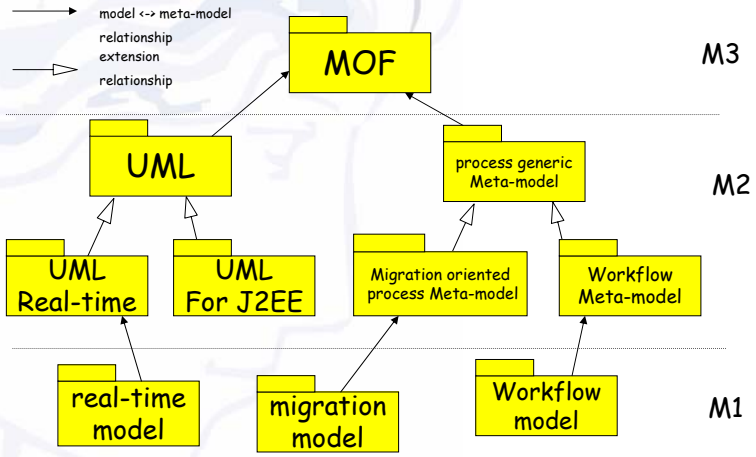
entity → meta-entity relationship

model → meta-model relationship



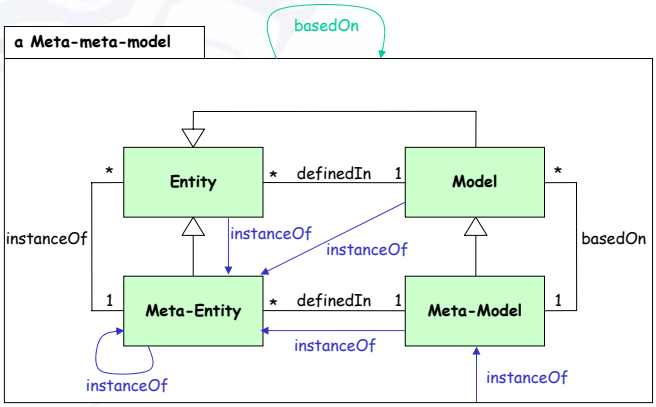
90

Meta-models and profiles



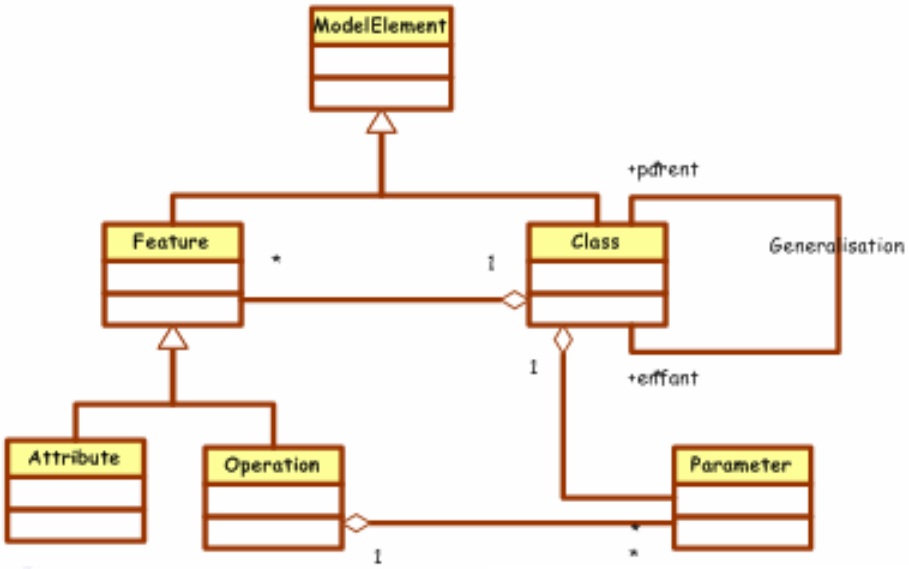
91

A meta-meta-model



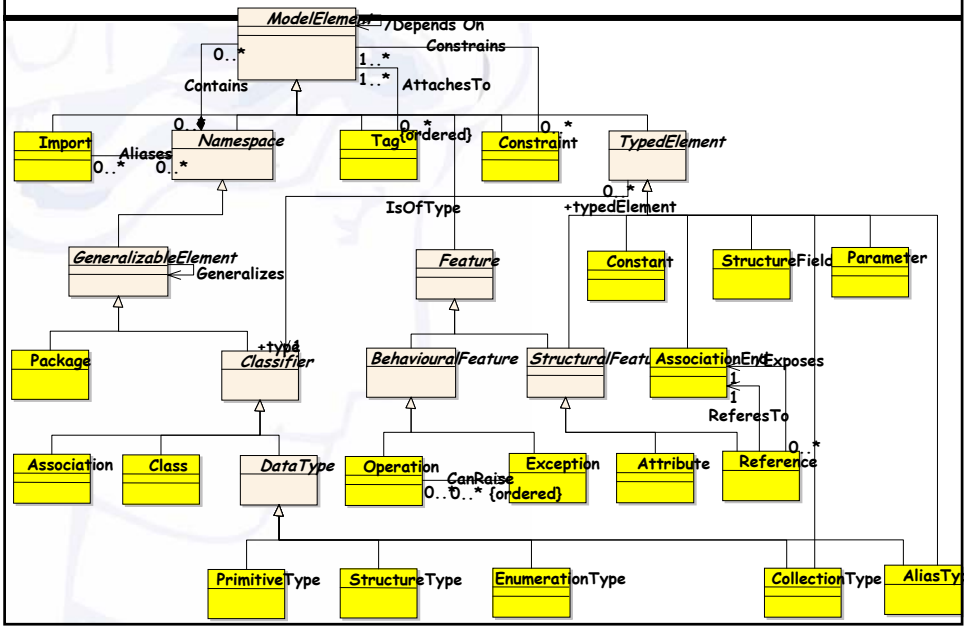
92

A meta-model fragment



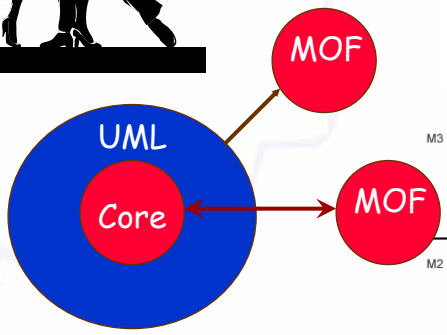
93

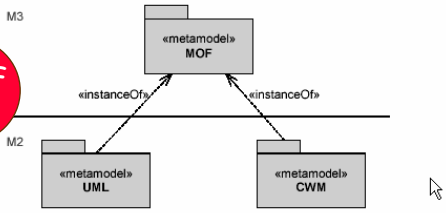
MOF Model (M3)



The complex relations between UML 2.0 and MOF 2.0

- An odd couple.
- Definition synchronization.
- UML is a drawing tool of the MOF.
- Multiple skills transfers.
- ...





95

General organization