UMR IRISA

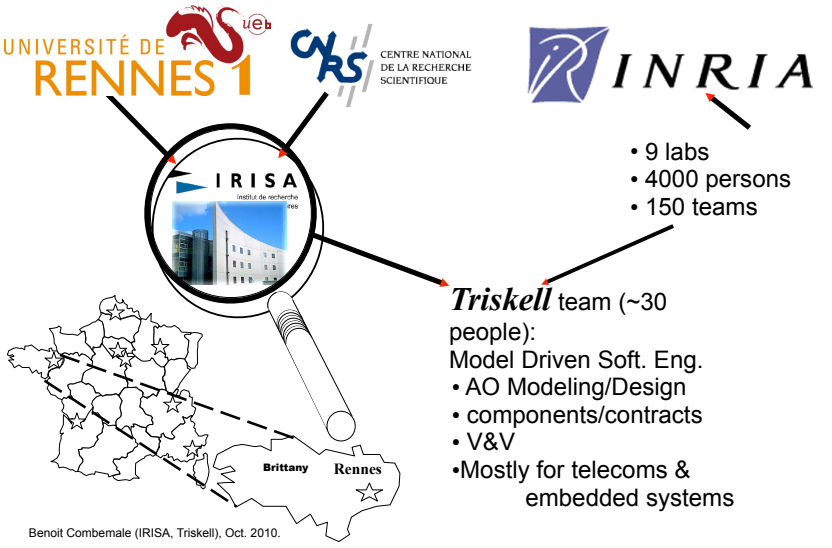
Advanced Features of Software Engineering: from Modeling to Metamodeling...

Dr. Benoit Combemale
benoit.combemale@irisa.fr
http://www.irisa.fr/triskell/perso_pro/combemale
With the help (and the slides) of Jean-Marc Jézéquel

INSA de Rennes, 2010-2011

UNIVERSITÉ DE RENNES 1 — INRIA — esir ECOLE SUPERIEURE D'INGENIEUR DE RENNES

Triskell (http://www.irisa.fr/triskell)



Triskell Presentation

- **Triskell team:** a world leader(*) of Model Driven Engineering (MDE) research, from Embedded Systems to Service Oriented Architectures.
  - 9 INRIA or IRISA Members (lead: Jean-Marc Jézéquel)
  - 6 Post-doc & Research Engineers
  - 15 PhD Student
- **New Results:**
  - Contract-based and Aspect Oriented Modeling
  - Model-Driven Engineering : model execution, model composition, model typing...
  - Model-Based Testing
  - Model transformation testing
  - ➡ Main software: Kermeta, a **Ker**nel **meta**modeling language
  - ➡ Others software: Kompose, Cartier, smartadapter@runtime, Sintaks, etc.

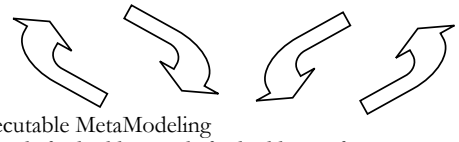
(*) refering e.g.; to the number of publications in recent years in the reference conference of the domain MODELS.

Scientific objectives of Triskell:
Combine formal methods & established practice

- Contracts and Aspects: from Requirements to Design to Runtime
  - Requirements by Contract, Analysis and Design by Contract (inc. QoS)
  - Aspect-Oriented Requirements & Design
  - Integration of IoT &IoS

Feed-back loop

- Model-Based Testing
  - requirement based testing, requirement validation
  - design based testing, design validation



- Executable MetaModeling
  - *tools for building tools for building software*
  - *Thanks to **Kermeta** this is not just meta-bla-bla*
    - It works for real! Have a look at http://www.kermeta.org/

Outline

- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up

Outline

- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up

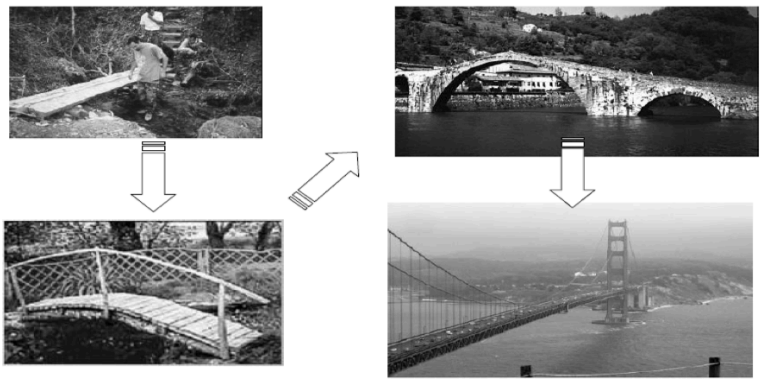
Introduction to Model-Driven Engineering

(or: Why I'd like write program that write programs rather than write programs)

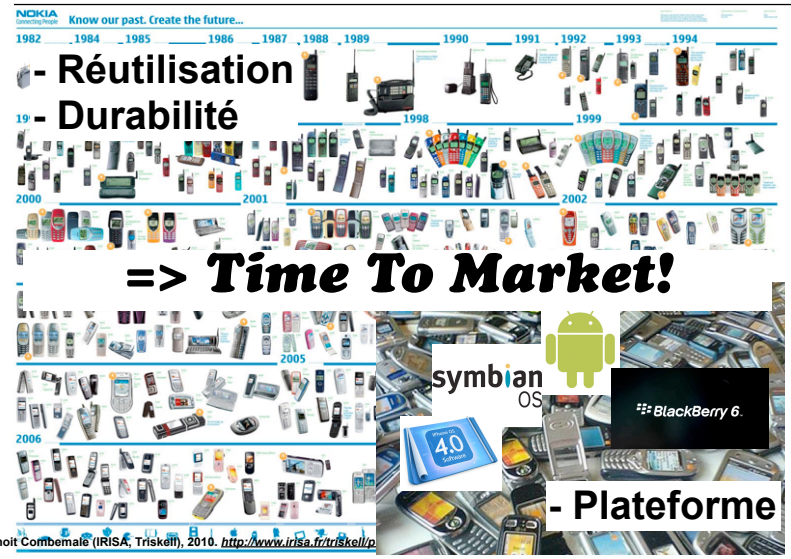
Problématique du génie logiciel aujourd'hui

Complexité des systèmes modernes

esir ECOLE SUPERIEURE D'INGENIEUR DE RENNES



Complexité des systèmes modernes



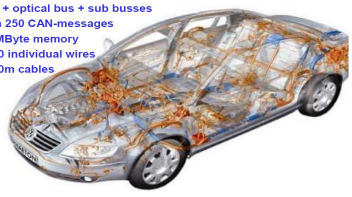
Benoit Combemale (IRISA, Triskell), 2010. http://www.irisa.fr/triskell/p

Complexité des systèmes modernes

- Critique
- Temps-réel
- Embarqué

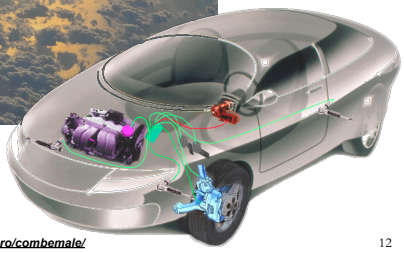
Phaeton

- 61 networked ECUs
- 3 bus systems + optical bus + sub busses
- 2500 signals in 250 CAN-messages
- more than 50 MByte memory
- more than 2000 individual wires
- more than 3800m cables



Software Complexity

- Critical
- Real-time
- Embedded



rso_pro/combemale/

Complexité des systèmes modernes

- Google :
  - 300 000 serveurs
    - répartis dans une vingtaine de datacenters.
    - répondre à plus d'1 milliard de requêtes par jour,
      - *chacune interrogeant 8 milliards de pages Web*
      - *en moins d'un cinquième de seconde*
  - Building for Scale:
    - 6,000 developer / 1,500+ projects
    - Each product has custom release cycles
      - *few days to few weeks*
    - 1(!!) code repository
    - No binary releases
      - *everything builds from HEAD*
    - 20+ code changes per minute
      - *50% of the code changes every month*

- Distribué
- Large-echelle

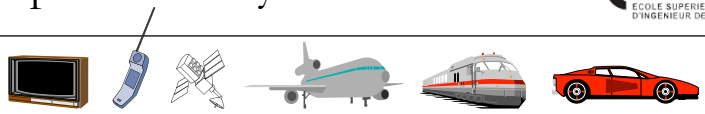
Google

Innovation Factory: Testing, Culture, & Infrastructure

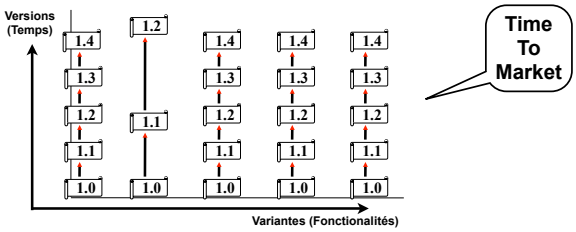
Patrick Copeland, Google
ICST 2010

Source: http://googletesting.blogspot.com/search/label/Copeland

Complexité des systèmes modernes



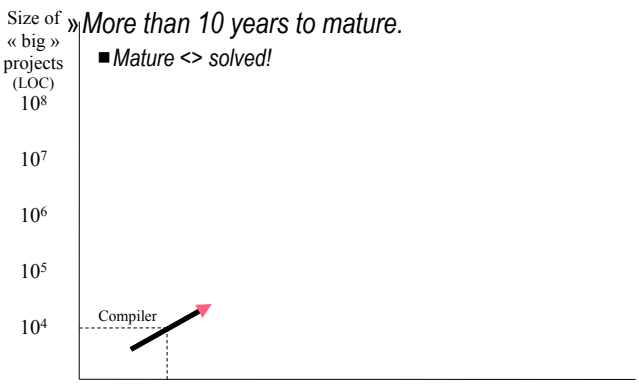
- Importance des aspects non fonctionnels
  - systèmes répartis, parallèles et asynchrones
  - qualité de service : fiabilité, latence, performances...
- Flexibilité accrue des aspects fonctionnels
  - notion de ***lignes de produits*** (espace, temps)



Benoit Combemale (IRISA, Triskell), 2010. *http://www.irisa.fr/triskell/perso_pro/combemale/*

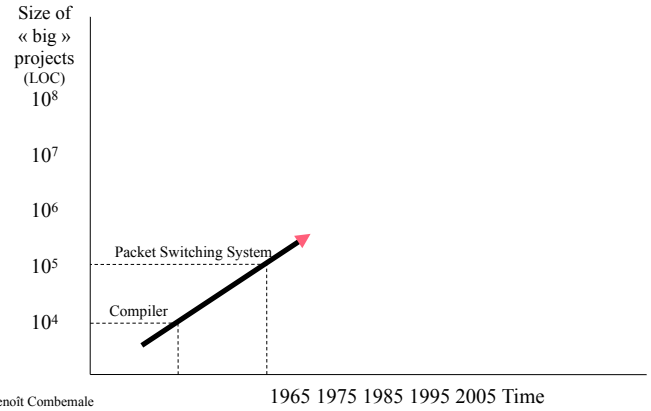
Problems addressed in SE

- 1960's: Cope with inherent complexity of software (Correctness)
  - Milestone: Floyd '*assigning meaning to programs*'
    - *More than 10 years to mature.*
      - *Mature <> solved!*



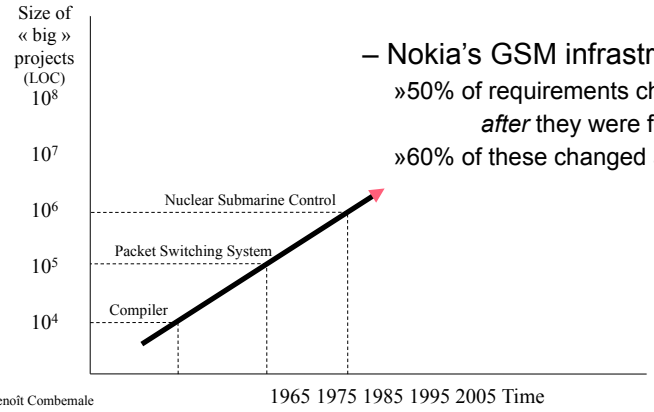
Problems addressed in SE

- 1970's: Cope with project size
  - Milestone: Parnas, Yourdon: *modularity & structure*
    - » *More than 10 years to mature*



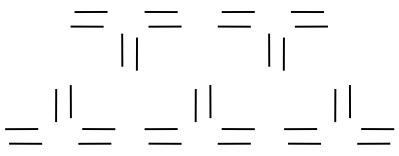
Problems addressed in SE

- 1980's: Cope with variability in requirements
  - Milestone: Jackson, Meyer: *modeling*, *object orientation*
    - » *More than 10 years to mature*



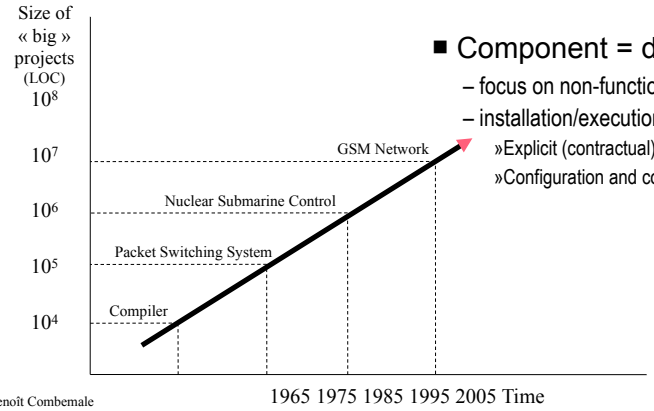
– Nokia's GSM infrastructure:
  - »50% of requirements changed *after* they were frozen
  - »60% of these changed at least twice!

OO approach: frameworks



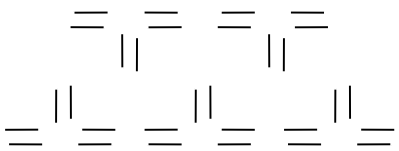
Problems addressed in SE

- 1990's: Cope with distributed systems and mass deployment:
  - Milestone: MS (COM), Szyperski: *product-lines & components*



- Component = deployment unit
  - focus on non-functional properties
  - installation/execution concept
    - »Explicit (contractual) dependencies
    - »Configuration and connection

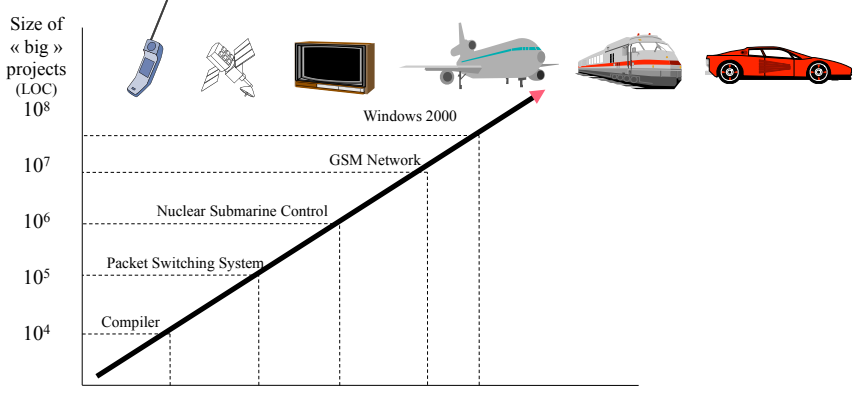
OO approach: Models and Components



- *frameworks*
  - Changeable software, from distributed/unconnected sources even after delivery, by the end user
  - Guarantees ?
    Functional , synchronization, performance, QoS

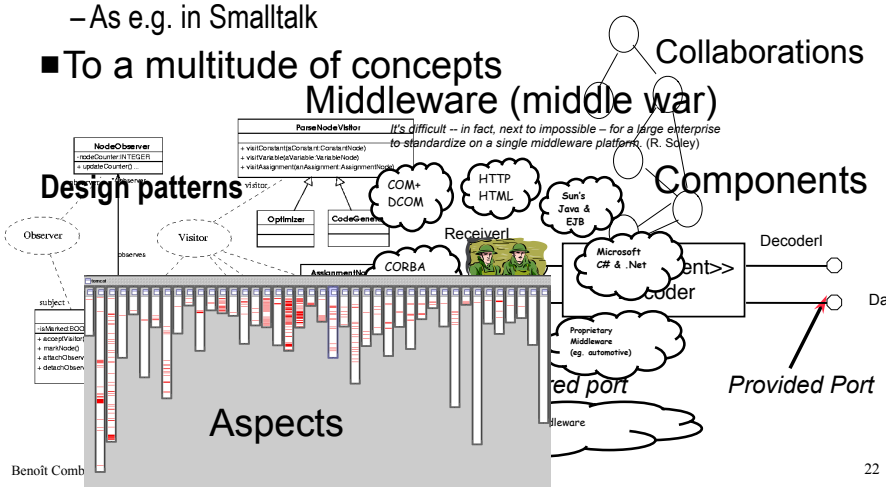
Problems addressed in SE

- 2000's: pervasive software integration, accelerating technological changes (platforms)
  - Milestone: ?



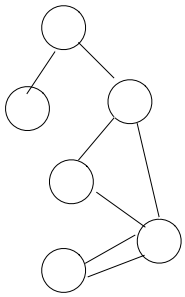
Once upon a time… software development looked simple

- From the object as the *only* one concept
  - As e.g. in Smalltalk
- To a multitude of concepts



Collaborations

- Objects should be as simple as possible
  - To enable modular understanding
- But then where is the complexity?
  - It is in the way objects interact!
  - Cf. Collaborations as a standalone diagram in UML (*T. Reenskaug's works)*



Design Patterns

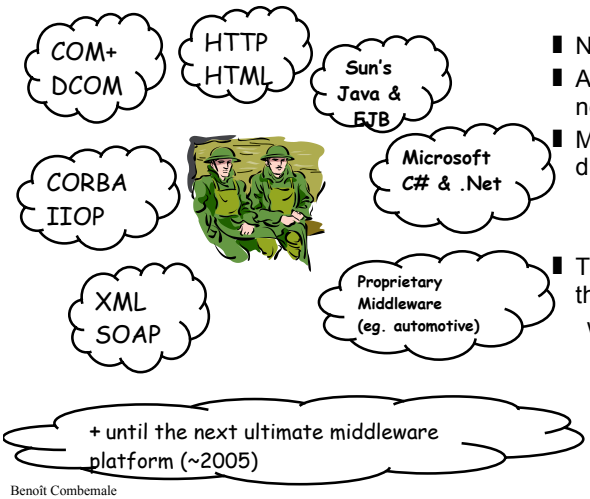
- Embody *architectural know-how* of experts
- As much about problems as about solutions
  - pairs problem/solution in a context
- About non-functional forces
  - reusability, portability, and extensibility…
- Not about classes & objects but *collaborations*
  - Actually, design pattern applications *are* parameterized collaborations

From Objects to Components

- Object = instance of a class
- Class = reusable unit of software
  - focus on structural and functional properties
  - development concept
- Component = deployment unit
  - focus on non-functional properties
  - installation/execution concept
    - » Explicit dependencies
    - » Configuration and connection

Middleware or Middle War?

It's difficult -- in fact, next to impossible – for a large enterprise to standardize on a single middleware platform. (R. Soley)



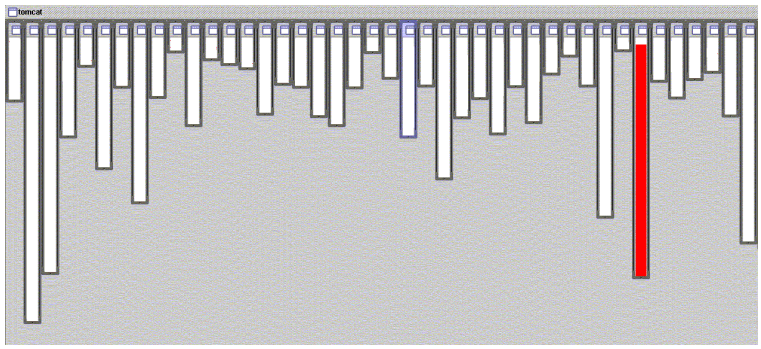
- No clear winner until now
- And probably not in the near future
- Migration is expensive and disruptive
- The OMG tried to send in the ''blue helmets'' with the MDA initiative

Aspect Oriented Programming

- Kiczales et al., ECOOP'97
  - MIT's one of 10 key technologies for 2010
- Encapsulation of cross-cutting concerns in OO programs
  - Persistence, contract checking, etc.
- Weaving at some specific points (join points) in the program execution
  - *Hence more than macros on steroids*
- AspectJ for AOP in Java
  - Some clumsiness in describing dynamic join points
- What about Aspect Oriented Design ?

good modularity

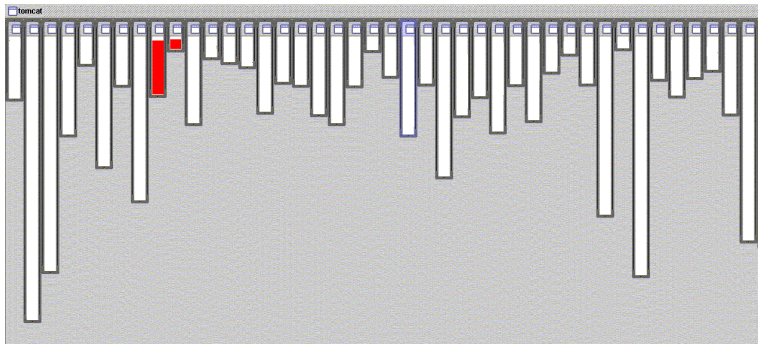
XML parsing



- XML parsing in org.apache.tomcat
  - red shows relevant lines of code
  - nicely fits in one box

good modularity

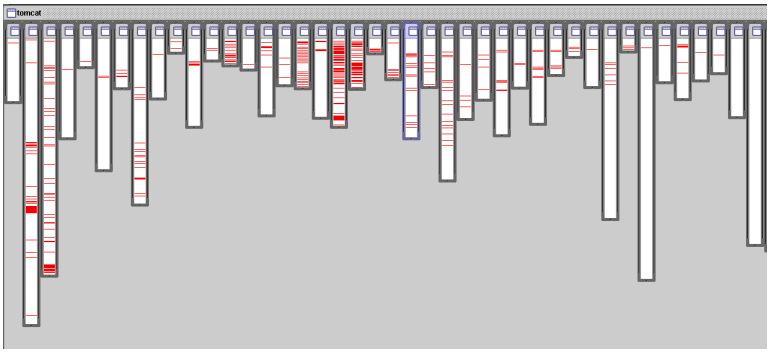
URL pattern matching



- URL pattern matching in org.apache.tomcat
  - red shows relevant lines of code
  - nicely fits in two boxes (using inheritance)

problems like...

logging is not modularized



- where is logging in org.apache.tomcat
  - red shows lines of code that handle logging
  - not in just one place
  - not even in a small number of places

AspectJ™ is…

- a small and well-integrated extension to Java
- a general-purpose AO language
  - just as Java is a general-purpose OO language
- freely available implementation
  - compiler is Open Source
- includes IDE support
  - Eclipse, JBuilder…
- user feedback is driving language design
  - users@aspectj.org
  - support@aspectj.org
- currently at 1.6.8 release

Expected benefits of using AOP

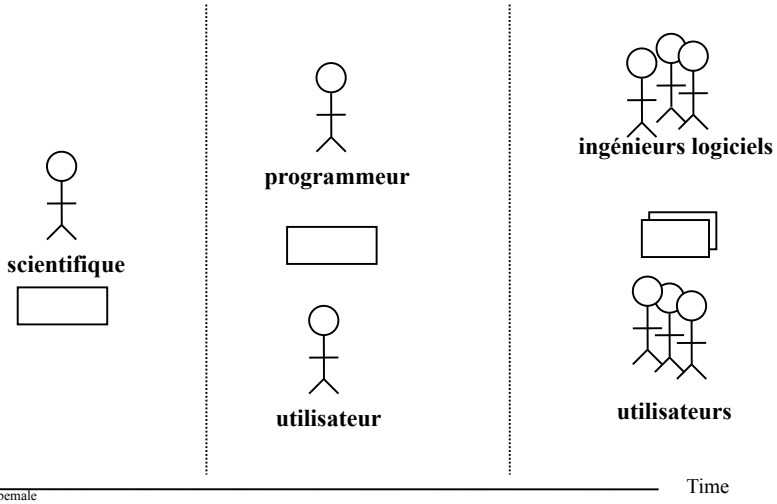
- good modularity,
  even for crosscutting concerns
  - less tangled code
  - more natural code
  - shorter code
  - easier maintenance and evolution
    - » easier to reason about, debug, change
  - more reusable
    - » library aspects
    - » plug and play aspects when appropriate

État de la pratique

Des logiciels complexes...

- Logiciels de grande taille
  - des millions de lignes de code
  - des équipes nombreuses
  - durée de vie importante
  - des lignes de produits
  - plateformes technologiques complexes
  - évolution continue
- Logiciels complexes
- Logiciels critiques
- ...

Évolution des acteurs



Oui, Oui, Oui, mais ...

- La mentalité de "l'informaticien moyen" a t elle radicalement évoluée ?
- Du "Codeur", au "Programmeur", à l' "Ingénieur Logiciel", ...
- L' "Ingénieur logiciel"
  - prouve-t-il ses programmes ?
  - maintient-il à jour la documentation, les spécifications ?

État de la pratique

50 ans après les débuts de l'informatique,

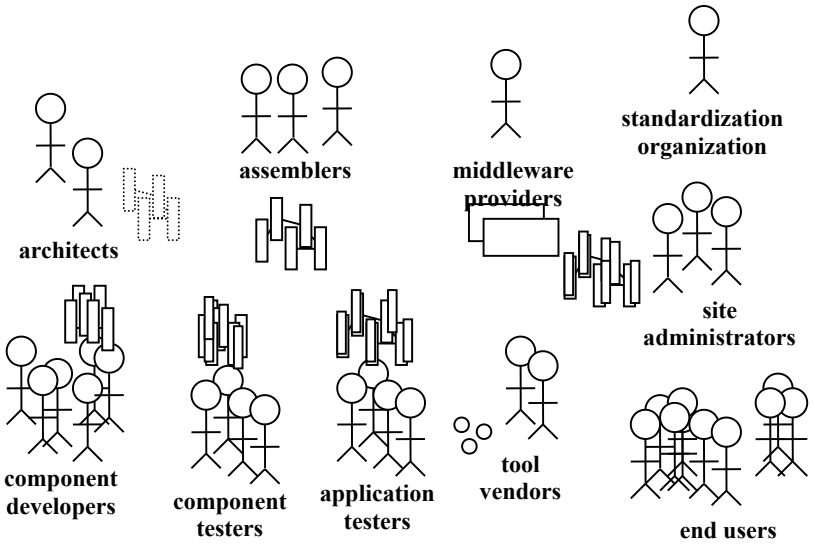
pour "l'informaticien moyen"

la seule chose importante dans le logiciel c'est ...

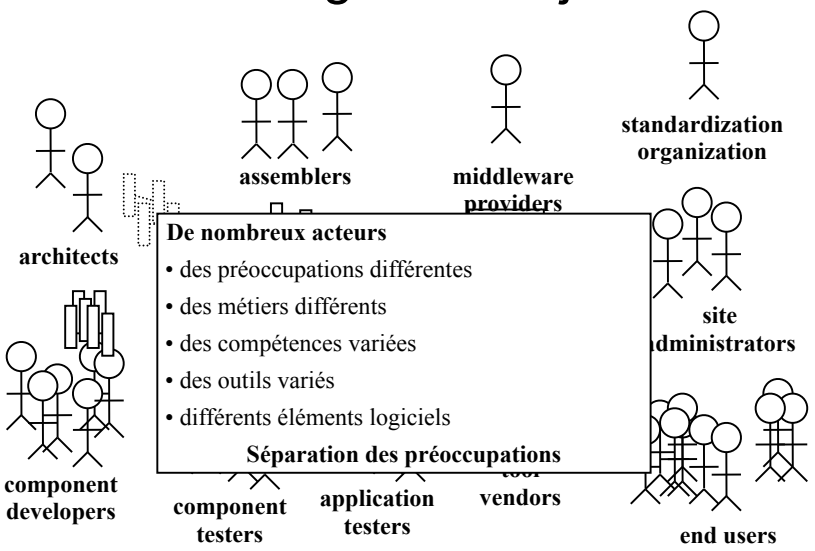
le Code !

=> Ingénierie Dirigée par le code

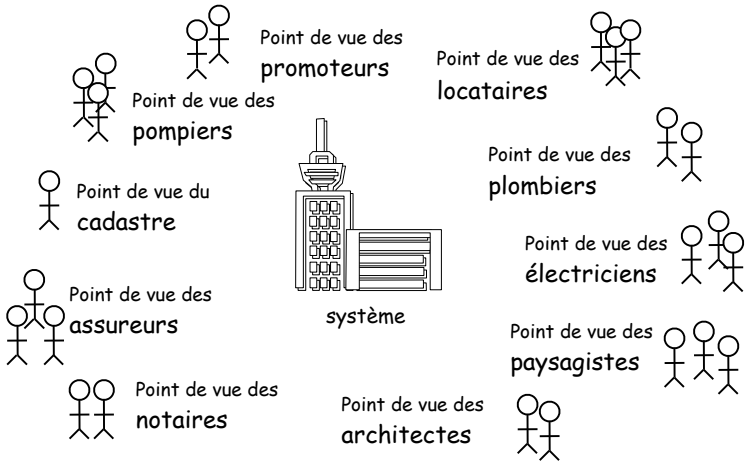
L'industrie logicielle aujourd'hui



L'industrie logicielle aujourd'hui

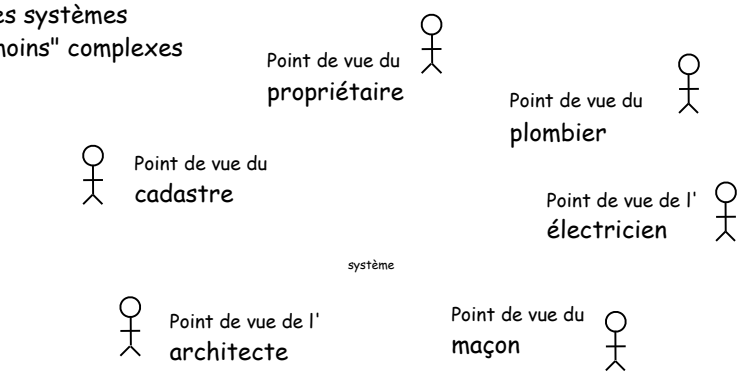


Séparations des préoccupations



Séparations des préoccupations

Utile même pour des systèmes "moins" complexes

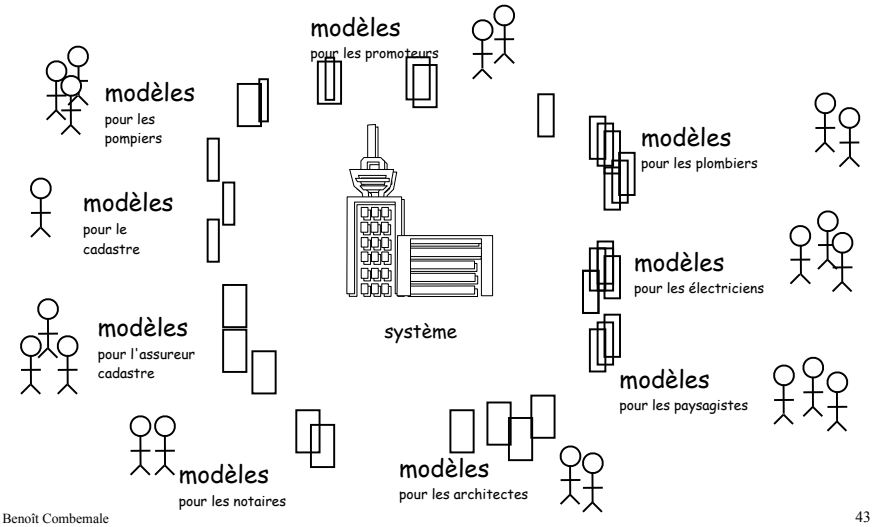


Problématique

- Complexité croissante des logiciels
- Séparations des préoccupations
- Séparations des métiers
- Multiplicité des besoins
- Multiplicité des plateformes
- Évolution permanente

Logiciel = Code ?
Est-ce la solution ?

Multiples modèles d'un même système

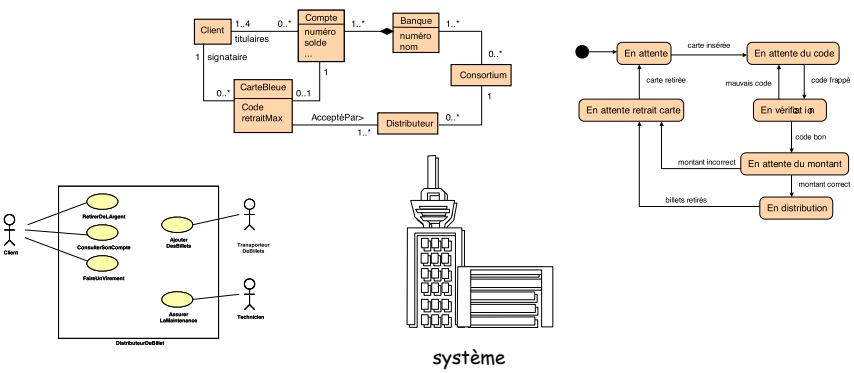


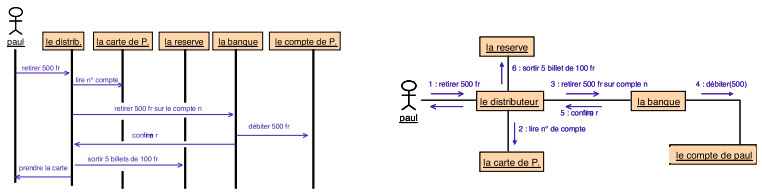
Ingénierie Dirigée par le **Code**

ou

Ingénierie Dirigée par les **Modèles** ?

Telle est la question...



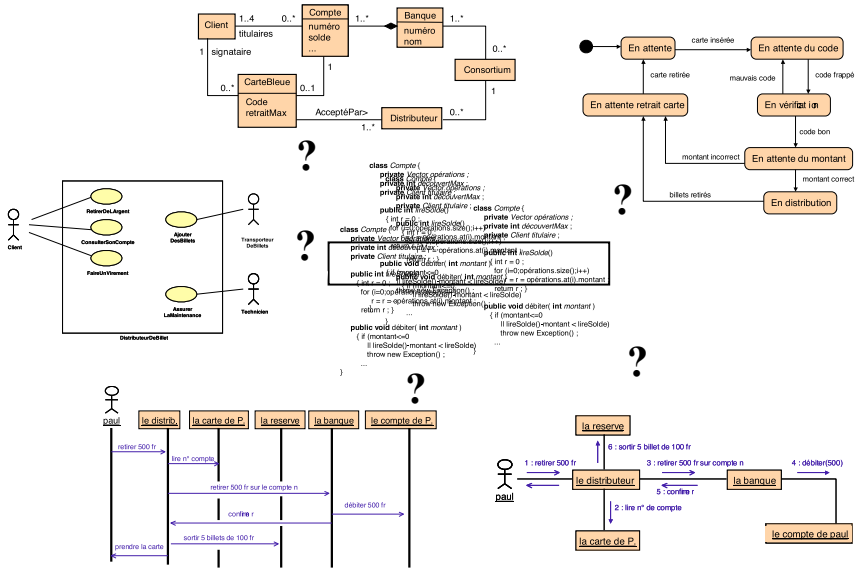


Oui, Oui, Oui, mais ...

pour "l'informaticien moyen"

la seule chose importante dans le logiciel c'est ...

le Code !



Synthèse

- Des Modèles plutôt que du Code
  - Un modèle est la simplification/abstraction de la réalité
  - Nous construisons donc des modèles afin de mieux comprendre les systèmes que nous développons
  - Nous modélisons des systèmes complexes parce que nous somme incapables de les comprendre dans leur totalité
  - Le code ne permet pas de simplifier/abstraire la réalité

Pourquoi modéliser

Why modeling: master complexity

- Modeling, in the broadest sense, is the *cost-effective use of something in place of something else for some cognitive purpose*. It allows us to use something that is *simpler*, *safer* or *cheaper* than reality instead of reality for some purpose.

- A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality.

Jeff Rothenberg.

Modeling in Science & Engineering

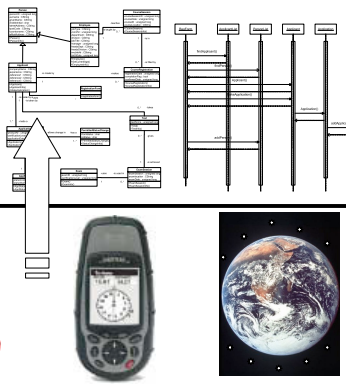
- A Model is a *simplified* representation of an *aspect* of the World for a specific *purpose*

Specificity of Engineering: Model something not yet existing (in order to build it)

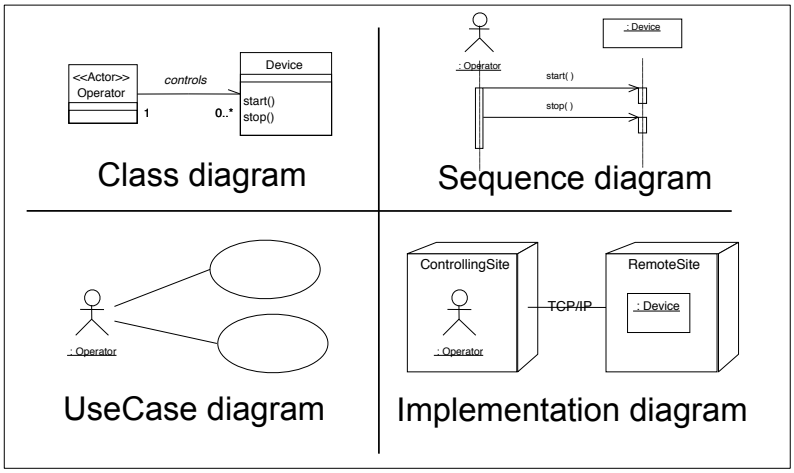
M_1 (modeling space)

Is represented by

M_0 (the world)



UML: one model, 4 main



The X diagrams of UML

- Modeling along 4 main viewpoints:
  - Static Aspect *(Who?)*
    - » Describes objects and their relationships
    - » Structuring with packages
  - User view *(What?)*
    - » Use cases
  - Dynamic Aspects *(When?)*
    - » Sequence Diagram
    - » Collaboration Diagram
    - » State Diagram
    - » Activity Diagram
  - Implementation Aspects *(Where?)*
    - » Component Diagram & deployment diagram

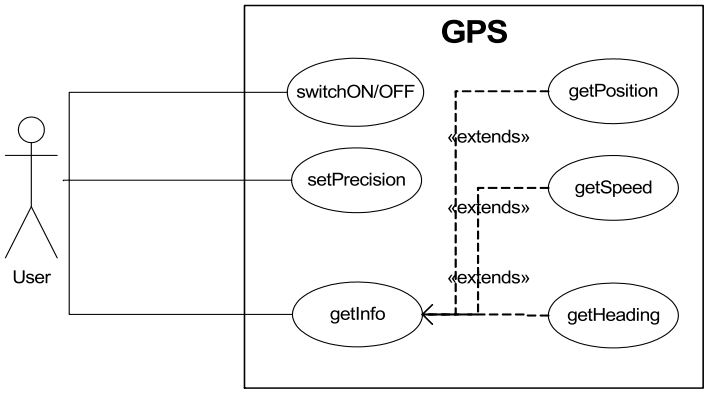
Example

- Modeling a (simplified) GPS device
  - Get position, heading and speed
    - » by receiving signals from a set of satellites
  - Notion of Estimated Position Error (EPE)
    - » Receive from more satellites to get EPE down
  - User may choose a trade-off between EPE & saving power
    - » Best effort mode
    - » Best route (adapt to speed/variations in heading)
    - » PowerSave

(Case Study borrowed from N. Plouzeau, K. Macedo & JP. Thibault. Big thanks to them)

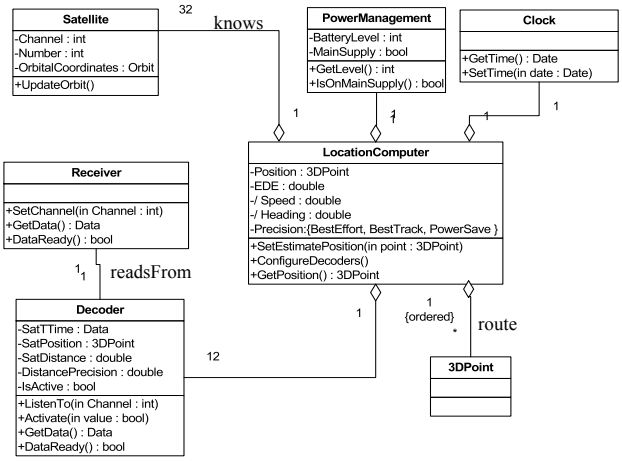
Modeling a (simplified) GPS device

- Use case diagram



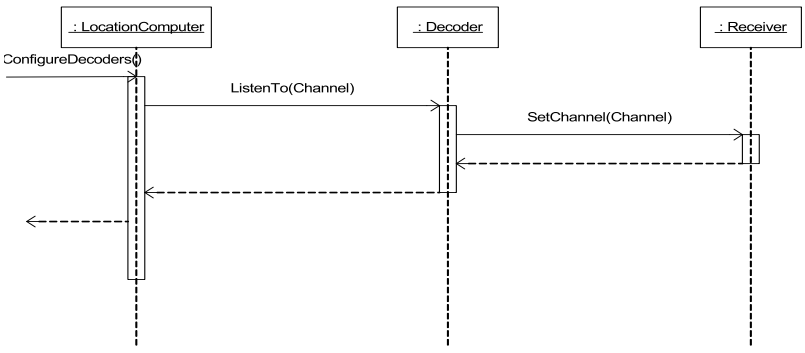
Modeling a (simplified) GPS device

- Class diagram



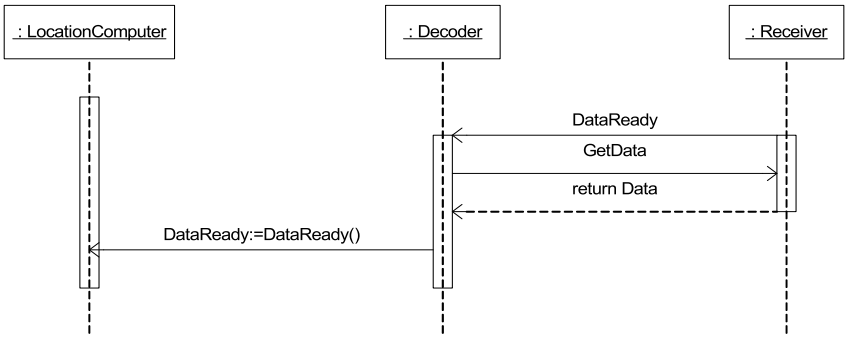
Modeling a (simplified) GPS device

- Sequence diagram: configuring decoders



Modeling a (simplified) GPS device

- Sequence diagram: interrupt driven architecture



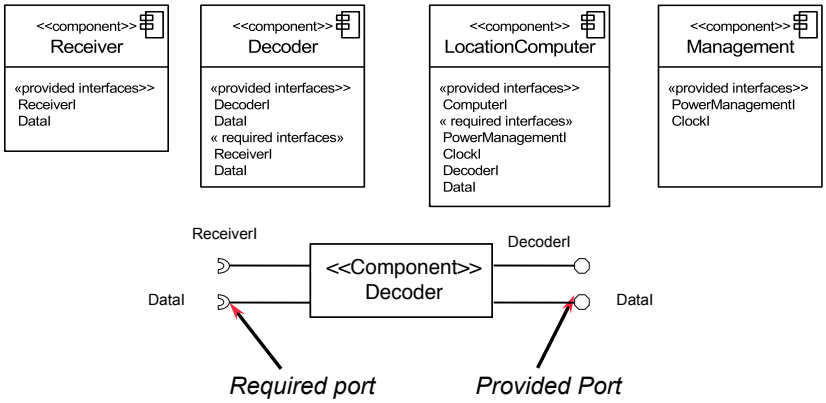
- Many more sequence diagrams needed…

Modeling a (simplified) GPS device

- Targeting multiple products with the same (business) model
  - Hand held autonomous device
  - Plug-in device for Smart Phone
  - Plug-in device for laptop (PCMCIA/USB)
  - May need to change part of the software after deployment
- We choose a component based delivery of the software

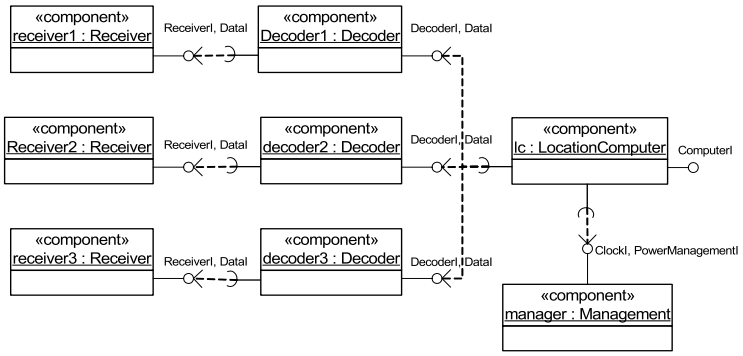
Modeling a (simplified) GPS device

- Component diagram



Modeling a (simplified) GPS device

- Deployment diagram

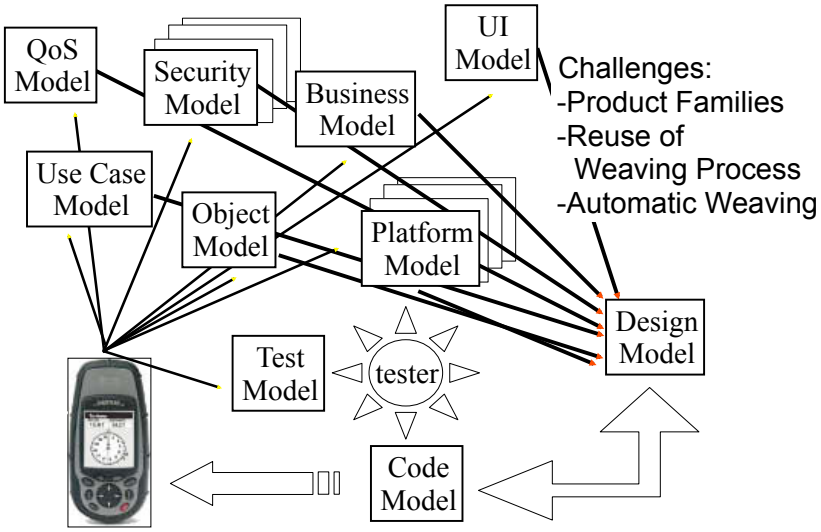


Model and Reality in Software

- Sun Tse: *Do not take the map for the reality*
- William James: *The concept 'dog' does not bite*
- Magritte:

- Software Models: from contemplative to productive

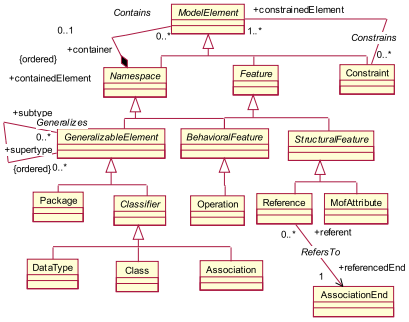
Modeling and Weaving

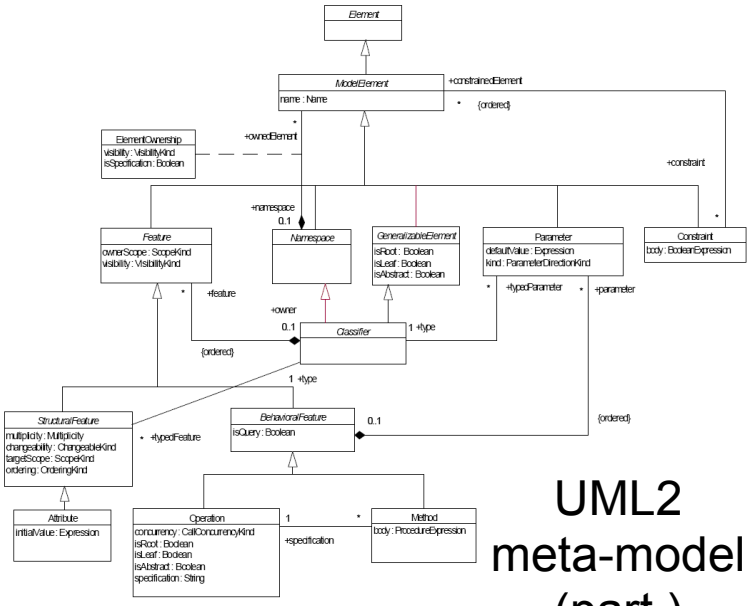


Challenges:
-Product Families
-Reuse of Weaving Process
-Automatic Weaving

Assigning Meaning to Models

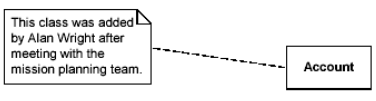
- If a UML model *is no longer* just
  - fancy pictures to decorate your room
  - a graphical syntax for C++/Java/C#/Eiffel...
- Then tools must be able to manipulate models
  - Let's make a model of what a model is!
  - => ***meta-modeling***
    - » & meta-meta-modeling..

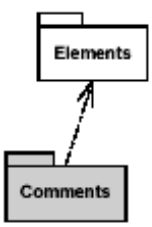


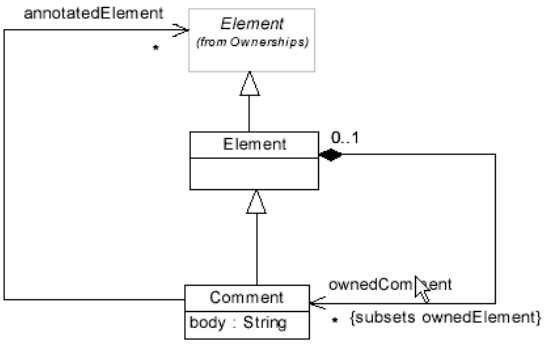


UML2 meta-model (part.)

Zoom: comments







Generalizations

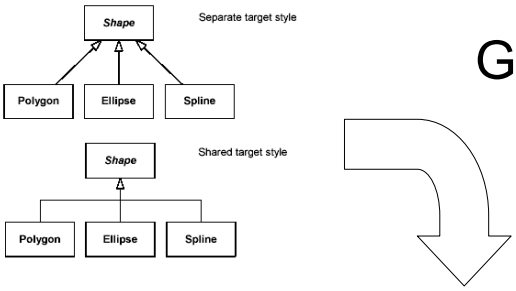


Figure 3-33. Examples of generalizations between classes.

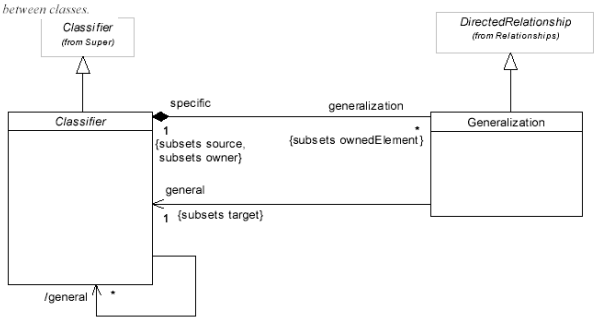
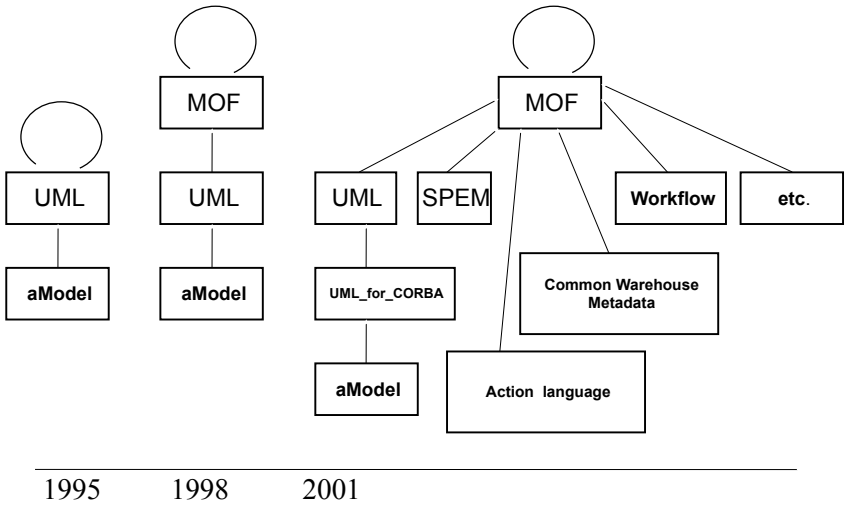
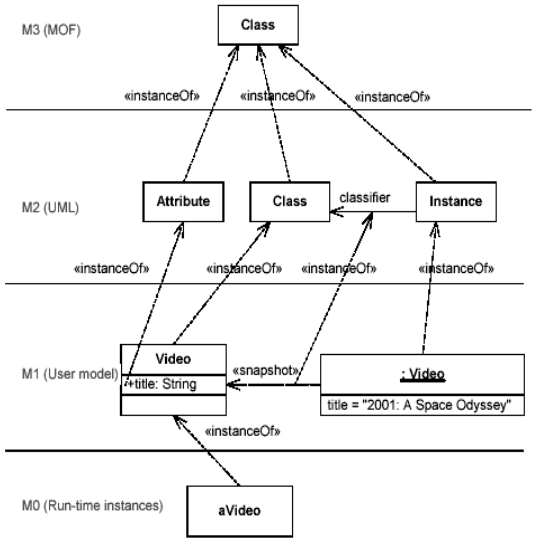


Figure 3-32. The elements defined in the Generalizations package.

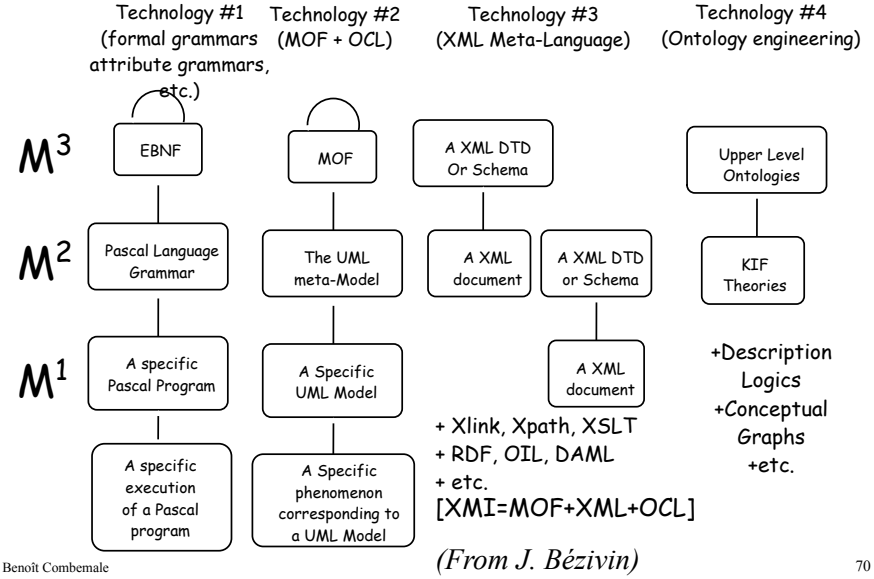
Modeling techniques at OMG: 3 steps



The 4 layers in practice



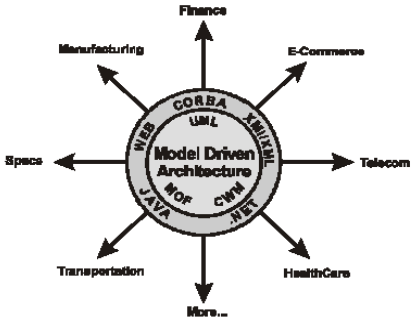
Comparing Abstract Syntax Systems



(From J. Bézivin)

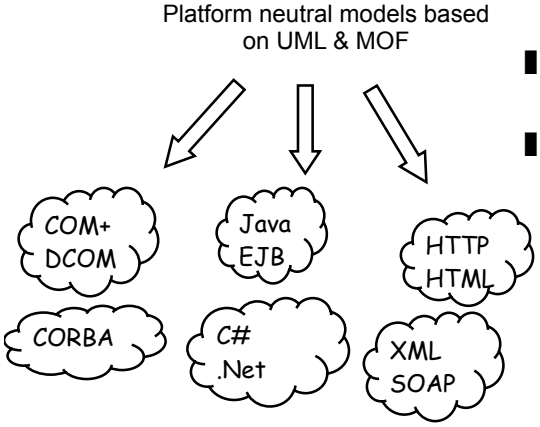
MDA: the OMG new vision

"OMG is in the ideal position to provide the model-based standards that are necessary to extend integration beyond the middleware approach… Now is the time to put this plan into effect. Now is the time for the Model Driven Architecture."



Richard Soley & OMG staff,
MDA Whitepaper Draft 3.2
November 27, 2000

Mappings to multiple and evolving



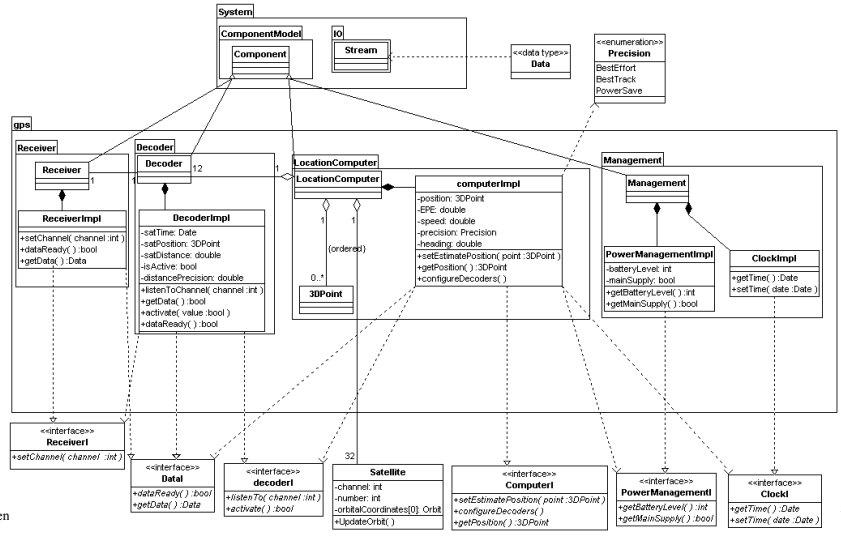
- MOF & UML as the core
- Organization assets expressed as models
- Model transformations to map to technology specific platforms

The core idea of MDA: PIMs & PSMs

- MDA models
  - **PIM**: Platform Independent Model
    - Business Model of a system abstracting away the deployement details of a system
    - Example: the UML model of the GPS system
  - **PSM**: Platform Specific Model
    - Operational model including platform specific aspects
    - Example: the UML model of the GPS system on .NET
      - Possibly expressed with a UML profile (.NET profile for UML)
  - Not so clear about platform models
    - Reusable model at various levels of abstraction
      - CCM, C#, EJB, EDOC, …

A PSM for our GPS

- Here, the platform is .NET

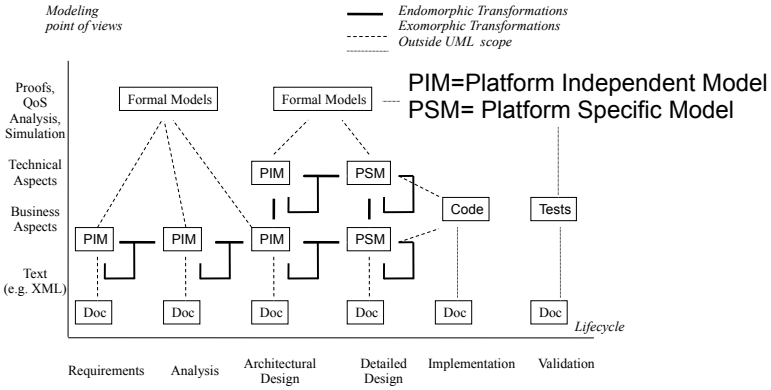


How to go From PIM to PSM?

- "just" weave the platform aspect !
- How can I do that?
  - Through Model transformations
  - Now hot topic at OMG with RFP Q/V/T
    - Query/View/Transformation

Weaving aspects into UML Models?

- It's what Model Driven Architecture is about!

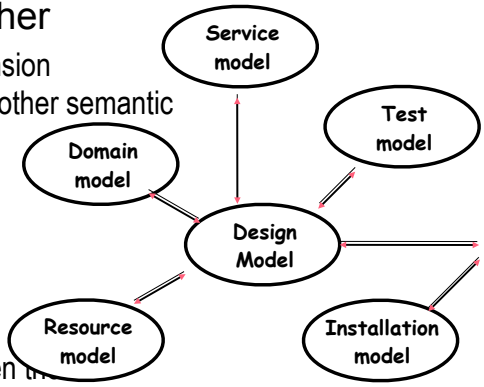


But many more dimensions in modeling!

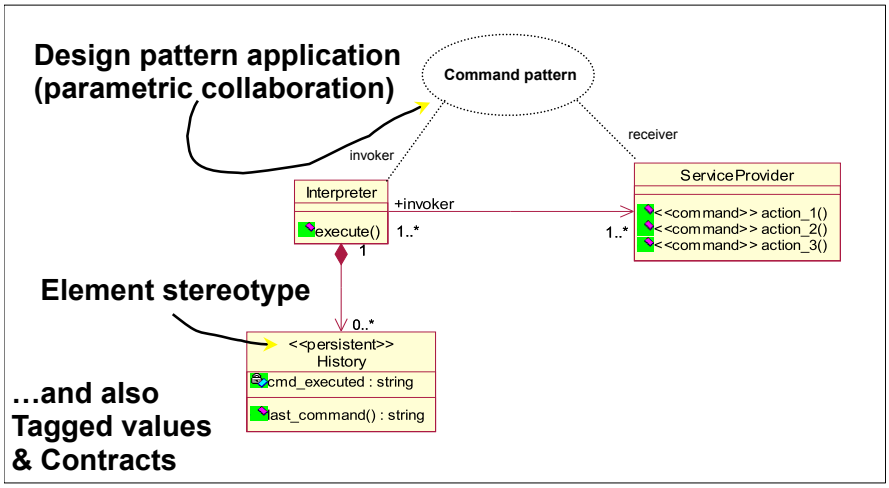
- Beyond Design Model
  - where UML is arguably good…
- Business model
- GUI model
- Development process model
- Performance & Resource model
- Deployment model
- Test model
- Etc.

How to take these dimensions into account?

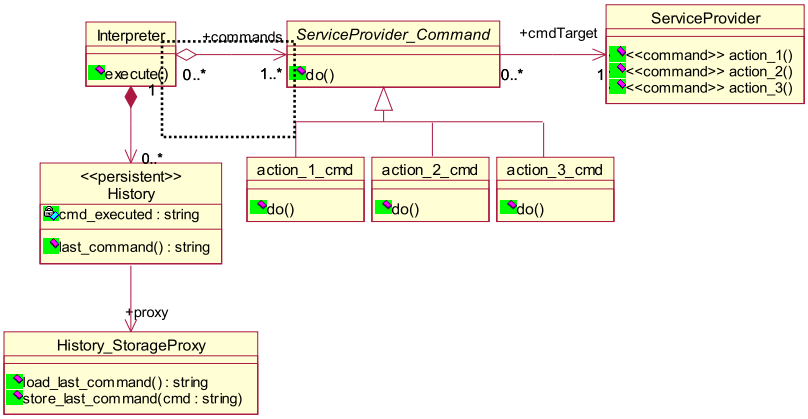
- Expression with either
  - UML, using built-in extension mechanisms to link with other semantic domains
  - DS(M)L, based on MOF
- Exploitation
  - Weave all these aspects into a design model
  - Define *mappings* between the aspects (as in e.g. *federations*)



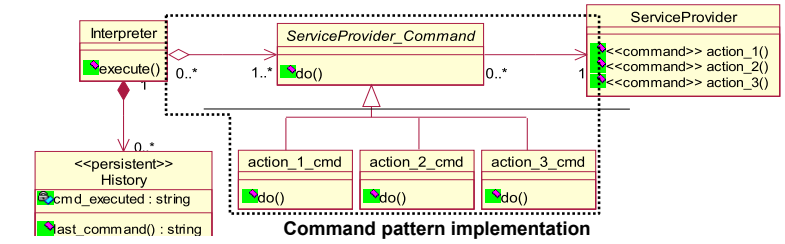
Embedding implicit semantics into a model

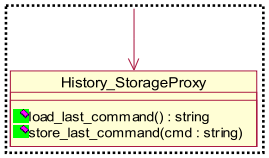


…and the result we want...



How To: Automatic Model Transformations





In some domains (e.g.; RT systems) transformations can get more complex than initial model!
=> must be managed with sound SE principles

Why complex transformations?

- Example: Air Traffic Management
  - "business model" quite stable & not that complex
- Various modeling languages used beyond UML
  - As many points of views as stakeholders
- Deliver software for (many) variants of a platform
  - Heterogeneity is the rule
- Reuse technical solutions across large product lines (e.g. fault tolerance, security…)
- Customize generic transformations
- Compose reusable transformations
- Evolve & maintain transformations for 15+ years!

The 3 ages of Transformations

- Awk-like (inc. *sed, perl*…)

```
BEGIN {action)
pattern #1 {action #1}
…
pattern #n {action #n}
END {action)
```

SE Limit: ~100 LOC

- XSLT
  - W3C standard for transforming XML
  - Operates on tree structures
  - syntactical & inefficient

SE Limit: ~1000 LOC

- QVT-like
  - OMG Standard for transforming model
    - » Query/View/Transformation
  - Operates on graphs

SE Limit: ?
Standard?
Which/When?

Transformations are Assets => apply sound SE principles

- Must be Modeled
  - with the UML, using the power of OO
- Must be Designed
  - Design by Contract, using OCL
- Must be Implemented
  - Made available through libraries of components, frameworks…
- Must be Tested
  - test cases
    - » input: a UML Model
    - » output: a UML Model, + contract checking
- Must be Evolved
  - Items of Configuration Management
  - Transformations of transformations

Principles

1. Everything relevant to the development process is a model
2. All the meta-models can be written in a language of a unique meta-meta-model
   - » Same M3 helps Interoperability of tools defined at M2
3. A development process can be modelled as a partially ordered set of model transformations, that take models as input and produce models as output

Consequences

1. Models are aspect oriented. Conversely: Aspects are models
2. Transformations are aspects weavers. They can be modeled.
3. Every meta-model defines a domain specific language
4. Software development has two dimensions: M1-model development and M2-transformation development

Challenges

- Language definition problems (Q/V/T)
  - Expressive, easy to use language(s) for transformations
- Technological Issues
  - Tool set, interoperability…
- Software Engineering Issues
  - From requirements to tests and SCM
- Theoretical issues
  - A transformation is a *mapping* between semantic domains
    - » Beyond Code Generation: Proof, Performance Evaluation, Test Synthesis …
  - Compositional weaving, cf. Aspect Oriented Software Dvp

UML & Model Driven Architecture: Summary

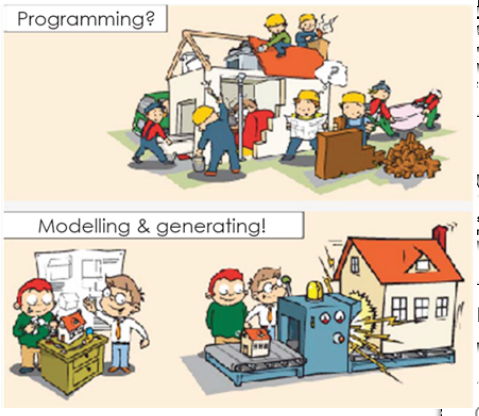
- Modeling to master complexity
  - Multi-dimensional and aspect oriented by definition
- Models: from contemplative to productive
  - Meta-modeling tools
- Model Driven Engineering
  - Weaving aspects into a design model
    - » E.g. Platform Specificities
- Model Driven Architecture (PIM / PSM): just a special case of Aspect Oriented Design
- Related: Generative Prog, Software Factories

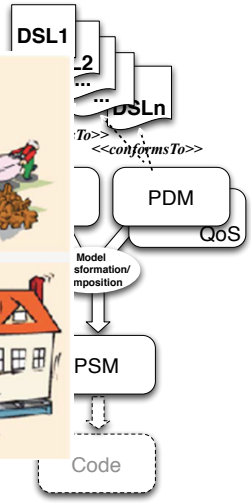
Une autre vision (grossière) du MDE ! en un slide...

MDE (Mode

=**DSL** (Doma
séparation de
(p.-ex., M
séparation du
forme)

+**Approch**
s'appuie sur l
raisonner (au
modèles





L'informaticien 2.0

- **Expert Métier :**

  Capitalise son expérience au sein de langages métiers qu'il utilise pour la définition de système complexe

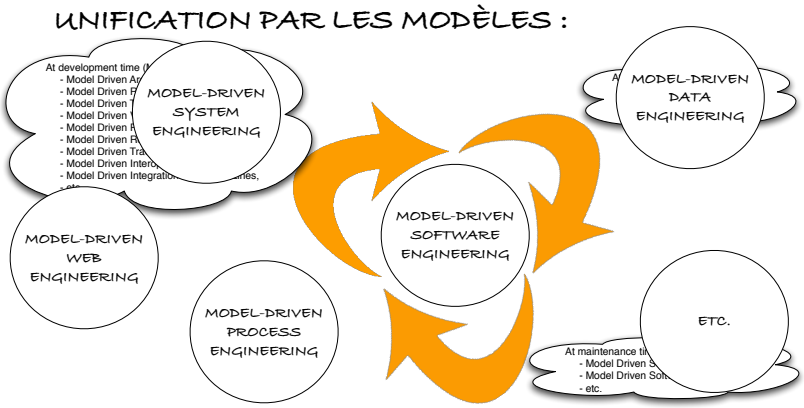
- **Expert Plate-forme :**

  Fournit des approches génératives prenant en compte les spécificités d'une plate-forme d'exécution particulière

- **Expert Langage :**

  Fournit des *méta*-approches génératives facilitant l'outillage d'un nouveau DSL

Domaines d'application



=> VERS UNE INGÉNIERIE DES LANGAGES (DSL)...

Outline

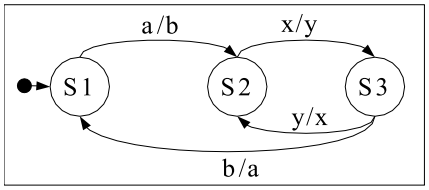
- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up

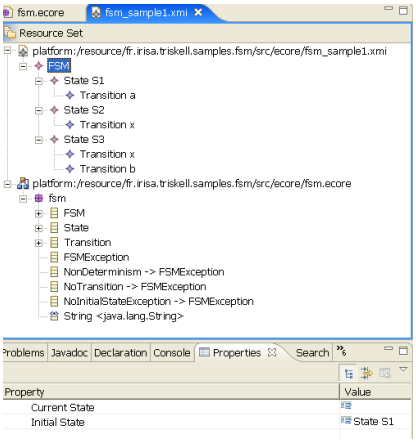
Meta-Models as Shared Knowledge

- Definition of an Abstract Syntax in E-MOF
  - Repository of models with EMF
  - Reflexive Editor in Eclipse
  - JMI for accessing models from Java
  - XML serialization for model exchanges
- Applied in more and more projects
  - SPEEDS, OpenEmbedd,DiVA…

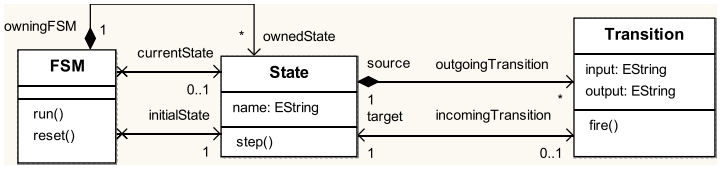
Example with StateMachines

Model

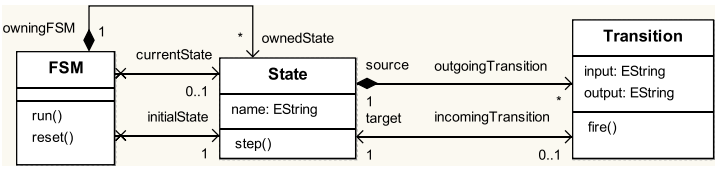




Meta-Model



Breathing life into Meta-Models




```
Context FSM
inv: ownedState->forAll(s1,s2|
s1.name=s2.name implies s1=s2)

class FSM {
   operation reset() : Void {
         currentState := initialState
}}
```

```
class Minimizer {
   operation minimize (source: FSM):FSM {…}
}
```

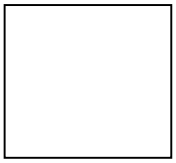

DIY with LOGO programs

- Consider LOGO programs of the form:

repeat 3 [pendown forward 3 penup forward 4]

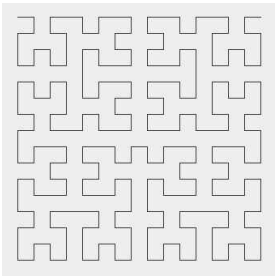
___ ___ ___

```
to square :width
  repeat 4 [ forward :width right 90 ]
end
pendown square 10
```



Fractals in LOGO

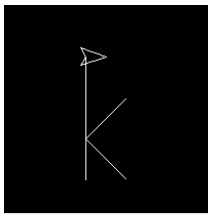
```
; lefthilbert
to lefthilbert :level :size
  if :level != 0 [
     left 90
     righthilbert :level-1 :size
     forward :size
     right 90
     lefthilbert :level-1 :size
     forward :size
     lefthilbert :level-1 :size
     right 90
     forward :size
     righthilbert :level-1 :size
     left 90
  ]
end
```

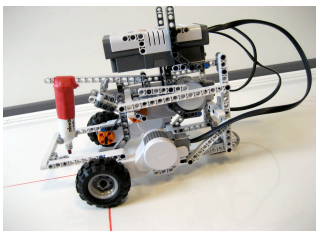


```
; righthilbert
to righthilbert :level :size
   if :level != 0 [
      right 90
      lefthilbert :level-1 :size
      forward :size
      left 90
      righthilbert level-1 :size
      forward :size
      righthilbert :level-1 :size
      left 90
      forward :size
      lefthilbert :level-1 :size
      right 90
   ]
end
```


Case Study: Building a Programming Environment for Logo

- Featuring
  - Edition in Eclipse
  - On screen simulation
  - Compilation for a Lego Mindstorms robot





Model Driven Language Engineering : the Process

- Specify abstract syntax
- Specify concrete syntax
- Build specific editors
- Specify static semantics
- Specify dynamic semantics
- Build simulator
- Compile to a specific platform

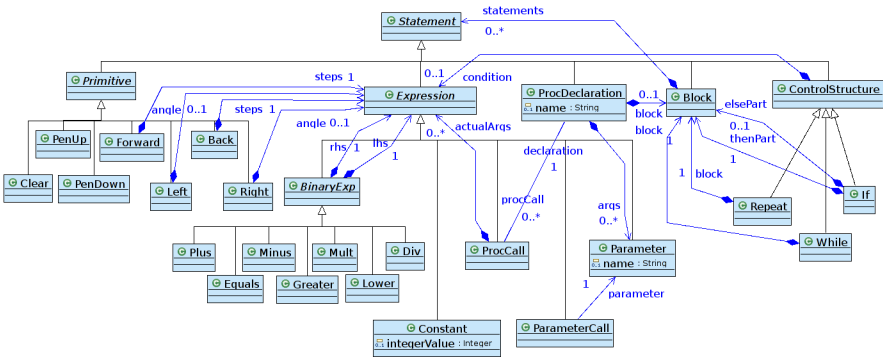
Meta-Modeling LOGO programs

- Let's build a meta-model for LOGO
  - Concentrate on the abstract syntax
  - Look for concepts: instructions, expressions…
  - Find relationships between these concepts
    - » It's like UML modeling !
- Defined as an ECore model
  - Using EMF tools and editors

LOGO metamodel

ASMLogo.ecore

LOGO metamodel



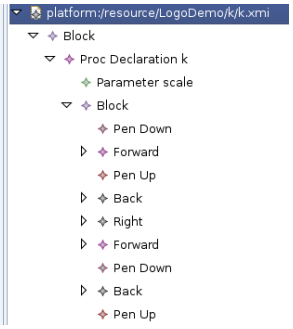
ASMLogo.ecore

Concrete syntax

- Any regular EMF based tools
- Textual using Sintaks, TMF...
- Graphical using GMF or TopCased

logo.sts

```
TO k :scale
    PENDOWN
    FORWARD *(30, :scale)
    PENUP
    BACK *(10, :scale)
    RIGHT 45
    FORWARD *(14, :scale)
    PENDOWN
    BACK *(14, :scale)
    PENUP
    RIGHT 90
    FORWARD *(14, :scale)
    PENDOWN
    BACK *(14, :scale)
    PENUP
    RIGHT 45
    FORWARD *(20, :scale)
    LEFT 180
END
```



Outline

- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up

Static Semantics with OCL

- Complementing a meta-model with Well-Formedness Rules, aka *Contracts* e.g.;
  - A procedure is called with the same number of arguments as specified in its declaration
- Expressed with the OCL (Object Constraint Language)
  - The OCL is a language of typed expressions.
  - A constraint is a valid OCL expression of type Boolean.
  - A constraint is a restriction on one or more values of (part of) an object-oriented model or system.

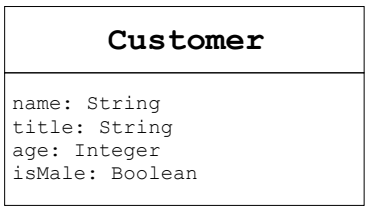
Contracts in OO languages

- Inspired by the notion of Abstract Data Type
- Specification = Signature +
  - Preconditions
  - Postconditions
  - Class Invariants
- Behavioral contracts are inherited in subclasses

OCL

- Can be used at both
  - M1 level (constraints on Models)
    - aka *Design-by-Contract* (Meyer)
  - M2 level (constraints on Meta-Models)
    - aka Static semantics
- Let's overview it with M1 level exemples

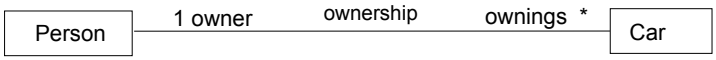
Simple constraints




```
title = if isMale then 'Mr.' else 'Ms.' endif
age >= 18 and age < 66
name.size < 100
```

Non-local contracts: navigating associations

- Each association is a navigation path
  - The context of an OCL expression is the starting point
  - Role names are used to select which association is to be traversed (or target class name if only one)



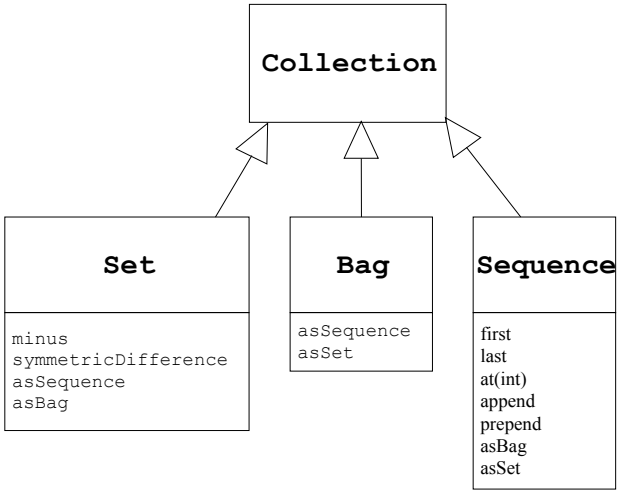
Context Car inv:
self.owner.age >= 18

Navigation of 0..* associations

- Through navigation, we no longer get a scalar but a *collection* of objects
- OCL defines 3 sub-types of collection
  - **Set** : when navigation of a 0..* association
    - » *Context Person inv: ownings* return a Set[Car]
    - » Each element is in the Set at most once
  - **Bag :** if more than one navigation step
    - » An element can be present more than once in the Bag
  - **Sequence** : navigation of an association {ordered}
    - » It is an ordered Bag
- Many predefined operations on type *collection*

Syntax::
Collection->operation

Collection hierarchy



Basic operations on collections

- *isEmpty*
  - *true* if collection has no element
- *notEmpty*
  - *true* if collection has at least one element

Context Person inv:
age<18 implies ownings->isEmpty

- *size*
  - Number of elements in the collection
- *count (elem*)
  - Number of occurrences of element *elem* in the collection

select Operation

- possible syntax
  - collection->select(elem:T | expr)
  - collection->select(elem | expr)
  - collection->select(expr)
- Selects the subset of *collection* for which property *expr* holds
- e.g.

context Person inv:
ownings->select(v: Car | v.mileage<100000)->notEmpty

- shortcut:

context Person inv:
ownings->select(mileage<100000)->notEmpty

forAll Operation

- possible syntax
  - collection->forall(elem:T | expr)
  - collection->forall(elem | expr)
  - collection->forall(expr)
- True iff *expr* holds for each element of the *collection*
- e.g.

context Person inv:
ownings->forall(v: Car | v.mileage<100000)

- shortcut:

context Person inv:
ownings->forall(mileage<100000)

Operations on Collections

Operation	Description
size	The number of elements in the collection
count(object)	The number of occurences of object in the collection.
includes(object)	True if the object is an element of the collection.
includesAll(collection)	True if all elements of the parameter collection are present in the current collection.
isEmpty	True if the collection contains no elements.
notEmpty	True if the collection contains one or more elements.
iterate(expression)	Expression is evaluated for every element in the collection.
sum(collection)	The addition of all elements in the collection.
exists(expression)	True if expression is true for at least one element in the collection.
forAll(expression)	True if expression is true for all elements.

Static Semantics for LOGO

- No two formal parameters of a procedure may have the same name:

- A procedure is called with the same number of arguments as specified in its declaration:

Static Semantics for LOGO

- No two formal parameters of a procedure may have the same name:

```
context ProcDeclaration
  inv unique_names_for_formal_arguments :
        args -> forAll ( a1 , a2 | a1. name = a2.name
              implies a1 = a2 )
```

- A procedure is called with the same number of arguments as specified in its declaration:

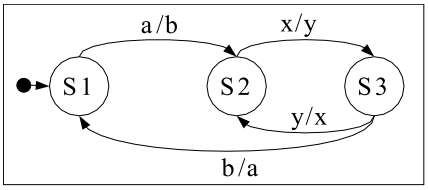
```
context ProcCall
  inv same_number_of_formals_and_actuals :
        actualArgs -> size = declaration .args -> size
```

Outline

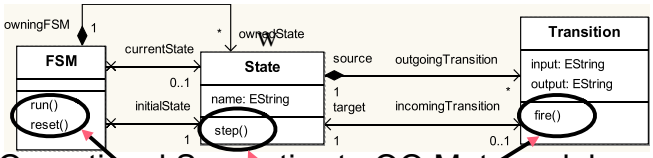
- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up

Operational Semantics of State Machines

- A model



- Its metamodel



- Adding Operational Semantics to OO Metamodels

Metadata languages

- (E)MOF => Only data structures
  - classes, properties, associations, ...
  - operations : only signatures
- Not sufficient to operate on models
  - Constraints
  - Actions
  - Transformations
  - ...

Typical example (excerpted from MOF spec)

- *Operation **isInstance(element : Element) : Boolean***
  - "*Returns true if the element is an instance of this type or a subclass of this type. Returns false if the element is null*".

A natural language specification

```
operation isInstance (element : Element) : Boolean is do
  // false if the element is null
  if element == void then result := false
  else
    // true if the element is an instance of this type
    // or a subclass of this type
    result := element.getMetaClass == self or
        element.getMetaClass.allSuperClasses.contains(self)
  end
end
```

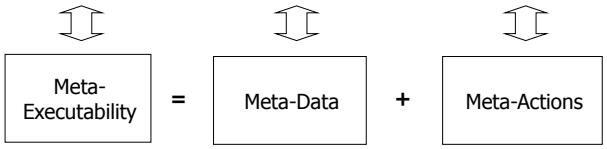
An operational specification

What is "meta"-executability?

- Basic CRUD Operations
- Merge, Composition…

- Simply an (object-oriented) program that manipulates model elements

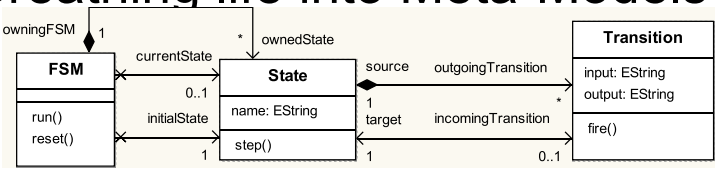
"Program = Data Structure + Algorithm", Niklaus Wirth



Kermeta Rationale

- Model, meta-model, meta-metamodel, DSLs…
  - Meta-bla-bla too complex for the normal engineer
- On the other hand, engineers are familiars with
  - OO programming languages (Java,C#,C++,..)
  - UML (at least class diagram)
  - May have heard of *Design-by-Contract*
- Kermeta leverages this familiarity to make Meta-modeling easy for the masses

Breathing life into Meta-Models

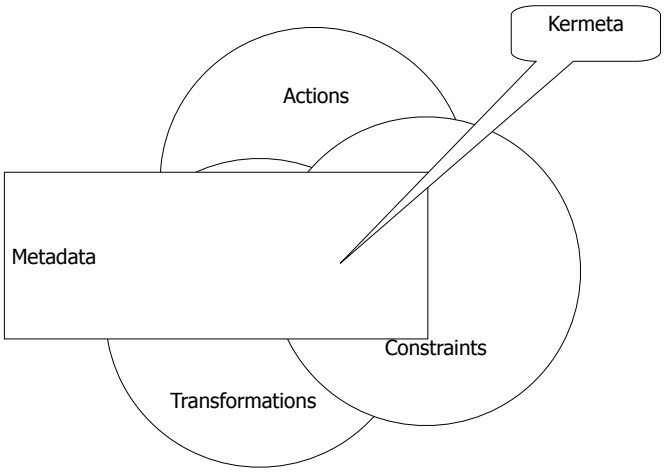



```
// MyKermetaProgram.kmt
// An E-MOF metamodel is an OO program that does nothing
    require "StateMachine.ecore" // to import it in Kermeta
// Kermeta lets you weave in aspects
    // Contracts (OCL WFR)
    require “StaticSemantics.ocl”
    // Method bodies (Dynamic semantics)
    require “DynamicSemantics.kmt”
    // Transformations

  class Minimizer {
     operation minimize (source: FSM):FSM {…}
  }
```

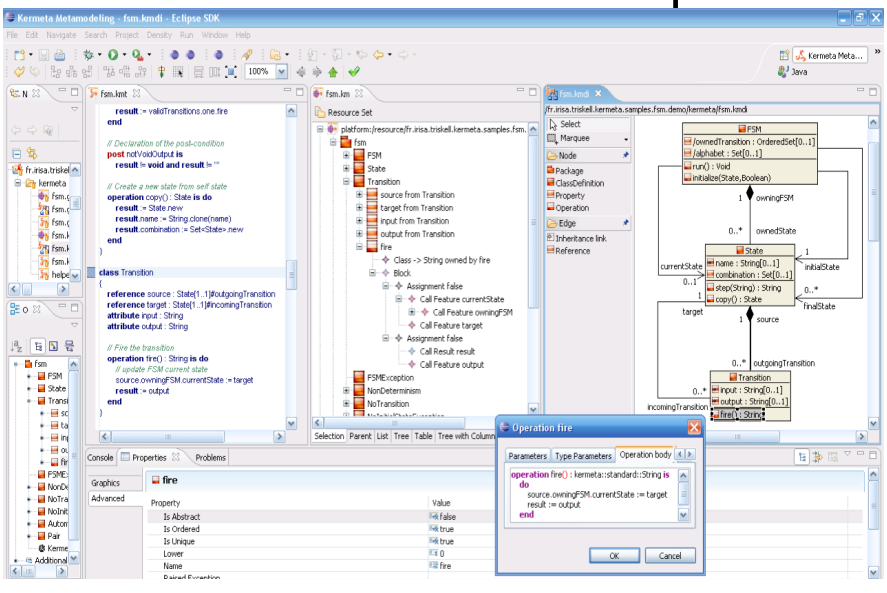
```
Context FSM
inv: ownedState->forAll(s1,s2|
s1.name=s2.name implies s1=s2)
```

```
aspect class FSM {
  operation reset() : Void {
        currentState := initialState
```


Kermeta, a Kernel to Meta



Kermeta workbench snapshot



Kermeta: a **Ker**nel **meta**modeling language

- Strict EMOF extension
- Statically Typed
  - Generics, Function types (for OCL-like iterators)
- Object-Oriented
  - Multiple inheritance / dynamic binding / reflection
- Model-Oriented
  - Associations / Compositions
  - Model are first class citizens, notion of model type
- Aspect-Oriented
  - Simple syntax for static introduction
  - Arbitrary complex aspect weaving as a framework
- Still “kernel” language
  - Seamless import of Java classes in Kermeta for GUI/IO etc.

Types & opérateurs usuels

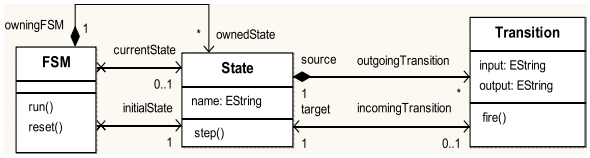
- Types scalaires très restreints
  - Integer, String, Boolean
- Opérateurs :
  - Affectation : := (naïve), ?= (cast)
  - Arithmétique : +,-,/,*
  - Comparaison : ==, !=, <, <=,>,>=
  - Logique : and, or, not
- Collections : fondées sur la définition d'OCL

Mot–clé	Classe générique	Unicité	Ordre
set	Set< T >	Oui	Non
oset	OrderedSet< T >	Oui	Oui
bag	Bag< T >	Non	Non
seq	Sequence< T >	Non	Oui

Définition de classes, opérations, méthodes

- Déclaration de classes à la Java (class C { })
  - Classes abstraites (abstract), généricité (class A<T>)
  - Héritage (inherits), multiple ou non
- Constructions : pas de constructeur ! (MyClass.new)
- Variables de classes : Attributs & Référence
  - attribute a: String ⇒ a est contenue par composition ()
  - reference r: String ⇒ r est référencée
  - self représente l'instance courante
  - Absence de visibilité : tout est public
- Méthode d'instance : Opérations & Méthodes
  - operation name(arg1: T): OutputType is do ... end
  - Redéfinition par héritage : operation → method
  - Variable locale : var tmp: String init String.new
  - Retour : pas de return ! On utilise la variable result
  - Pas de surcharge dans le langage (simplification)

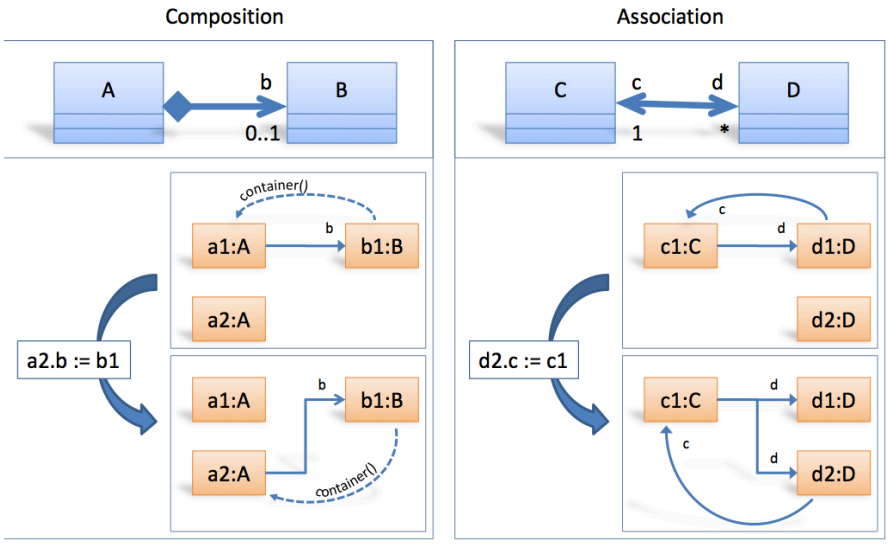
EMOF ⇔Kermeta




```
class FSM
{
attribute ownedState : State[0..*]#owningFSM
reference initialState : State[1..1]
reference currentState : State
operation run() : kermeta::standard::~Void is do
end
operation reset() : kermeta::standard::~Void is do
end
}
```

```
class State{
  reference owningFSM : FSM[1..1]#ownedState
  attribute name : String
  attribute outgoingTransition : Transition[0..*]#source
  reference incomingTransition : Transition#target
  operation step(c : String) : kermeta::standard::~Void is do
   end
}
class Transition{
  reference source : State[1..1]#outgoingTransition
  reference target : State[1..1]#incomingTransition
  attribute input : String
  attribute output : String
  operation fire() : String is do
  end
}
```

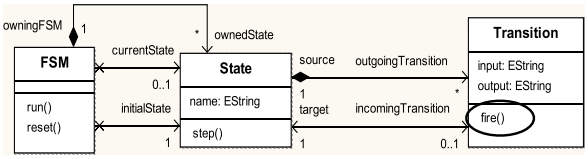

Assignment semantics



Fermetures & λ-fonctions pour l'itération

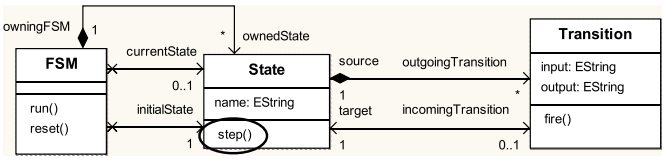
- Effectuer une action ∀e ∈ C each
  - f.each{ n | stdio.write(n.toString + " ") }
- Vérifier une condition ∀e ∈ C forAll
  - var b: Boolean init f.forAll{ n | n < 250 }
- Sélection d'un sous-ensemble (filter) select
  - var f2: Sequence<Integer> init f.select{n | n < 100}
- Exclusion d'un sous-ensemble reject
  - var f3: Sequence<Integer> init f.reject{n | n < 100}
- Mapping de fonction collect
  - var f4: Sequence<Integer> init fib.collect{n | n + 1}
- Détection d'un élément detect
  - var x: Integer init fib.detect{n | n > 47}
- Existence d'un élément exists
  - var b2: Boolean init fib.exists{n | n > 47 }

Example



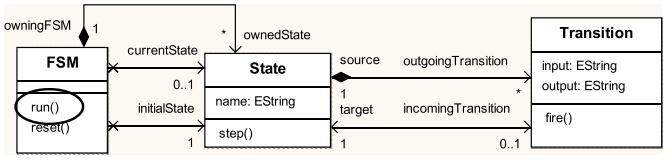
operation fire() : String

```
source.owningFSM.currentState := target
result := output
```

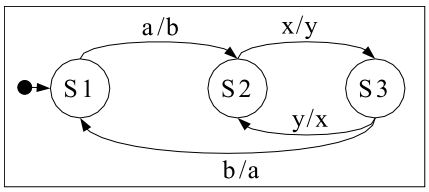
operation step(c : String) : String

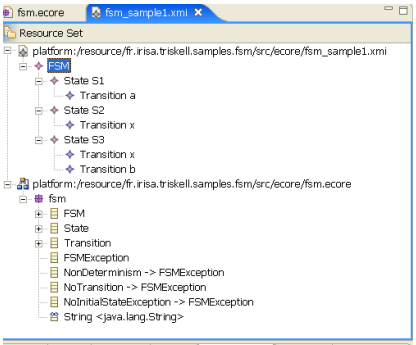
```
// Get the valid transitions
var validTransitions : Collection<Transition>
validTransitions :=       outgoingTransition.select { t |
        t.input.equals(c)
}
// Check if there is one and only one valid transition
if validTransitions.empty then raise NoTransition.new
end
if validTransitions.size > 1 then
        raise NonDeterminism.new
end
// fire the transition
result := validTransitions.one.fire
```

operation run() : **Void**

```
from var str : String
until str == "exit"
loop
        stdio.writeln("current state is " + currentState.name)
        str := stdio.read("Enter an input string or 'exit'
                                                to exit simulation : ")
        stdio.writeln(str)
        if str != "exit" then
                do
                        stdio.writeln("Output string : " + currentState.step
(str))
                rescue (ex : FSMException)
                        stdio.writeln("ERROR : " + ex.toString)
                end
        end
end
stdio.writeln("* END OF SIMULATION *")
```



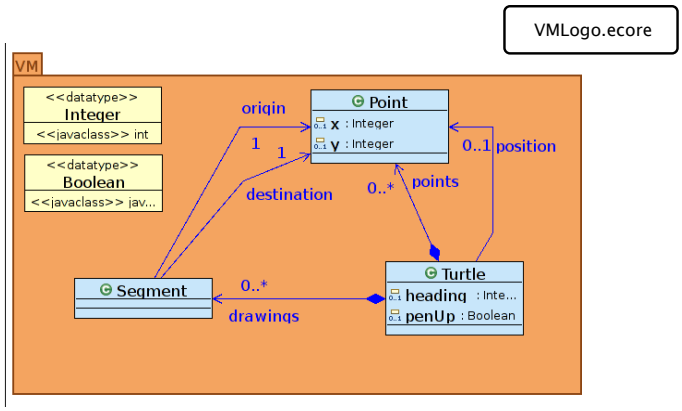



```
/**
 * Load a sample FSM from a xmi2 file
 */
operation loadFSM() : FSM is do
  var repository : EMFRepository init EMFRepository.new
  var resource : EMFResource
  resource ?= repository.createResource("../models/fsm_sample1.xmi", "../metamodels/fsm.ecore")
  resource.load


  // Load the fsm (we get the main instance)
  result ?= resource.instances.one
end
```


Operational Semantics for LOGO

- Expressed as a mapping from a meta-model to a virtual machine (VM)
- LOGO VM ?
  - Concept of Turtle, Lines, points…
  - Let's Model it !
  - (Defined as an Ecore meta-model)

Virtual Machine - Model



- Defined as an Ecore meta-model

Virtual Machine - Semantics

LogoVMSemantics.kmt

```
require "VMLogo.ecore"
require "TurtleGUI.kmt"

aspect class Point {
  method toString() : String is do
    result := "[" + x.toString + "," + y.toString + "]"
  end
}

aspect class Turtle {
  operation setPenUp(b : Boolean) is do
    penUp := b
  end
  operation rotate(angle : Integer)  is do
    heading := (heading + angle).mod(360)
  end
}
```


Map Instructions to VM Actions

- Weave an interpretation aspect into the meta-model
  - add an *eval()* method into each class of the LOGO MM

```
aspect class PenUp {
    eval (ctx: Context) {
        ctx.getTurtle().setPenUp
      (true)
    }
…
aspect class Clear {
    eval (ctx: Context) {
        ctx.getTurtle().reset()
    }
```


Meta-level Anchoring

- Simple approach using the Kermeta VM to « ground » the semantics of basic operations
- Or reify it into the LOGO VM
  - Using eg a stack-based machine
  - Ultimately grounding it in kermeta though

```
…
aspect class Add {
   eval (ctx: Context)  : Integer {
        result = lhs.eval(ctx)
                + rhs.eval(ctx)
}
```

```
…
aspect class Add {
   eval (ctx: Context) {
        lhs.eval(ctx) // put result
                // on top of ctx
     stack
        rhs.eval(ctx) // idem
        ctx.getMachine().add()
}
```

Handling control structures

- Block
- Conditional
- Repeat
- While

Operational semantics

LogoDynSemantics.kmt

```
require "ASMLogo.ecore"
require "LogoVMSemantics.kmt"

aspect class If {
  operation eval(context : Context) : Integer is do
    if condition.eval(context) != 0 then
      result := thenPart.eval(context)
    else result := elsePart.eval(context)
    end
  end
}

aspect class Right {
  operation eval(context : Context) : Integer is do
    context.turtle.rotate(angle.eval(context))
  end
}
```


Handling function calls

- Use a stack frame
  - Owned in the Context

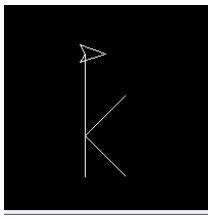
Getting an Interpreter

- Glue that is needed to load models
  - ie LOGO programs
- Vizualize the result
  - Print traces as text
  - Put an observer on the LOGO VM to graphically display the resulting figure

Simulator

- Execute the operational semantics

```
TO k :scale
    PENDOWN
    FORWARD *(30, :scale)
    PENUP
    BACK *(10, :scale)
    RIGHT 45
    FORWARD *(14, :scale)
    PENDOWN
    BACK *(14, :scale)
    PENUP
    RIGHT 90
    FORWARD *(14, :scale)
    PENDOWN
    BACK *(14, :scale)
    PENUP
    RIGHT 45
    FORWARD *(20, :scale)
    LEFT 180
END

CLEAR
$k(4)
```



Problems | Javadoc | Declaration | Console | Pro

KM Logo Console

```
Launching logo interpreter on file : /home/
Tortue trace vers [0,120]
Tortue se deplace en [0,80]
Tortue se deplace en [39,119]
Tortue trace vers [0,80]
Tortue se deplace en [39,41]
Tortue trace vers [0,80]
Tortue se deplace en [0,0]
Execution terminated successfully.
```


Outline

- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up

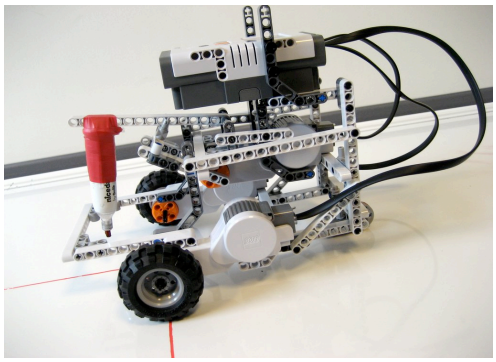
Implementing a model-driven compiler

- Map a LOGO program to Lego Mindstroms
  - The LOGO program is like a PIM
  - The target program is a PSM
  - => model transformation
- Kermeta to weave a « compilation » aspect into the logo meta-model

```
aspect class PenUp {
        compile (ctx: Context) {

    }
…
aspect class Clear {
    }
```


Specific platform

- Lego Mindstorms Turtle Robot
  - Two motors for wheels
  - One motor to control the pen



Model-to-Text vs. Model-to-Model

- Model-to-Text Transformations
  - For generating: code, xml, html, doc.
  - Should be limited to syntactic level transcoding
- Model-to-Model Transformations
  - To handle more complex, semantic driven transformations

Model-to-Text Approaches

- For generating: code, xml, html, doc.
  - Visitor-Based Approaches:
    - » Some visitor mechanisms to traverse the internal representation of a model and write code to a text stream
    - » Iterators, Write ()
  - Template-Based Approaches
    - » A template consists of the target text containing slices of meta-code to access information from the source and to perform text selection and iterative expansion
    - » The structure of a template resembles closely the text to be generated
    - » Textual templates are independent of the target language and simplify the generation of any textual artefacts

Classification of Model-to-Model Transformation Techniques

1. General purpose programming languages
   - Java/C#...
2. Generic transformation tools
   - Graph transformations, XSLT…
3. CASE tools scripting languages
   - Objecteering, Rose…
4. Dedicated model transformation tools
   - OMG QVT style
5. Meta-modeling tools
   - Metacase, Xactium, Kermeta…

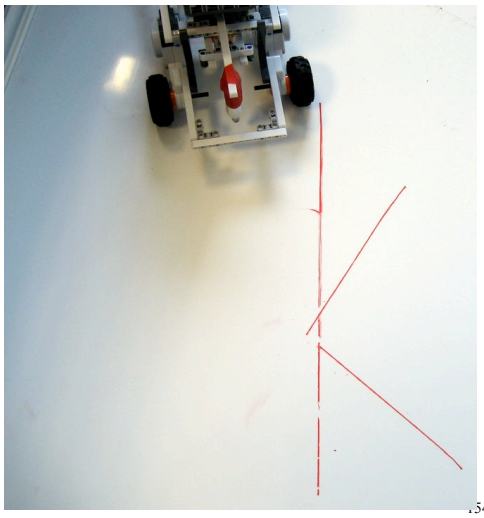
Logo to NXC Compiler

- Step 1 – Model-to-Model transformation

- Step 2 – Code generation with template

Execution

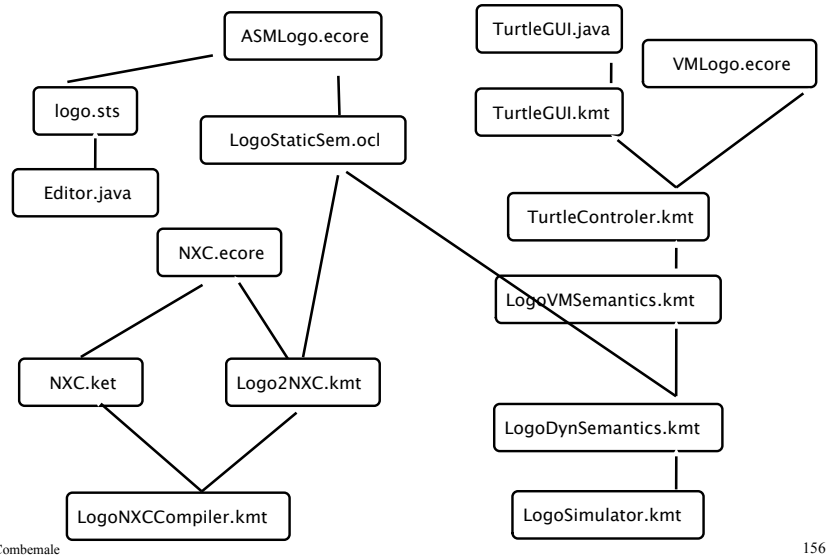
```
TO k :scale
    PENDOWN
    FORWARD *(30, :scale)
    PENUP
    BACK *(10, :scale)
    RIGHT 45
    FORWARD *(14, :scale)
    PENDOWN
    BACK *(14, :scale)
    PENUP
    RIGHT 90
    FORWARD *(14, :scale)
    PENDOWN
    BACK *(14, :scale)
    PENUP
    RIGHT 45
    FORWARD *(20, :scale)
    LEFT 180
END

CLEAR
$k(4)
```



Outline

- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up

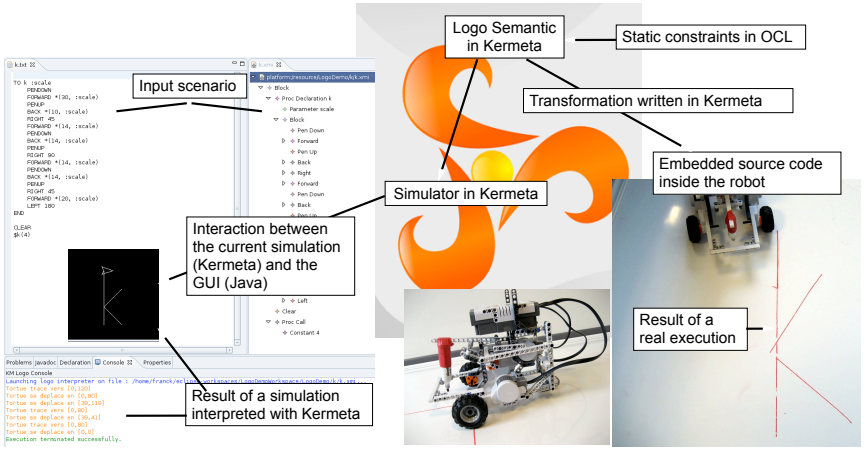
Logo Summary (1)



Logo Summary (2)

- Integrate all aspects coherently
  - syntax / semantics / tools
- Use appropriate languages
  - MOF for abstract syntax
  - OCL for static semantics
  - Kermeta for dynamic semantics
  - Java for simulation GUI
  - ...
- Keep separation between concerns
  - For maintainability and evolutions

From LOGO to Mindstorms



Kermeta in real projects

- **Artist2**, *the European network of excellence on real-time embedded systems*
- **UsineLogicielle**, *a System@tic project where Kermeta based operational semantic is associated to functional requirement for test synthesis purposes.*
- **Speeds**, *a European FP6 project for aspect-oriented metamodeling of avionics and automotive systems, including operational semantics aspects*
- **OpenEmbedd**, *A French project building a MDE platform for realtime system.*
- **Mopcom**, *a French project applying MDE to hardware for generating SOC and introduce dynamic reconfigurability to them.*
- **Topcased**, *a consortium that aims to build modelers for avionics and system engineering*
- **DiVA**, *a European FP7 STREP project on handling Dynamic variability in complex, adaptive systems*
- *Etc.*

Conclusion and Wrap-up

- Kermeta is an open-source initiative
  - Started January 2005
  - More than 10 active developers
- Feel free to use
  - Start with a meta-model in EMF
    - » Get XML an reflective editor for free
  - Weave in static semantics in OCL
  - Weave in an interpreter,
    - » connect to a simulation platform
  - Weave in one or more compilers
  - Finaly care for concrete syntax issues
- Feel free to contribute!
  - www.kermeta.org

Do It Yourself !

- Source code of the Logo demo:
  - https://gforge.inria.fr/scm/viewvc.php/trunk/kmlogo_projects/?root=kermeta
- Kermeta (http://www.kermeta.org/):
  - reference documentation: http://www.kermeta.org/documents
  - formation supports: https://gforge.inria.fr/scm/viewvc.php/integration/training_projects/?root=openembedd
- More information:
  - Eclipse Modeling (EMF, GMF...): http://www.eclipse.org/modeling/
  - OMG (UML, OCL, MOF...): http://www.omg.org/
- Webpage of this talk:
  - http://www.irisa.fr/triskell/perso_pro/combemale/teaching/mde/

Thank you !

- Questions ?