Introduction to Model-Driven Engineering

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

Prof. Jean-Marc Jézéquel
(Univ. Rennes 1 & INRIA)
Triskell Team @ IRISA
Campus de Beaulieu
F-35042 Rennes Cedex
Tel : +33 299 847 192 Fax : +33 299 847 171
e-mail : jezequel@irisa.fr
http://www.irisa.fr/prive/jezequel

Who (does not) know this?

- Rovio's Angry Bird
  - more than 1.7 billion downloads
  - hundreds of millions of monthly active users
  - Revenue > $500M

How would you build Angry Birds?

- Only from a technical perspective
  - Leaving away the Art Design & brilliant marketing
- The game is physics-based
  - you adjust the trajectory and power of the slingshot with your finger
- Architecture?

Additional issues

- Frameworks: Box2d, PlayN
- Plateform: Android, Chrome, webOS, iOS, Mac, Maemo, Symbian, PlayStation Portable, PlayStation 3,Windows, Windows Phone, Bada
- Versions: Angry Birds, Angry Birds Seasons, Angry Birds Rio, Angry Birds Space, Angry Birds Heikki, Angry Birds Star Wars, Bad Piggies
- Pb: sync accross devices?

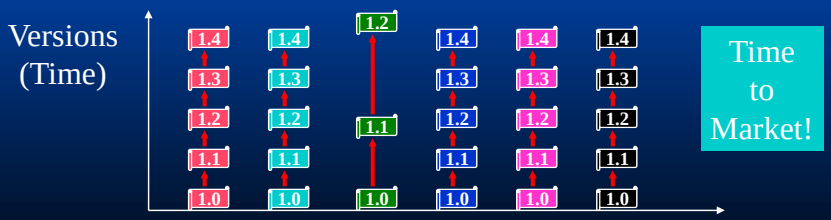
Philips (Medical Systems)

- Not just for games

Airbus

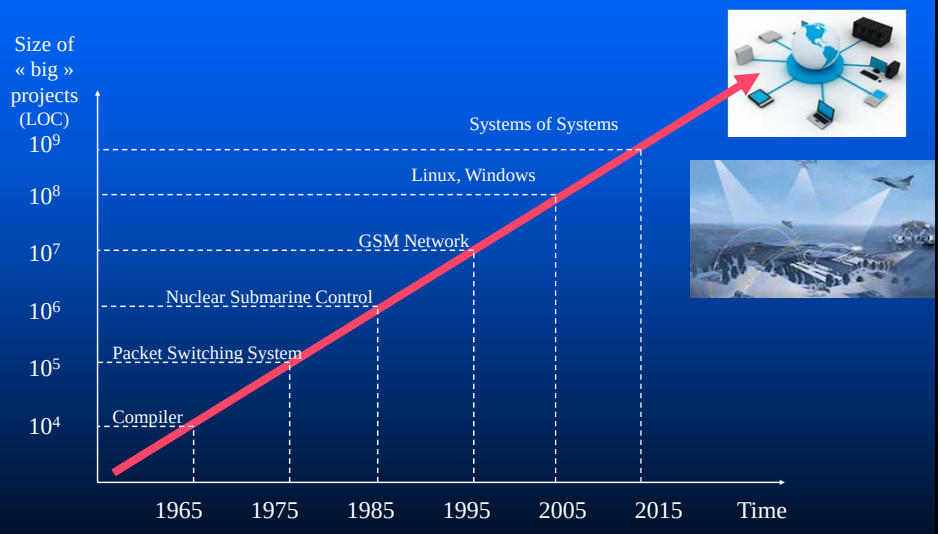
Modern Software Problems

- Importance of non-functional properties
  - distributed systems, parallel & asynchronous
  - quality of service : reliability, latency, performance...
- Flexibility of functional aspects: Product Lines
  - notion of *product lines* (space, time)



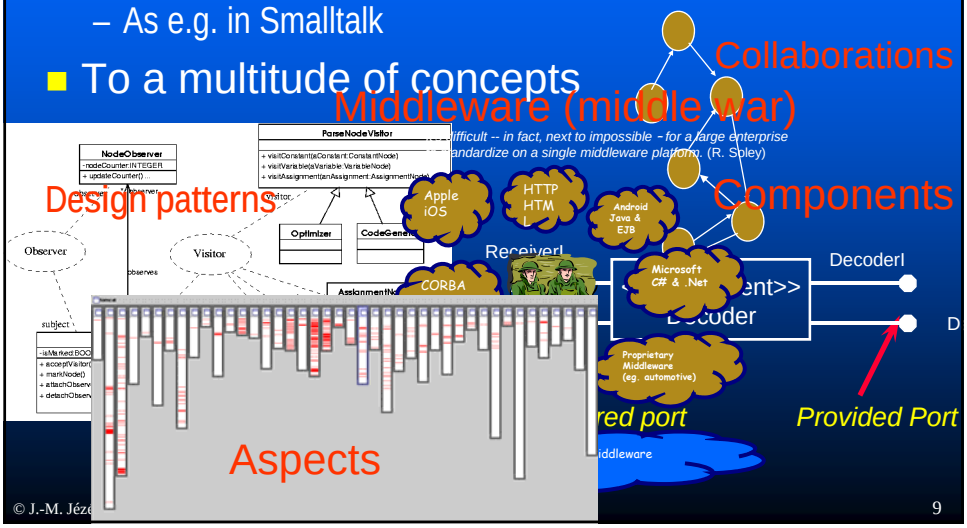
Time to Market!

Practice of Software Engineering



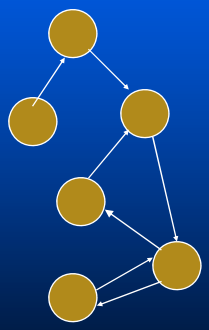
Once upon a time… software development looked simple

- 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)

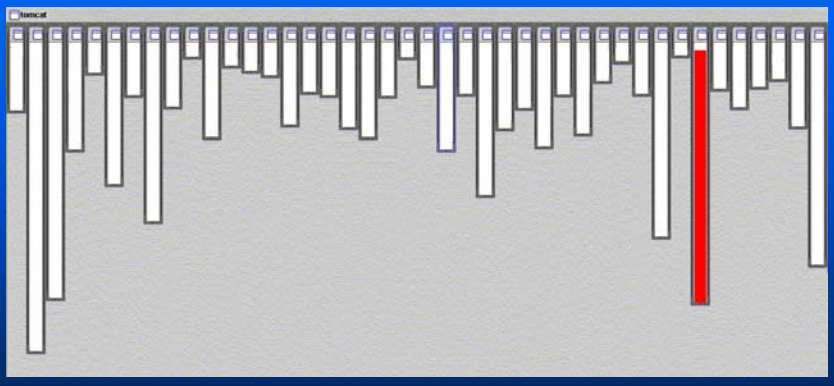
- No clear winner until now
- And probably not in the near future
- Migration is expensive and disruptive
- The OMG tried to send in the ''blue helmets'' with the MDA initiative

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
- Spring AOP complements Spring IoC

good modularity

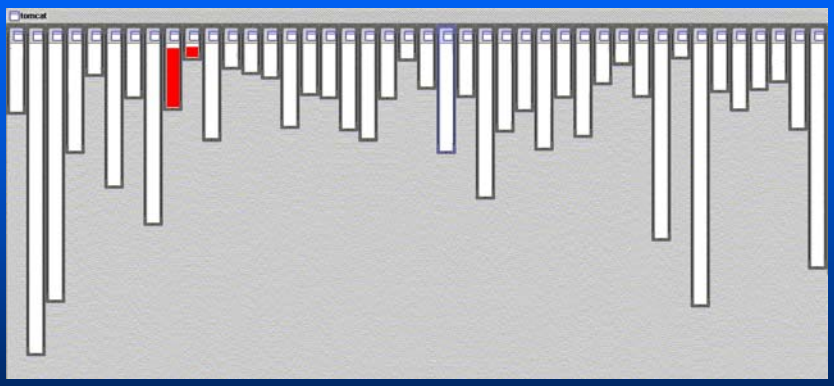
XML parsing



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

good modularity

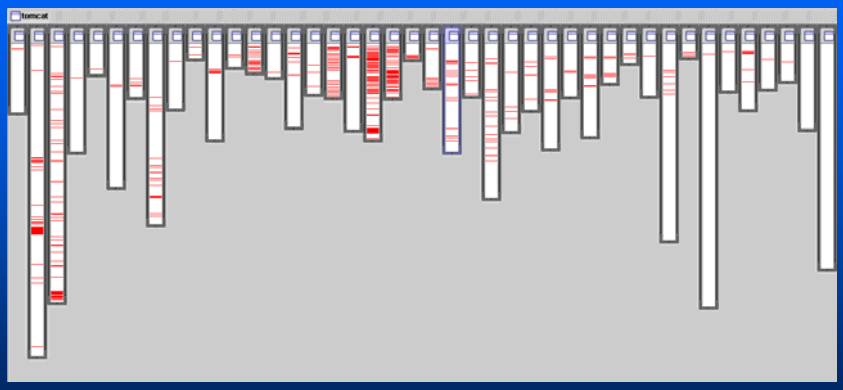
URL pattern matching



- 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

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

AspectJ™ basic mechanisms

- 1 overlay onto Java
  - Dynamic join points
    - » "points in the execution" of Java programs
- 4 additions to Java
  - Pointcuts : specification of joinpoints
    - » pick out join points and values at those points
      - primitive pointcuts : **`call(void Line.setP1(Point))`**
      - user-defined pointcuts
  - Advice
    - » additional action to take at join points in a pointcut
  - Intra-class declarations (aka "open classes")
  - Aspect
    - » a modular unit of crosscutting behavior
      - comprised of advice, intra-class declarations, field,constructor and method declarations

```
pointcut move():
  call(void Line.setP1(Point)) ||
  call(void Line.setP2(Point));
after() returning: move() {
  <code here runs after each move>
}
```

aspectj.org

An Aspect in AspectJ

```
aspect DisplayUpdating {

  pointcut move():
    call(void FigureElement.moveBy(int, int)) ||
    call(void Line.setP1(Point))              ||
    call(void Line.setP2(Point))              ||
    call(void Point.setX(int))                ||
    call(void Point.setY(int));

  after() returning: move() {
    Display.update();
  }
}
```

Spring AOP

- Provides declarative enterprise services, especially as a replacement for EJB declarative services.
  - e.g... declarative transaction management
- Allow users to implement custom aspects, complementing their use of OOP with AOP.

```
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
@Aspect
public class BeforeExample {
  @Before("execution(* com.xyz.myapp.dao.*.*(..))")
  public void doAccessCheck() { // ... }
}
```


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 ?

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

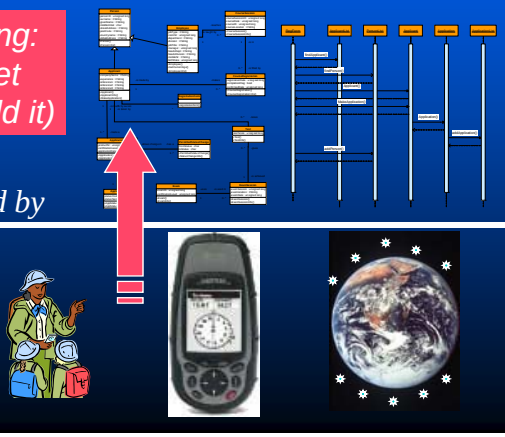
- A Model is a *simplified* representation of an *aspect* of the World for a specific *purpose*

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

M_1 (modeling space)

Is represented by

M_0 (the world)

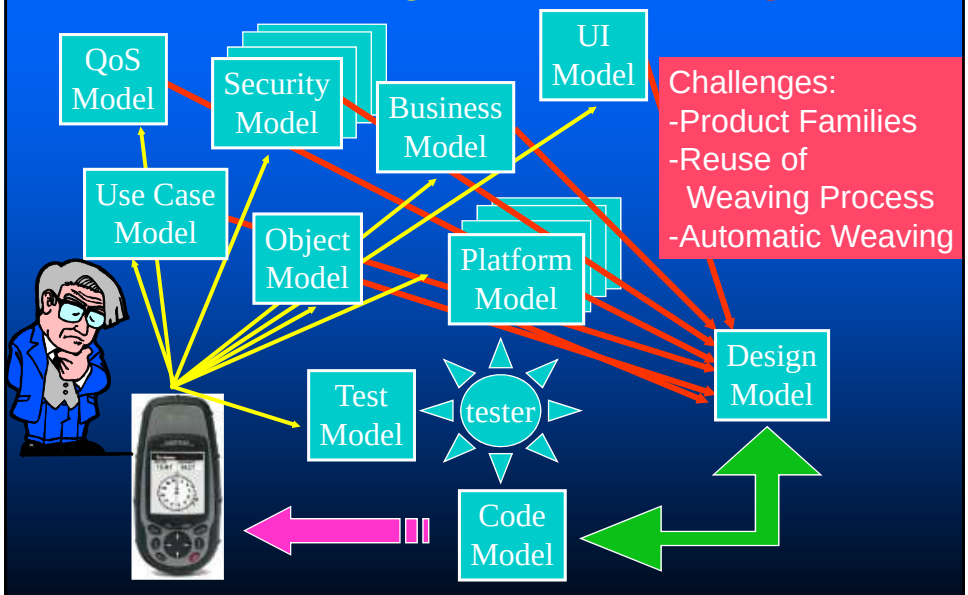


Model and Reality in Software

- Sun Tse: *Do not take the map for the reality*
- Magritte

- Software Models: from contemplative to productive

Modeling and Weaving

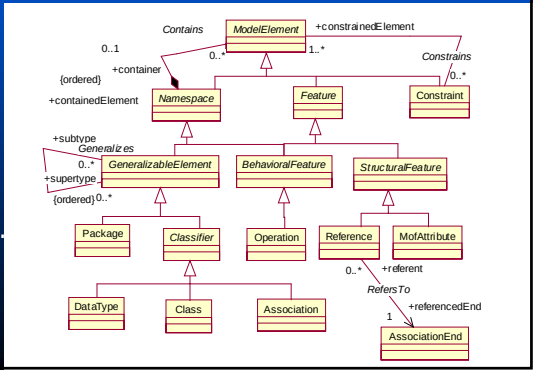


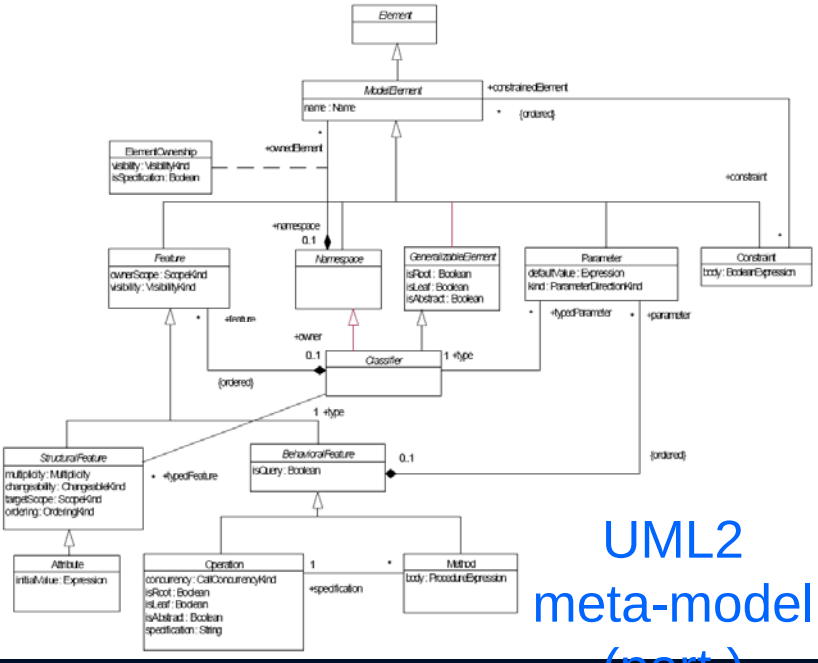
Modeling Languages

- General Purpose Modeling Languages
  - UML and its profiles (MARTE for RT…)
- Domain Specific Modeling Languages
  - Airbus, automotive industry…
  - Matlab/Simulink
- General Purpose Programming Languages
  - With restrictions (not everything allowed)
    - » GWT (Google Web Toolkit)
- Annotations, aspects…
- *In any case, Need for Language Processors*

Assigning Meaning to Models

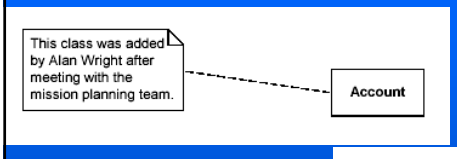
- If a UML model *is no longer* just
  - fancy pictures to decorate your room
  - a graphical syntax for C++/Java/C#/Eiffel...
- Then tools must be able to manipulate models
  - Let's make a model of what a model is!
  - => *meta-modeling*
    - » & meta-meta-modeling..

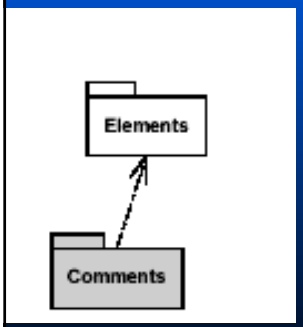


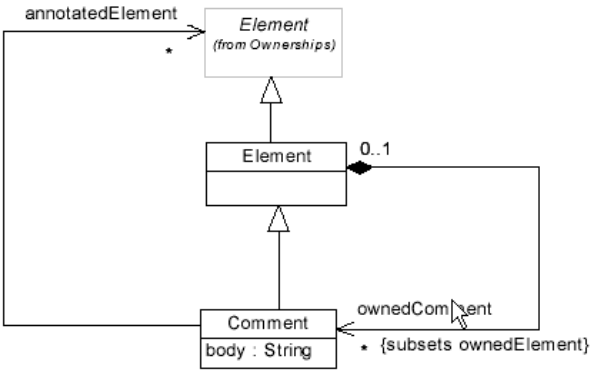


UML2 meta-model (part.)

Zoom: comments







Generalizations

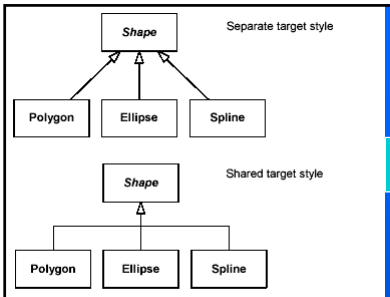


Figure 3-33. Examples of generalizations between classes.

NB: Tell you nothing about:
•generalization being acyclic,
•or semantics of dynamic binding

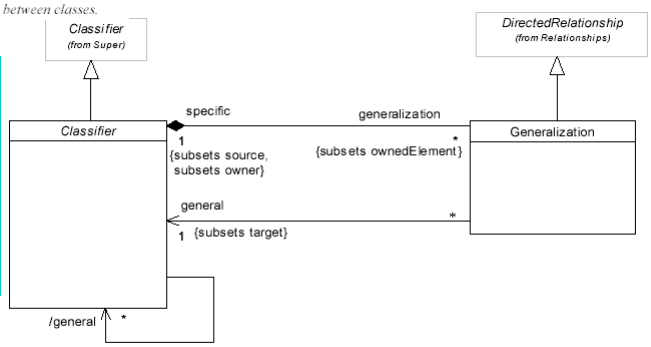
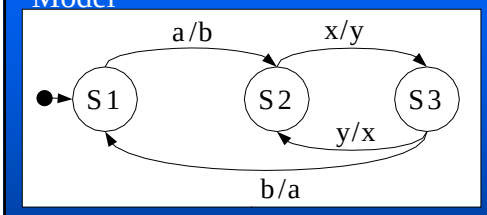
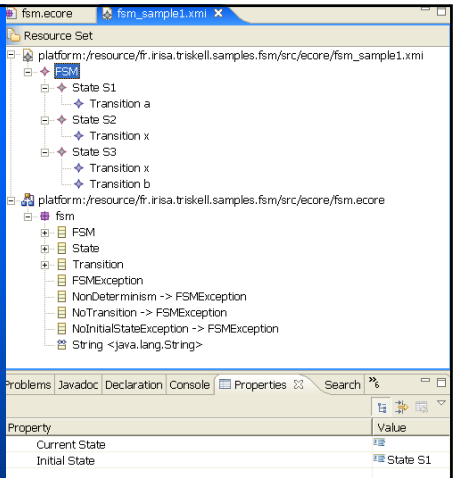


Figure 3-32. The elements defined in the Generalizations package.

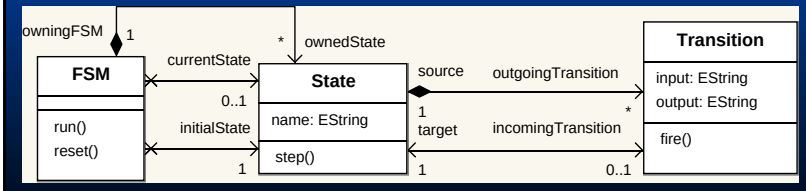
Example with StateMachines

Model

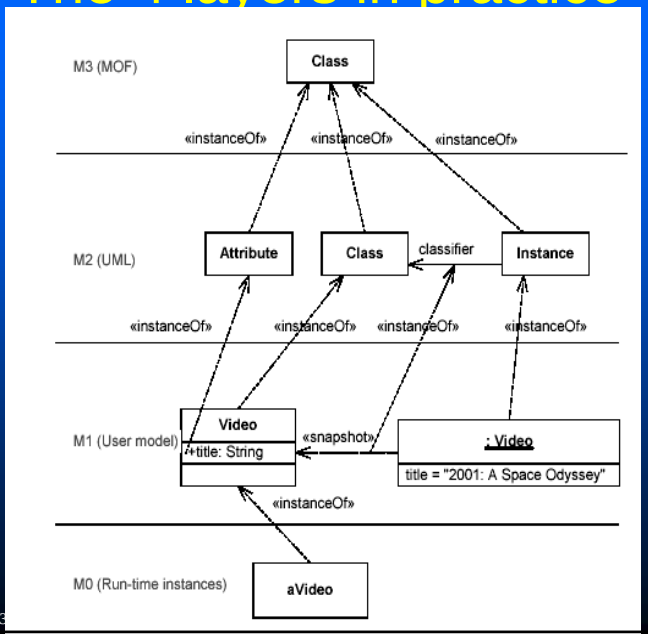




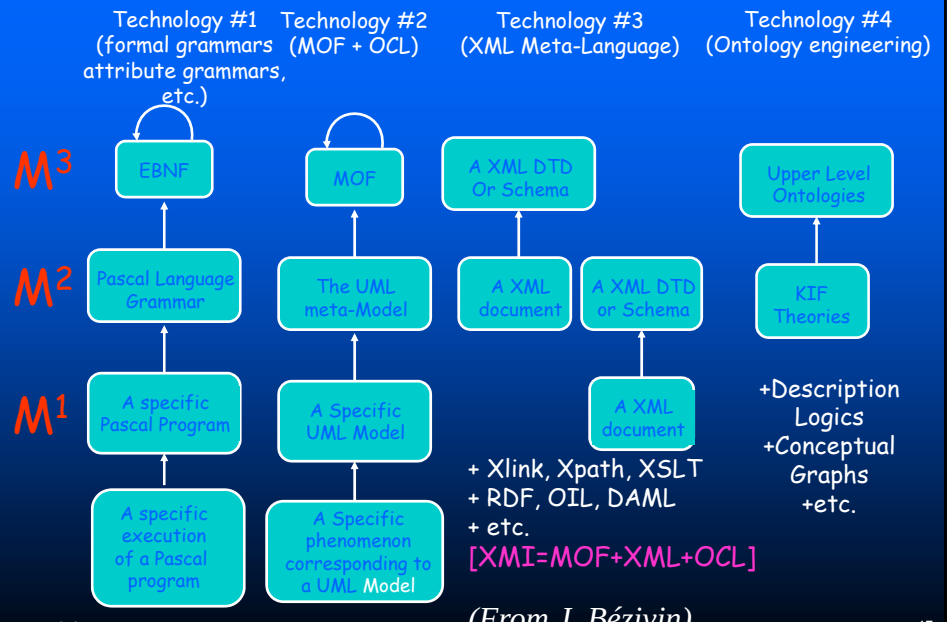
Meta-Model



The 4 layers in practice



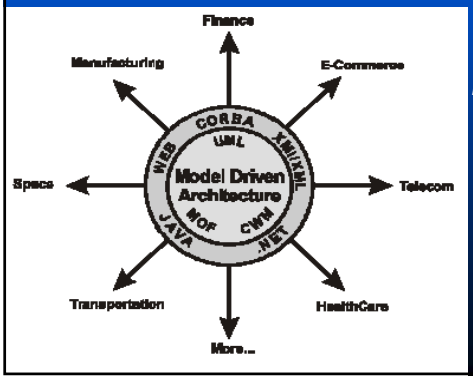
Comparing Abstract Syntax Systems



(From J. Bézivin)

MDA: the OMG new vision

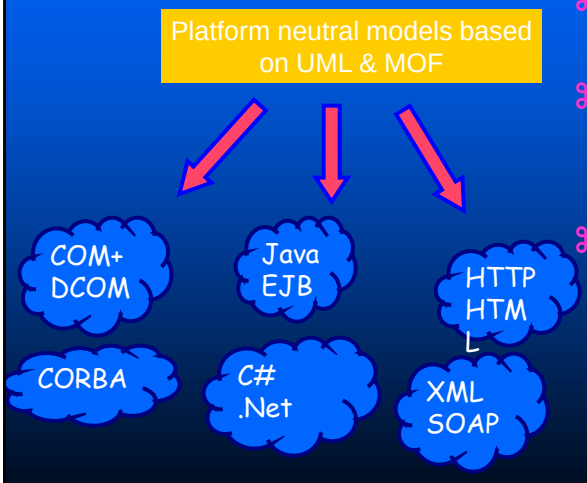
"OMG is in the ideal position to provide the model-based standards that are necessary to extend integration beyond the middleware approach… Now is the time to put this plan into effect. Now is the time for the Model Driven Architecture."



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

46

Mappings to multiple and evolving platforms



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

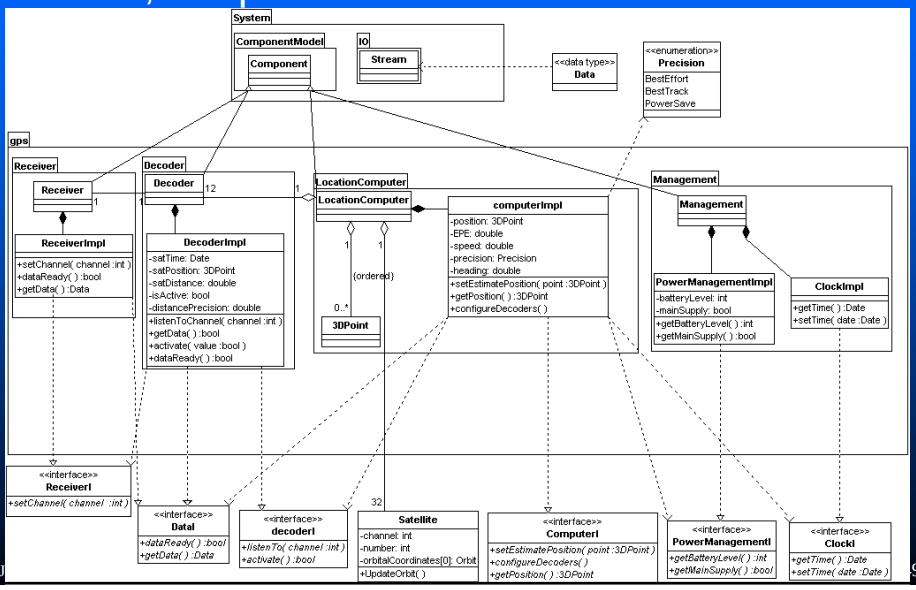
47

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

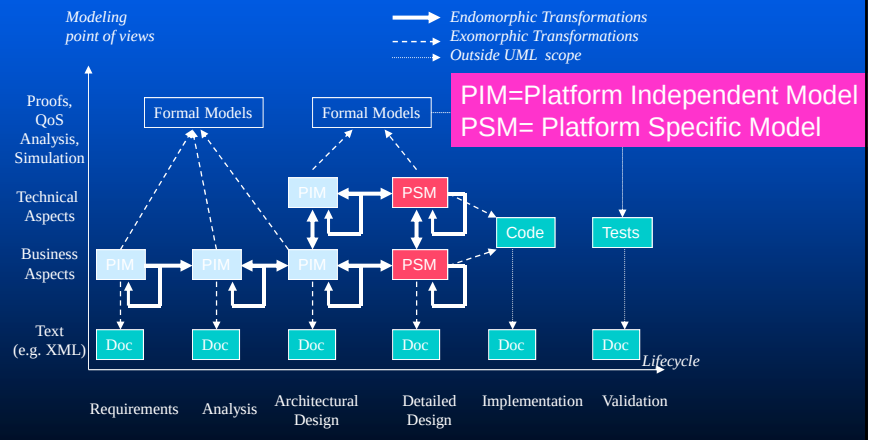


How to go From PIM to PSM?

- "just" weave the platform aspect !
- How can I do that?
  - Through Model transformations
  - OMG Standard: QVT
    - » Query/View/Transformation
  - But many other tools used in practice
    - » Other model transformation tools
    - » but also
      - Annotation processors
      - Language processors (GWT…): cf Angry birds!

Weaving aspects into UML Models?

- It's what Model Driven Architecture is about!

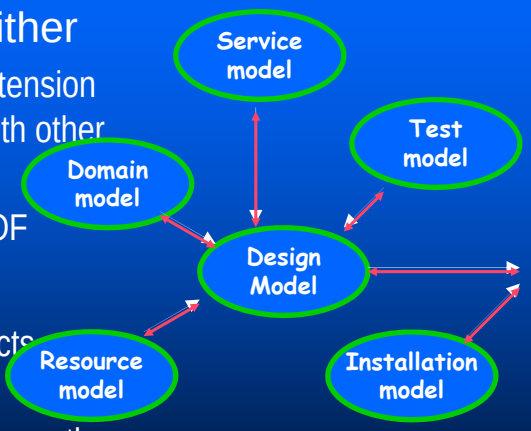


But many more dimensions in modeling!

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

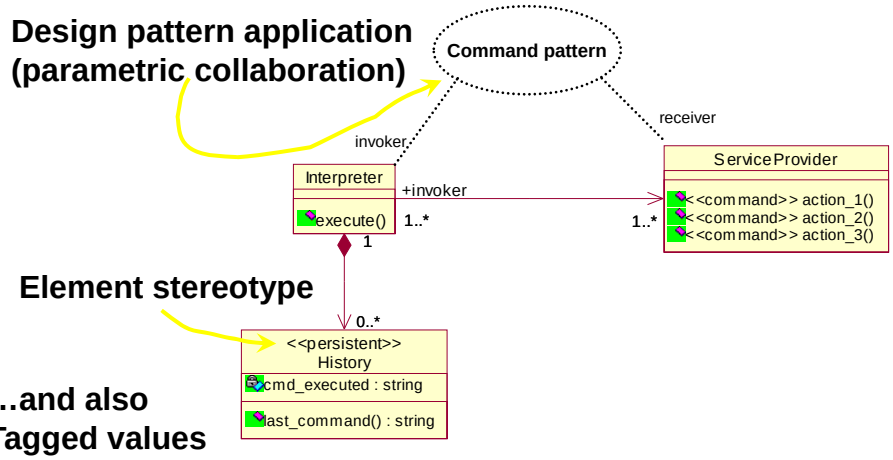
How to take these dimensions into account?

- 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 these aspects (as in e.g. *federations*)



Embedding implicit semantics into a model

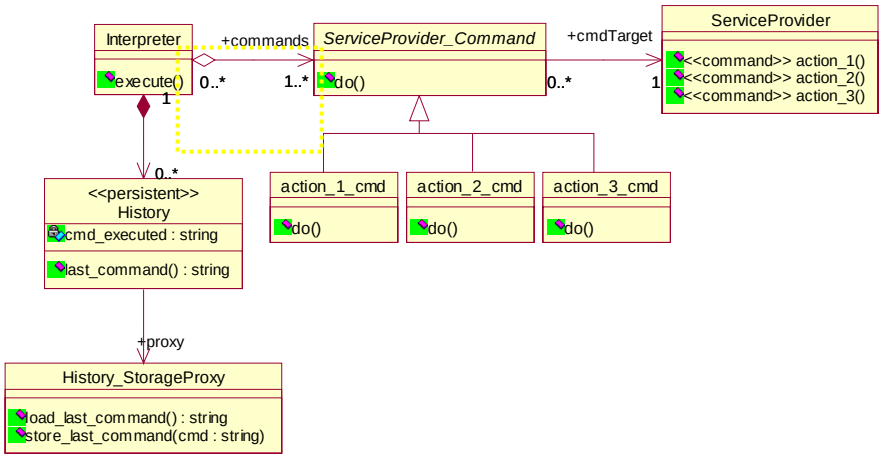
Design pattern application (parametric collaboration)



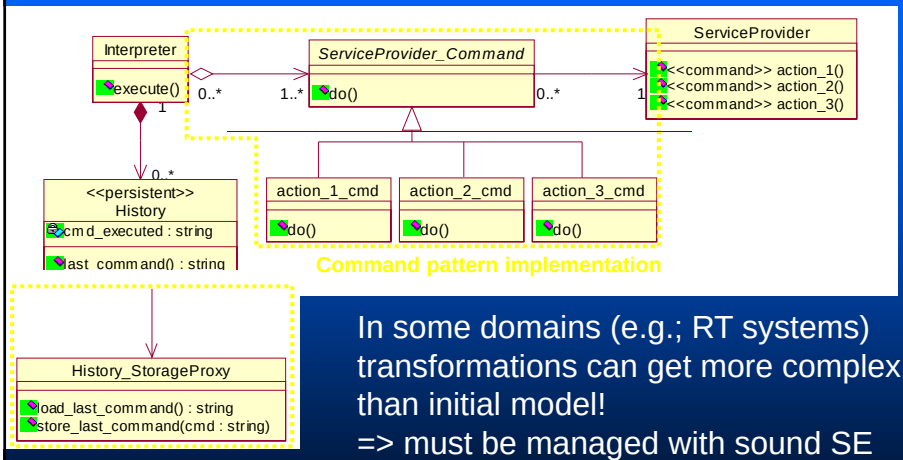
Element stereotype

…and also
Tagged values
& Contracts

…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
  - Now hot topic at OMG with RFP Q/V/T
    - » 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

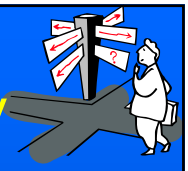
Consequences

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

Preview of the rest of this course

- Meta-Modeling
  - Meta-models & abstract syntaxes
  - Static Semantics with OCL
  - Dynamic Semantics with Kermeta/Xtend
- Model Transformations & code generation
- Applications of MDE
  - Design Pattern Application
  - Model Refactoring
  - Product Line Modeling and Derivation
  - Model based Testing