

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

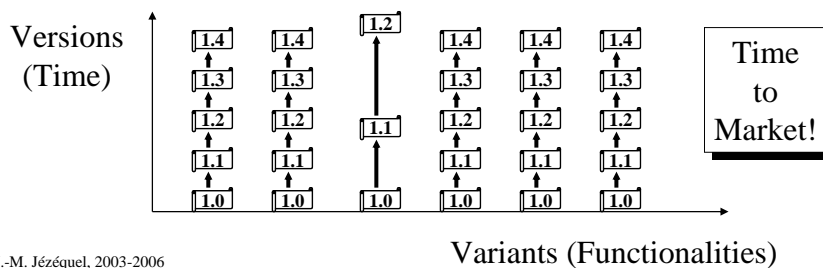
e-mail : jezequel@irisa.fr

<http://www.irisa.fr/prive/jezequel>

Modern Software Problems



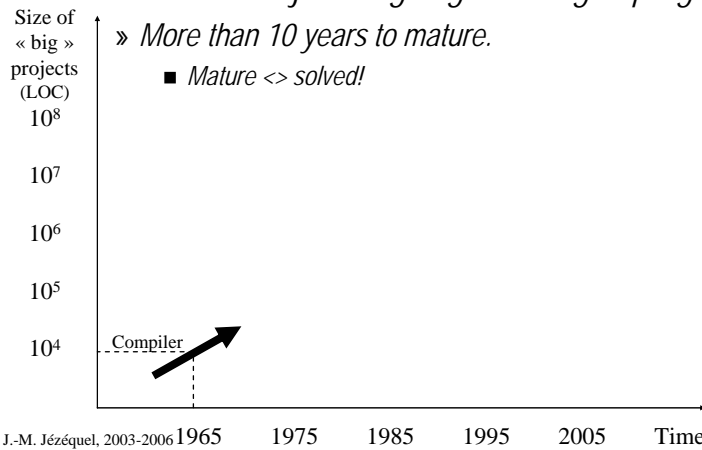
- Importance of non-functional properties
 - distributed systems, parallel & asynchronous
 - quality of service : reliability, latency, performance...
- Flexibility of functional aspects: Product Lines
 - notion of *product lines* (space, time)



Problems addressed in SE

- 1960's: Cope with inherent complexity of software (Correctness)

– Milestone: Floyd '*assigning meaning to programs*'



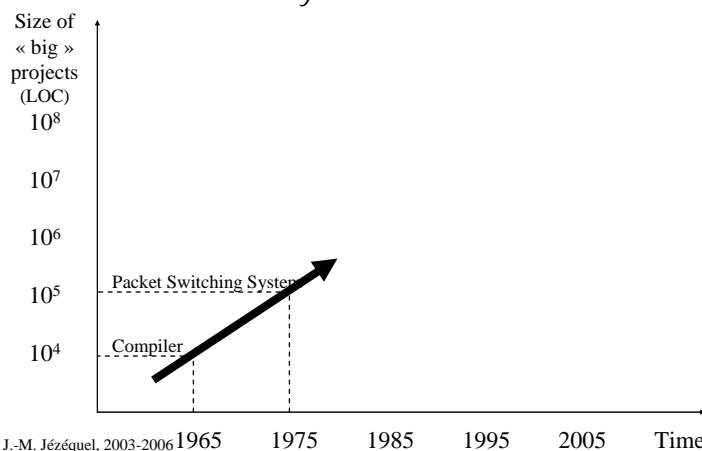
3

Problems addressed in SE

- 1970's: Cope with project size

– Milestone: Parnas, Yourdon: *modularity & structure*

» *More than 10 years to mature*



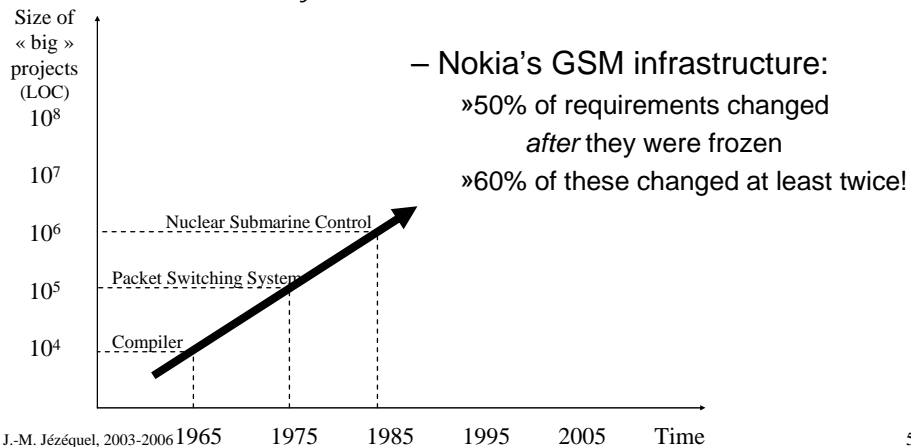
4

Problems addressed in SE

■ 1980's: Cope with variability in requirements

– Milestone: Jackson, Meyer: *modeling, object orientation*

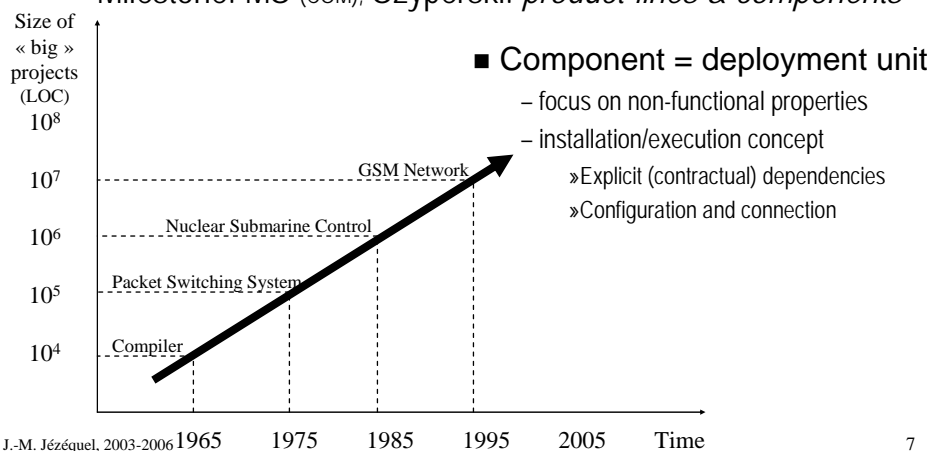
» *More than 10 years to mature*



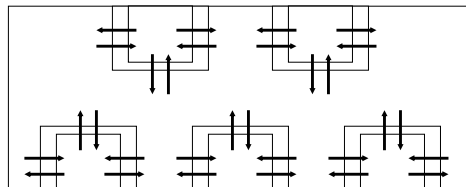
Problems addressed in SE

■ 1990's: Cope with distributed systems and mass deployment:

– Milestone: MS (COM), Szyperski: *product-lines & components*



OO approach: Models and Components



■ frameworks

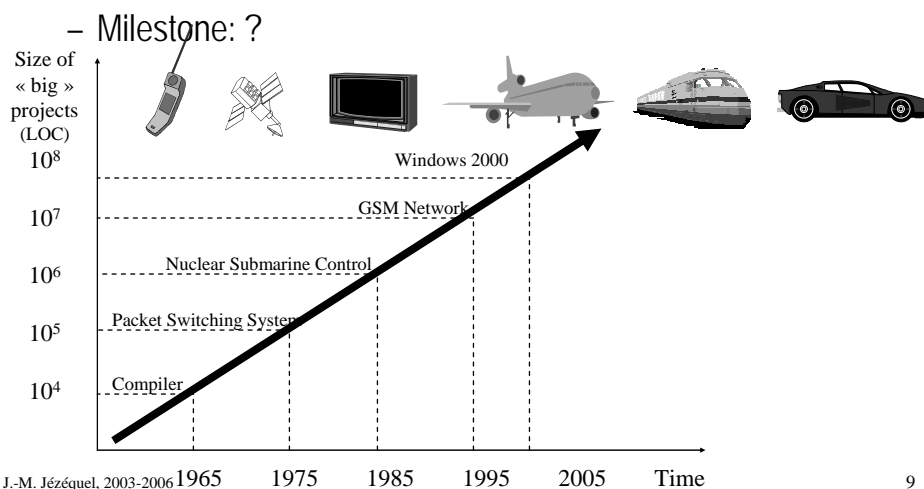
- Changeable software, from distributed/unconnected sources even after delivery, by the end user
- Guarantees ?
Functional , synchronization, performance, QoS

© J.-M. Jézéquel, 2003-2006

8

Problems addressed in SE

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



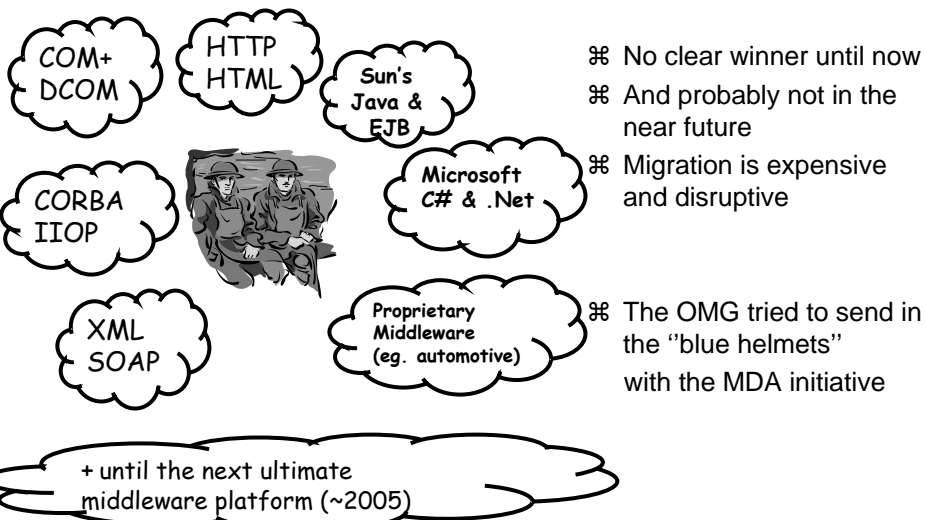
9

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

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)



Aspect Oriented Programming

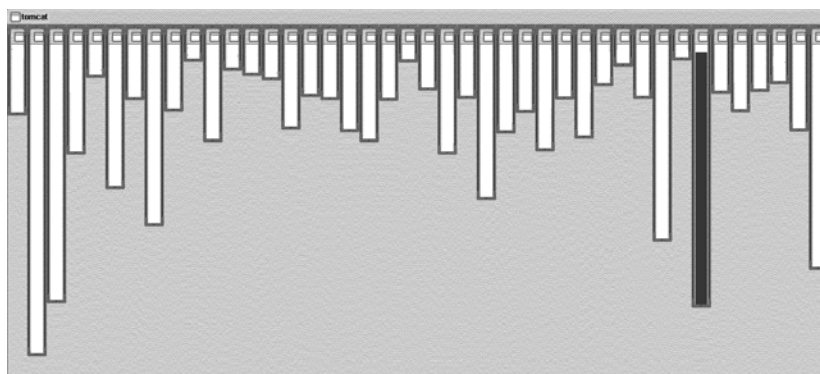
- Kiczales et al., ECOOP'97
 - MIT's one of 10 key technologies for 2010
- Encapsulation of cross-cutting concerns in OO programs
 - Persistence, contract checking, etc.
- Weaving at some specific points (join points) in the program execution
 - *Hence more than macros on steroids*
- AspectJ for AOP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

© J.-M. Jézéquel, 2003-2006

15

good modularity

XML parsing



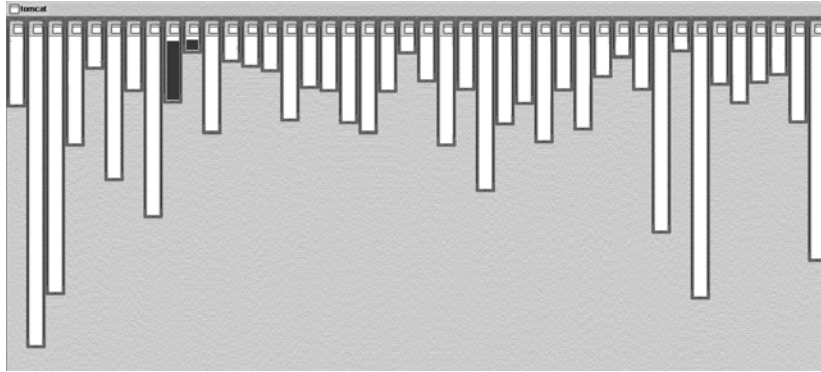
- XML parsing in org.apache.tomcat
 - red shows relevant lines of code
 - nicely fits in one box

© J.-M. Jézéquel, 2003-2006

16

good modularity

URL pattern matching



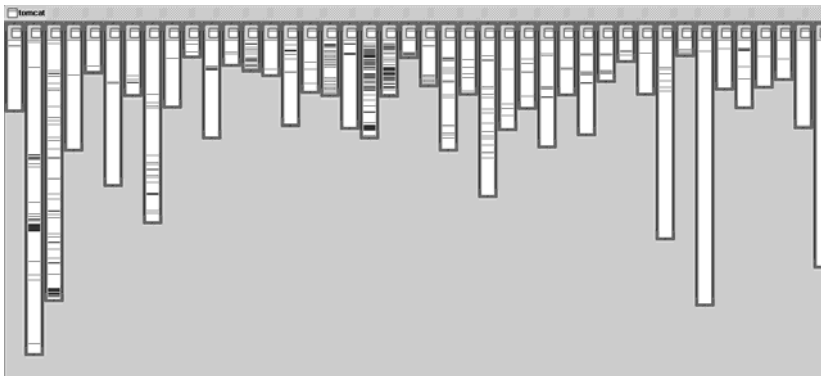
- URL pattern matching in org.apache.tomcat
 - red shows relevant lines of code
 - nicely fits in two boxes (using inheritance)

© J.-M. Jézéquel, 2003-2006

17

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

© J.-M. Jézéquel, 2003-2006

18

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
 - emacs, JBuilder, Forte
- user feedback is driving language design
 - users@aspectj.org
 - support@aspectj.org
- currently at 1.0b1 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

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.

The World and the Model

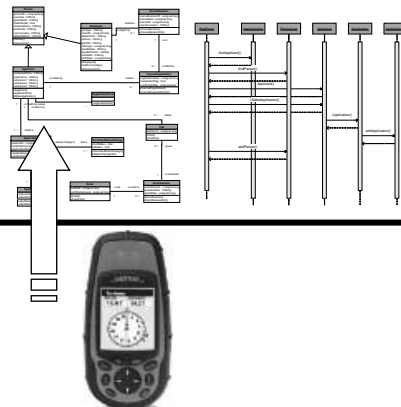
- A Model is a *simplified* representation of an *aspect* of the World for a specific *purpose*
 - Consider modeling both the machine & its environment (M. Jackson)
- UML paved the way from OOP to Model Based SE

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

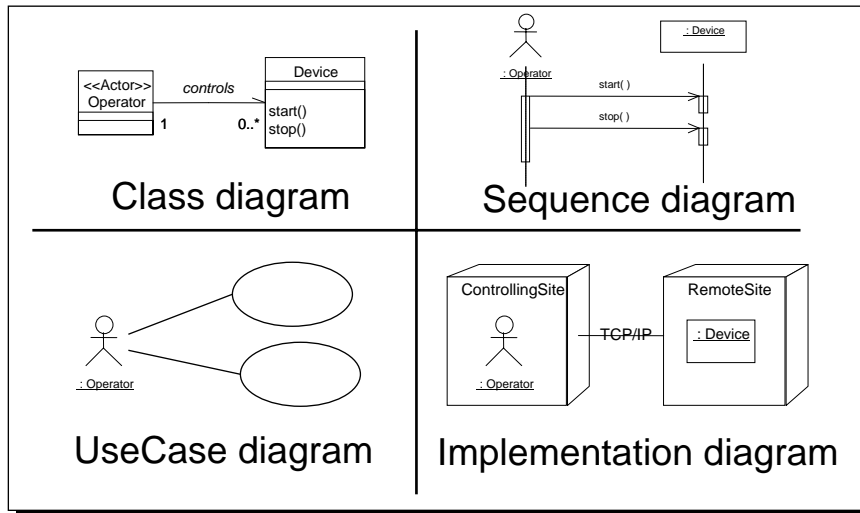
M_1
(modeling
space)

Is represented by

M_0
(the world)



UML: one model, 4 main dimensions, multiple views



© J.-M. Jézéquel, 2003-2006

23

Model and Reality in Software

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

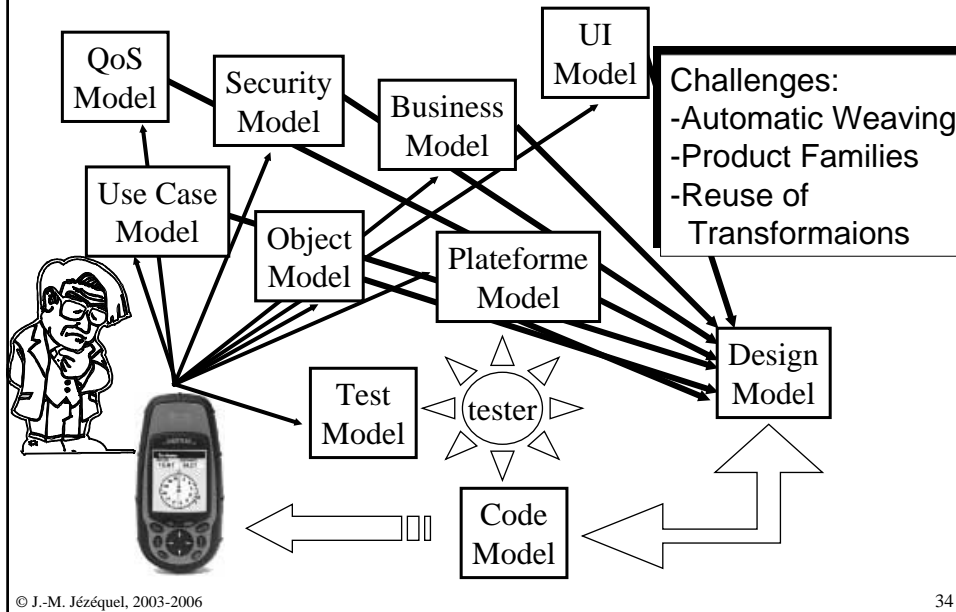


- Software Models: from contemplative to productive

© J.-M. Jézéquel, 2003-2006

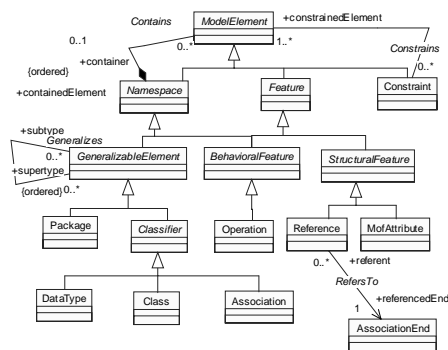
33

Modeling and Weaving

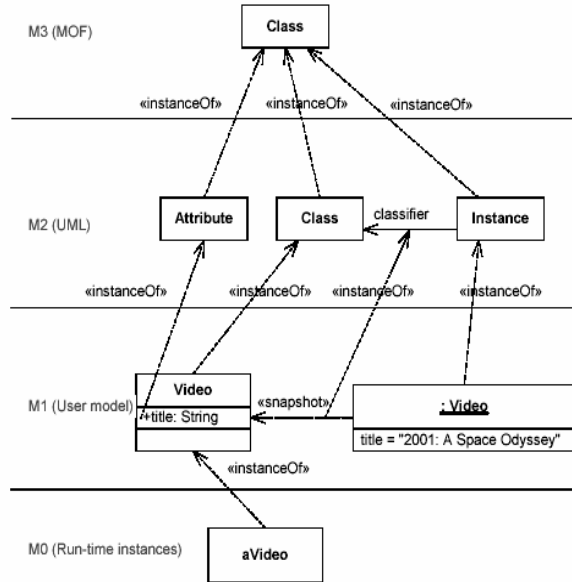


Assigning Meaning to Models

- 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..



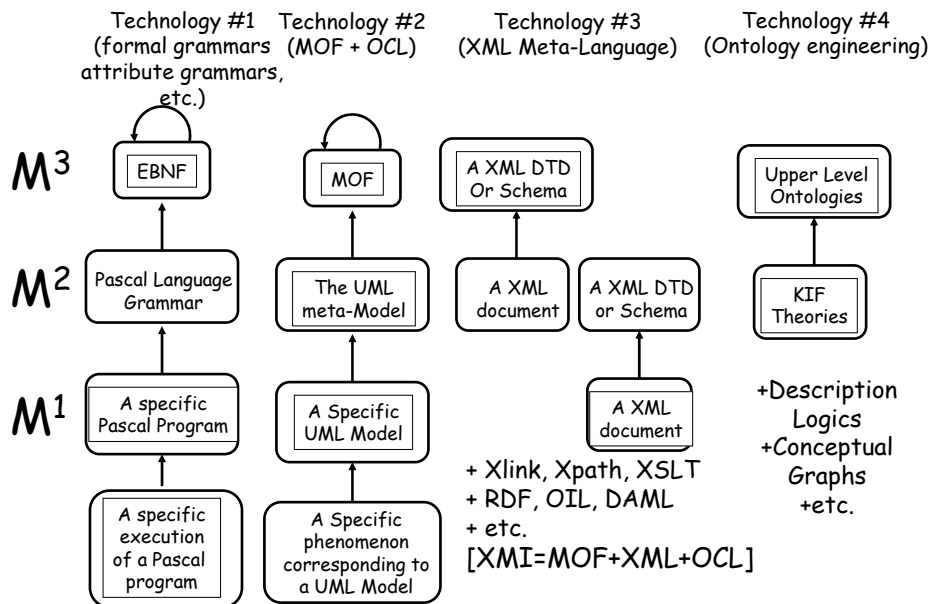
The 4 layers in practice



© J.-M. Jézéquel, 2003-2

37

Comparing Abstract Syntax Systems



© J.-M. Jézéquel, 2003-2006

(From J. Bézivin)

38

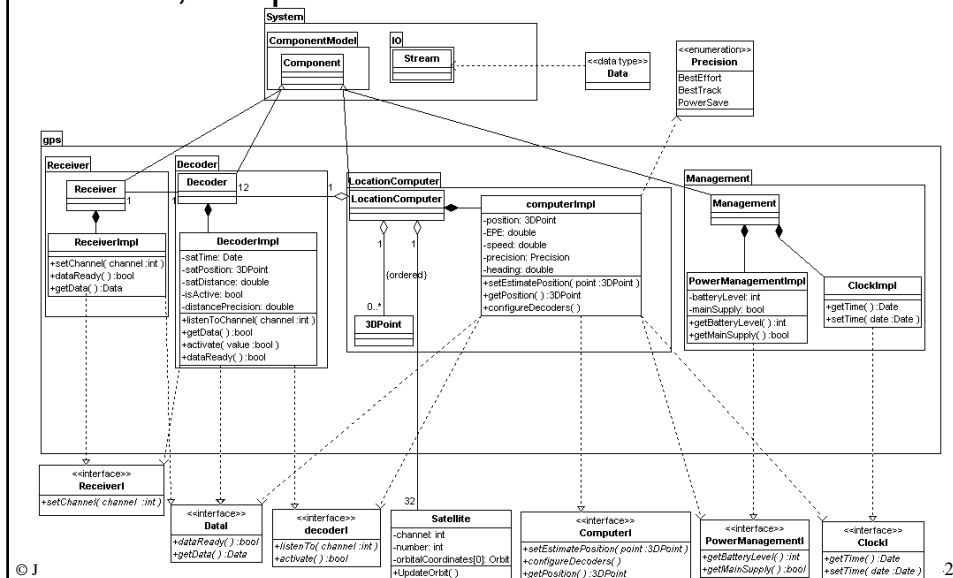
The core idea of MDA: PIMs & PSMs

■ MDA models

- PIM: Platform Independent Model
 - » Business Model of a system abstracting away the deployment 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

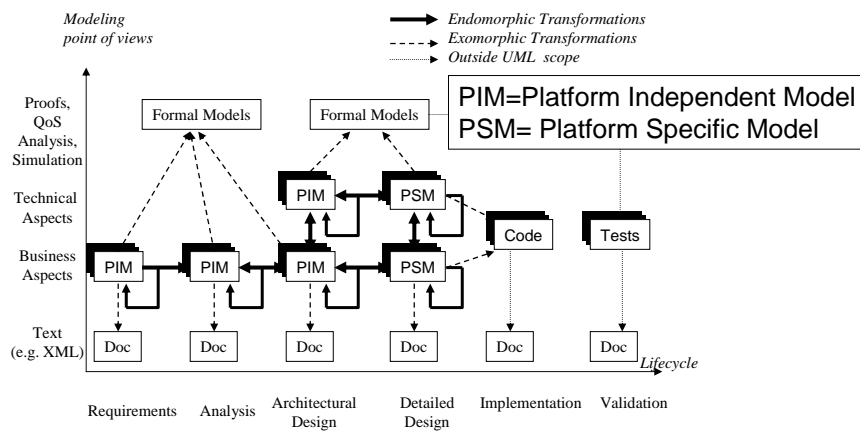


How to go From PIM to PSM?

- "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.

© J.-M. Jézéquel, 2003-2006

45

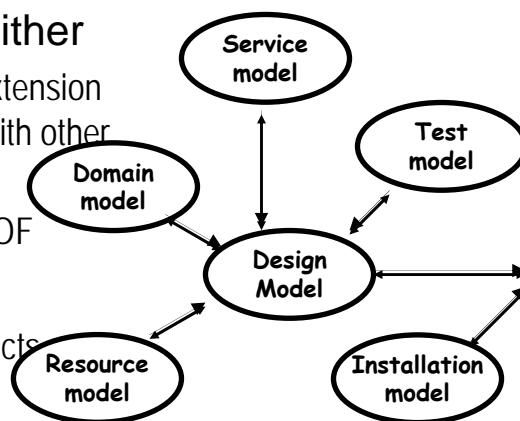
How to take these dimensions into account?

- 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*)

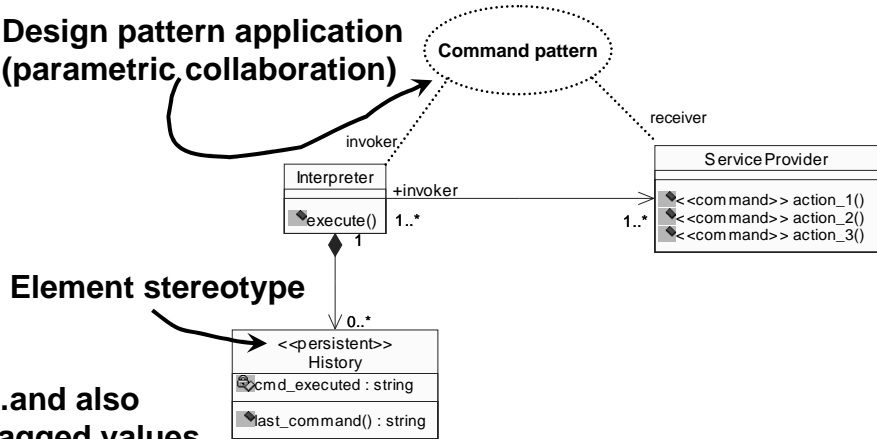


© J.-M. Jézéquel, 2003-2006

46

Embedding implicit semantics into a model

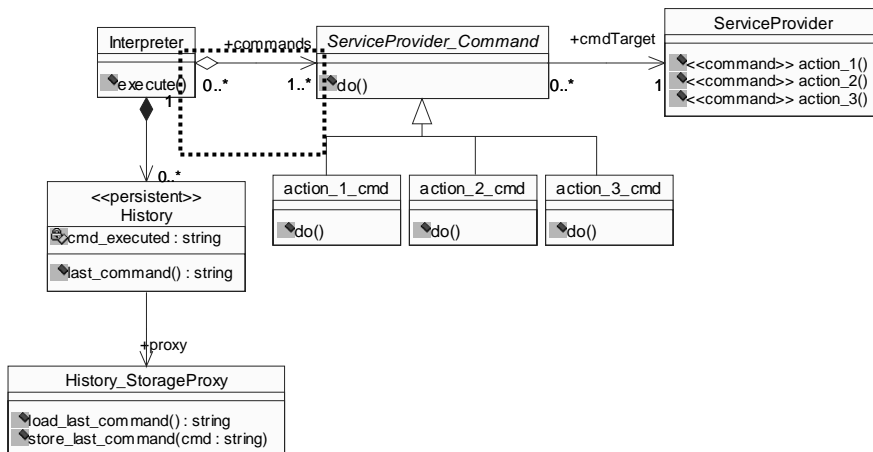
Design pattern application
(parametric collaboration)



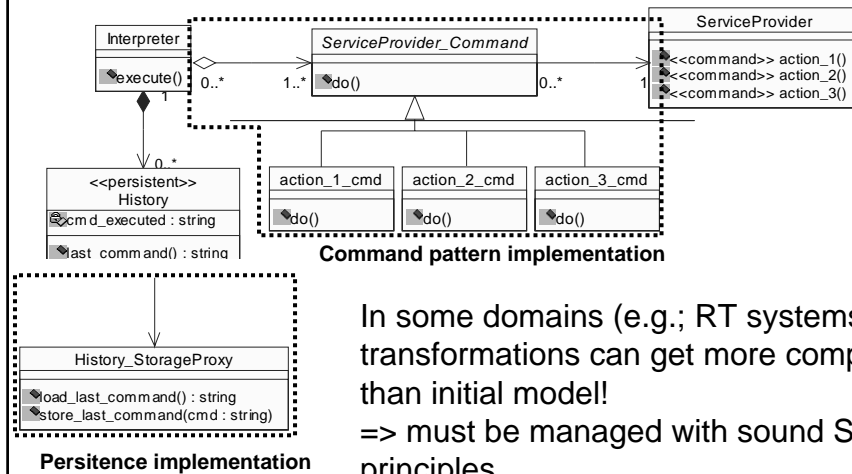
Element stereotype

...and also
Tagged values
& Contracts

...and the result we want...



How To: Automatic Model Transformations



© J.-M. Jézéquel, 2003-2006

49

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!

© J.-M. Jézéquel, 2003-2006

50

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?

© J.-M. Jézéquel, 2003-2006

51

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

© J.-M. Jézéquel, 2003-2006

52

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

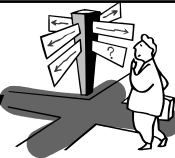
Challenges

- 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

© J.-M. Jézéquel, 2003-2006

55

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

© J.-M. Jézéquel, 2003-2006

56

Preview of the rest of this course

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