

Model-Driven Engineering: Core Principles and Challenges*

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>

**Joint work with J. Bézivin (INRIA/IRIN-Univ. Nantes) and B. Rumpe (Irisa & TUM)*

Outline

■ Context

- a brief and partial historical perspective on practical software engineering

■ Modeling and meta-modeling

- What the MDA is really about

■ Model transformations

- reflective MDE

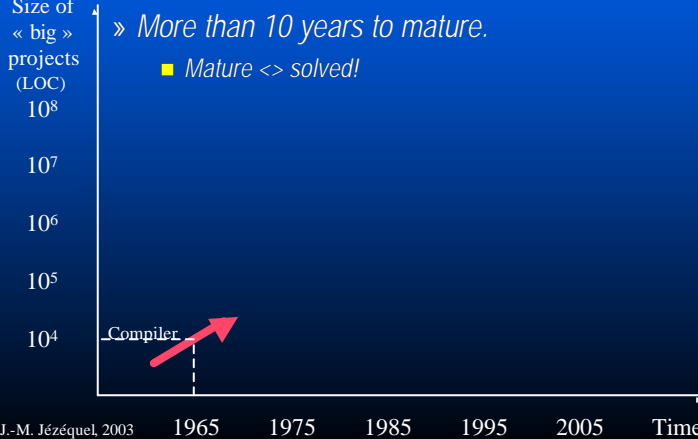
■ Challenges

- An example linking MDE with Components, Contracts, and Aspects.

Problems addressed in SE

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

– Milestone: Floyd 'assigning meaning to programs'



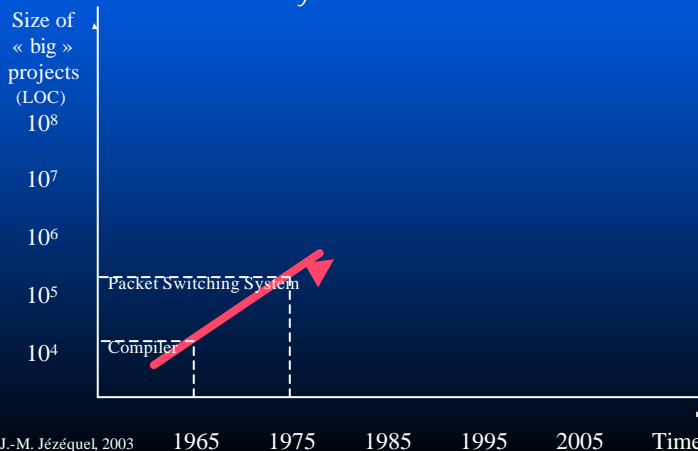
3

Problems addressed in SE

- 1970's: Cope with project size

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

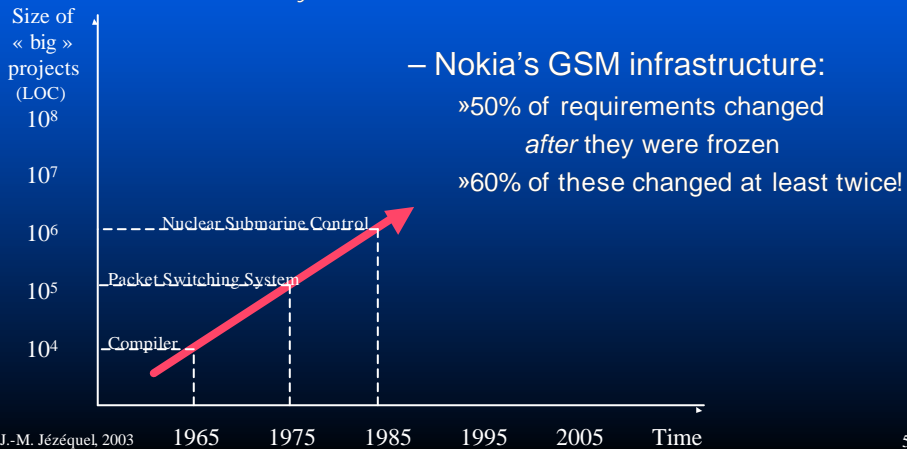
» More than 10 years to mature



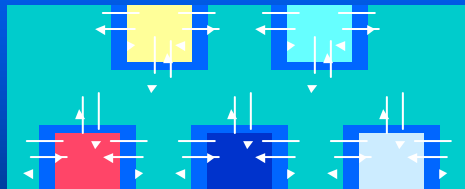
4

Problems addressed in SE

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



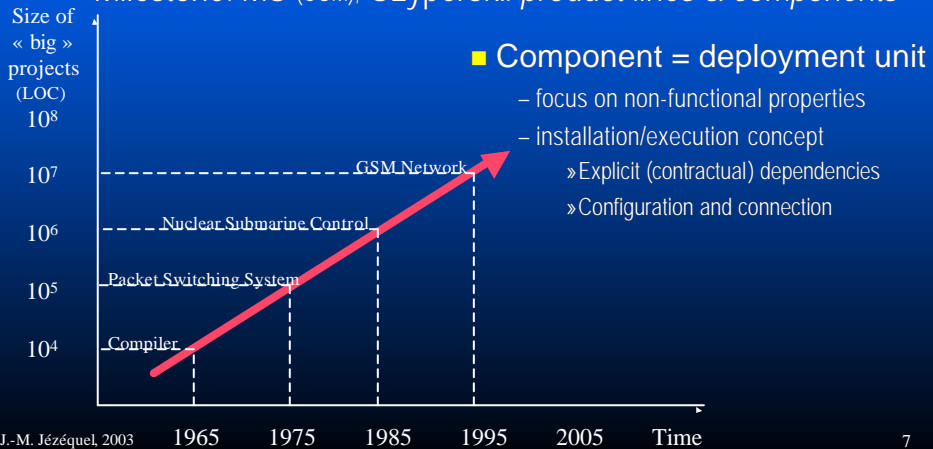
OO approach: frameworks



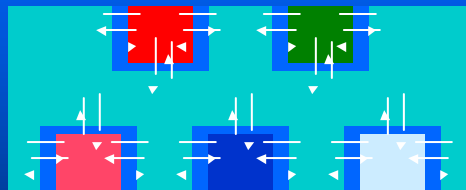
Problems addressed in SE

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

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



Components



- *Enabling technology*

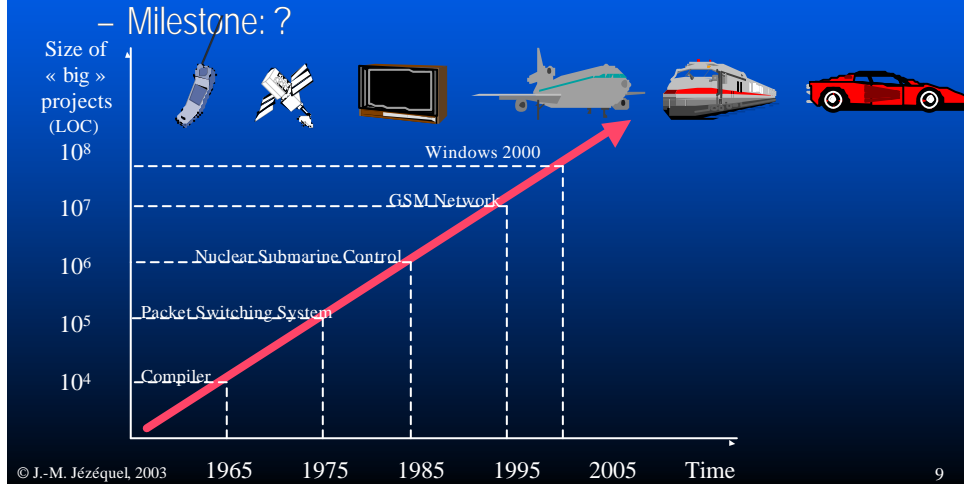
- Changeable software, from distributed/unconnected sources even after delivery, by the end user
- Guarantees ?

Functional, synchronization, performance, QoS

Expressed using the notion of "contract"

Problems addressed in SE

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



Formal Methods: How do they help?

- Good at solving 1960's problems
 - Plus some concurrency/distribution issues
- FM do work extraordinary well in some settings:
 - E.g. with frozen requirements
 - and/or when platform is close to FM underlying semantics (Esterel and circuits, SDL & Telephony, etc.)
- But FM adoption by industry at large never took off
 - not just because engineers are stupid or not educated well enough!

The standard of correctness and completeness necessary to get a computer program to work at all is a couple of orders of magnitude higher than the mathematical communities standard of valid proofs.

William P. Thurston, "On Proof and Progress in Mathematics" Bulletin of the American Mathematical Society, v. 30, n. 2, Apr. 1994

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.

"The Nature of Modeling." **Jeff Rothenberg**.
in *Artificial Intelligence, Simulation, and Modeling*,
L.E. William, K.A. Loparo, N.R. Nelson, eds.
New York, John Wiley and Sons, Inc., 1989, pp. 75-92

<http://poweredge.stanford.edu/BioinformaticsArchive/PrimarySite/NIHpanelModeling/RothenbergNatureModeling.pdf>
© J.-M. Jézéquel, 2003 13

The World and the Model

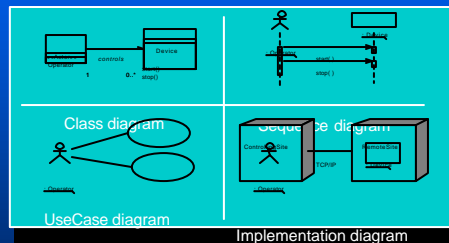
A Model is a *simplified* representation of an aspect of the World

Consider modeling both the machine & its environment (M. Jackson)

M_1
(modeling
space)

Is represented by

M_0
(the world)



UML paved the way...

From Object-Oriented Programming
to
Model-Based Software Engineering



© J.-M. Jézéquel, 2003

(From J. Bézivin)

15

Model and Reality in Software

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

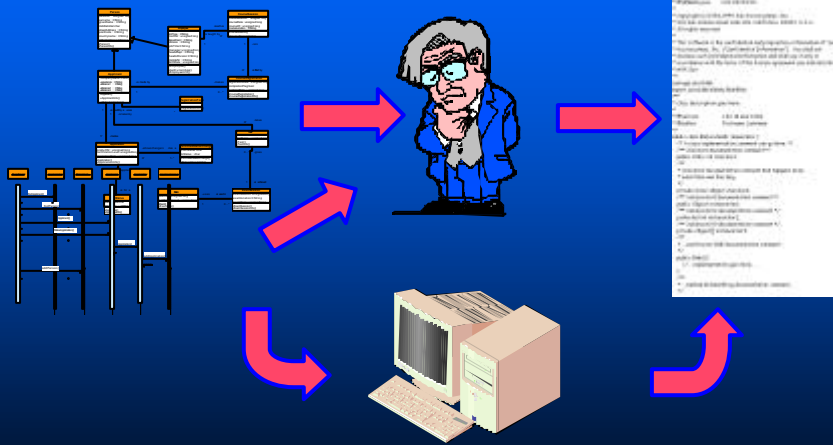


- But models are software...
 - And conversely!

© J.-M. Jézéquel, 2003

16

Models: from contemplative to productive



"from human-readable to computer-understandable"

© J.-M. Jézéquel, 2003

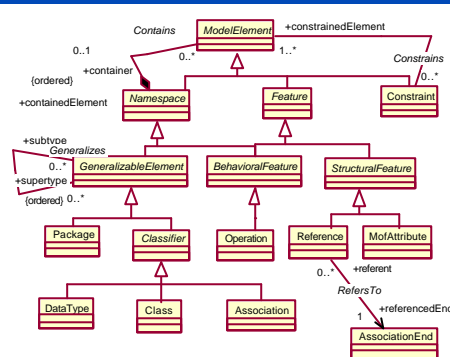
From J. Bézivin

17

Assigning Meaning(s) to Models

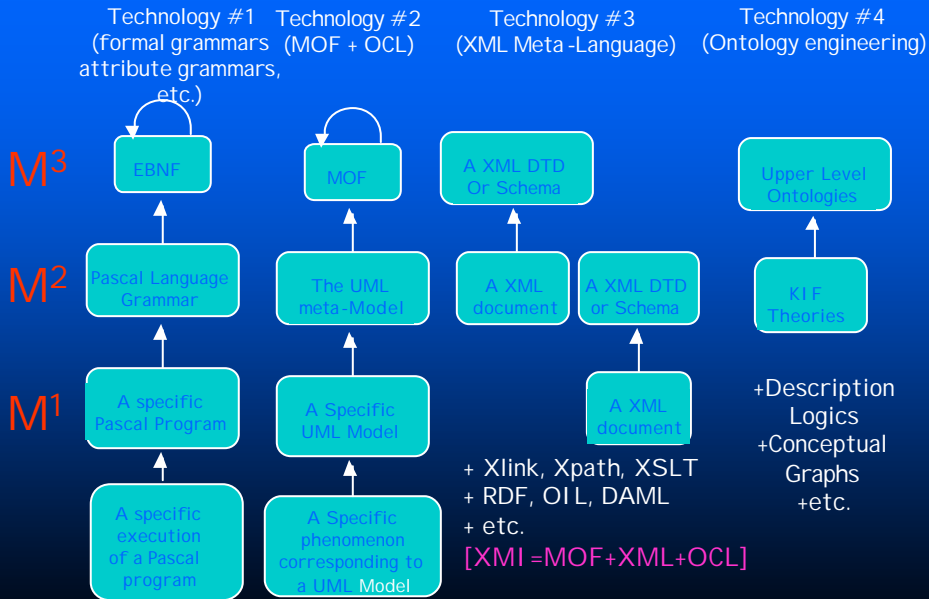
- If a UML model *is no longer* just
 - fancy pictures to decorate your room
 - a graphical syntax for C++/Java/C#/Eiffel...
- Then tools must be able to manipulate models

- Let's make a model of what a model is!
- semantic variation points
- => **meta-modeling**
» & meta-meta-modelling..



© J.-M. Jézéquel, 2003

Comparing Abstract Syntax Systems



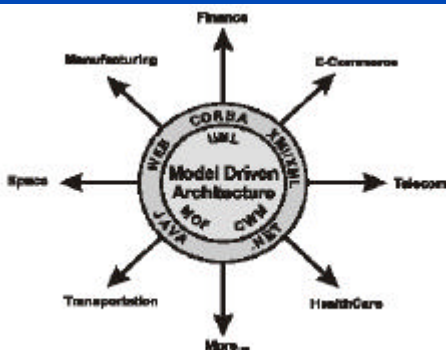
© J.-M. Jézéquel, 2003

(From J. Bézivin)

19

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."



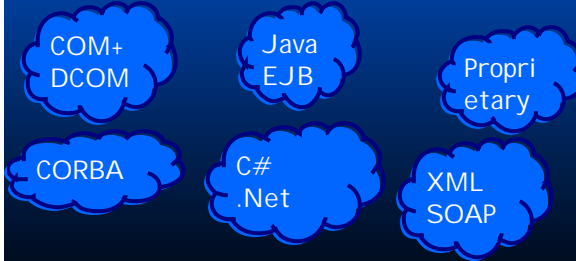
Richard Soley & OMG staff,
 MDA Whitepaper Draft 3.2
 November 27, 2000

To be taken with
 a grain of salt;-)

20

Mappings to multiple and evolving platforms

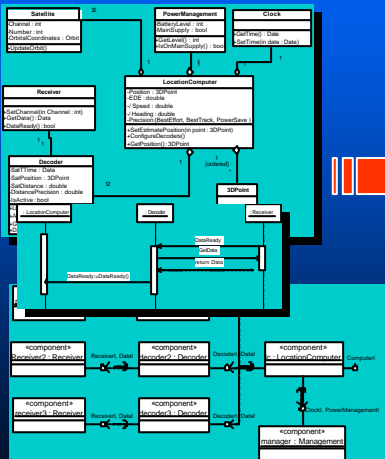
Platform neutral models based on UML & MOF



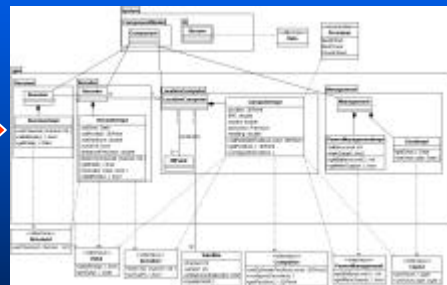
- ⌘ MOF & UML as the core
- ⌘ Organization assets expressed as models (PIM)
- ⌘ Model transformations to map to technology specific platforms (PSM)

How to go from PIM to PSM?

PIM Model for a GPS

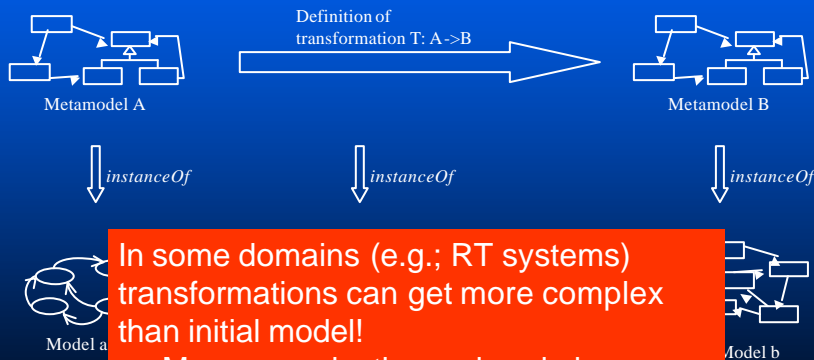


.NET PSM Model for a GPS



How to go From PIM to PSM?

- "Just" weave the platform aspect !
- How to? Through Model transformations



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?

Reflective MDE

- How to express transformations independently from a specific CASE tool?
 - Unacceptable risk to tie transformation assets to a specific CASE tool + multiple tools among BUs anyway
- The CASE tool is the platform for executing the transformations
- Apply MDA to transformations!
 - Platform Independent Transformation (PIT)
 - » Expressed as a model
 - Platform Specific Transformation (PST)
 - » Transformations to go from PIT to PST

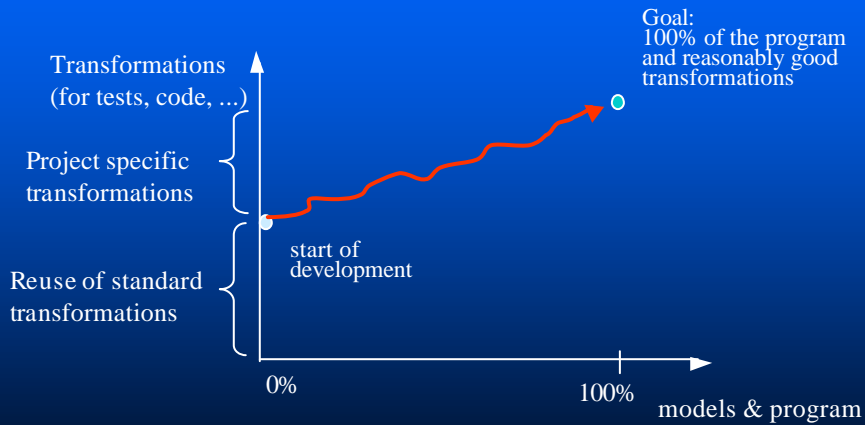
Model the *Model-Transformations* with the UML

- **Class diagrams**
 - A transformation rule is expressed as an operation
 - » OCL for pre/post + navigation
 - A set of rules is a class (module+information hiding)
 - Variability builds on subclassing and dynamic binding +DPs
- **Model management diagrams**
 - Packages, components => reusable assets
- **Activity diagrams**
 - Dependencies between subtasks
- **Deployment diagrams**

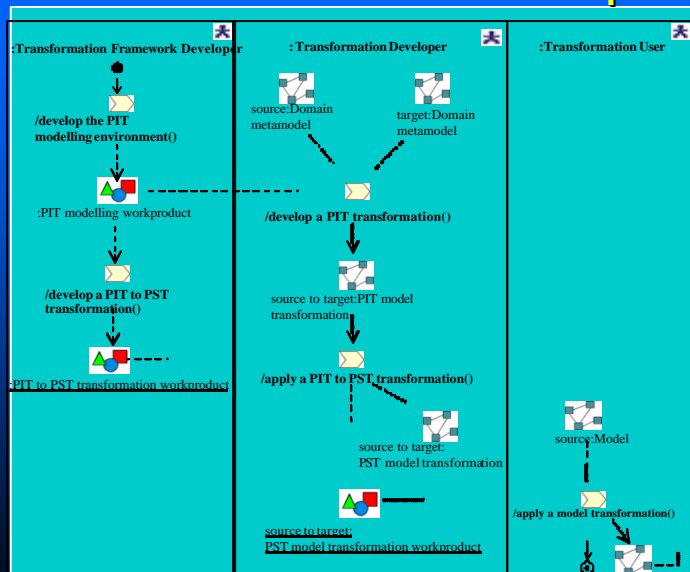
Transformations are Assets => apply sound SE principles

- **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

Two Dimensions in Development



3 roles in software development



Principles

1. Everything relevant to the development process is a model
2. All the meta-models are written in a language of a unique meta-meta-model
3. A development process can be modeled as a partially ordered set of model transformations, that take models as input and produce models as output

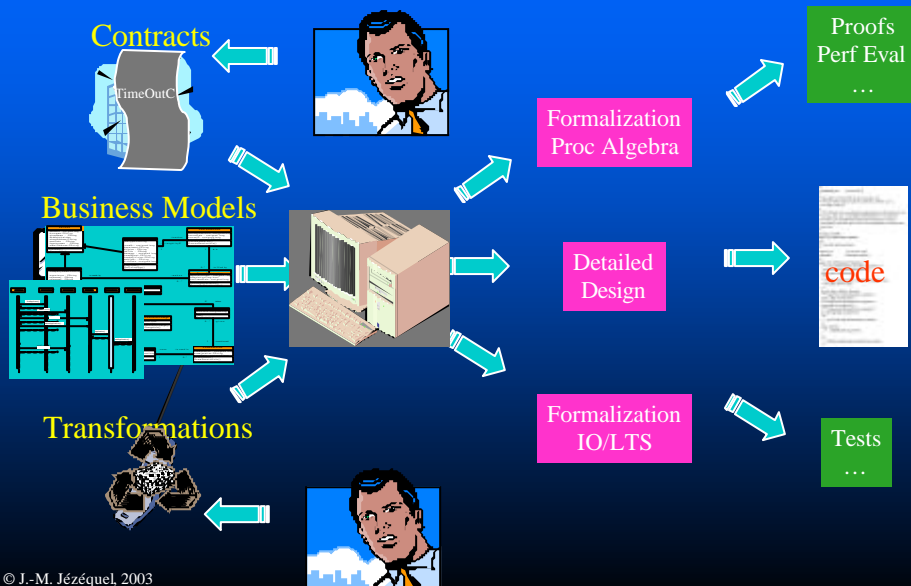
Consequences

1. Models are aspect oriented. Conversely: Aspects are models
2. Transformations are models
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, connection with repository...
- Software Engineering Issues
 - From requirements to tests and SCM
- Leverage Results from Formal Methods
 - Let's detail this one...

MDA & Formal Methods



Example for the GPS

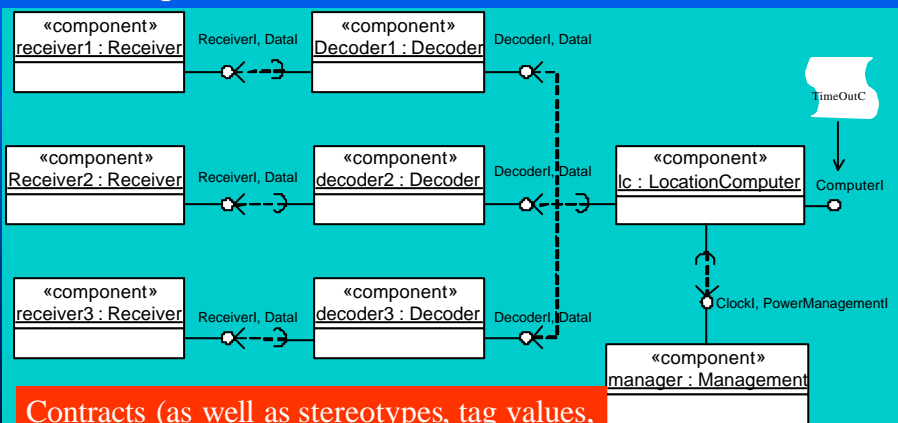
- Getting location data from a receiver should be done quickly enough
 - Can take a long time in case of radio reception problems
 - Big power consumption while the receiver is active
- TimeOut contracts for the GPS
 - Just one QoS dimension
 - » Name = responseTime
 - » Type = int
 - » Direction = down
 - » Unit = us



The semantics is in the eye of the beholder

Example: adding a contract

- Adding QoS contracts to our GPS device



Contracts (as well as stereotypes, tag values, DP applications) are annotations whose semantics is *not* in the model.

Object Contracts vs. Component contracts

- Component Based Systems are not layers of functionalities
 - abstraction ? hiding: *you cannot (completely) hide the platform*
- *Provided but also required* contracts
 - Engagements valid only if *clients and providers* observe their own ones
- Most offered contracts explicitly depend upon required ones
 - E.g. response time depends on platform spec
 - And even for objects, this can happen (callback)

Examples of contract dependencies in the GPS

- The *TimeOutContract* on the **LocationComputer** depends on *TimeOutContracts* from the active **Decoders**
- The *TimeOutContract* on the **Decoder** depends on a *ReceptionQuality* contract on the **Receiver**
 - Monitoring the quality of the reception of satellite data
 - Known at runtime only in this case

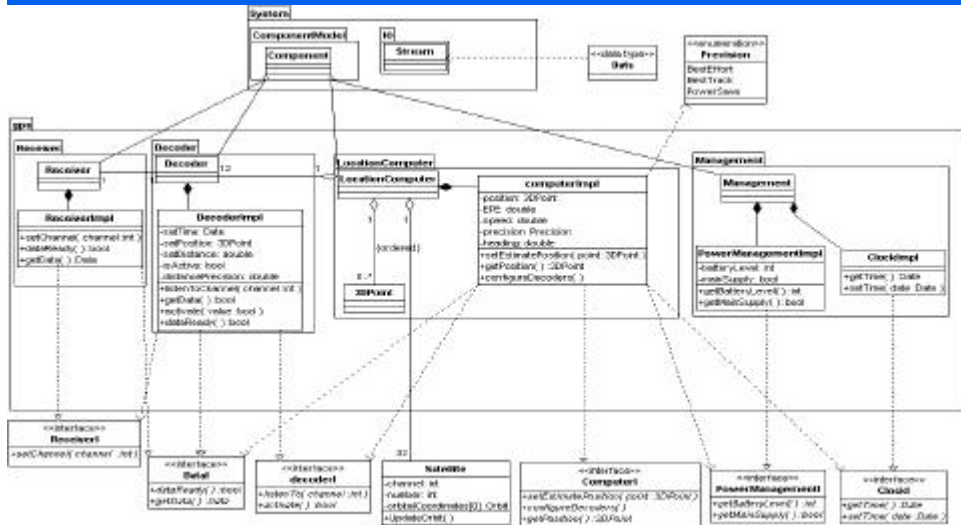
Contract space

- A component actually offers a *range of contracts*
 - One contract will be enforced (hopefully)
 - Depending on the obtained required contracts
 - At binding time or at run-time
- Many possible ways to exploit this information:
 - Logical deduction
 - » Top-down = dimensioning
 - » Bottom-up = end to end QoS
 - Contract checking

Contract Management is Crosscutting

- Contract Description
- Contract Subscription, Termination
- Contract Checking
 - static/dynamic, sequential/concurrent/distributed...
 - Level of Service actually provided
- Dealing with Contract Violations
 - ignore, reject, wait, negotiate ...

How to implement these contracts for this .NET PSM?



© J.-M. Jézéquel, 2003

41

Weave contract management

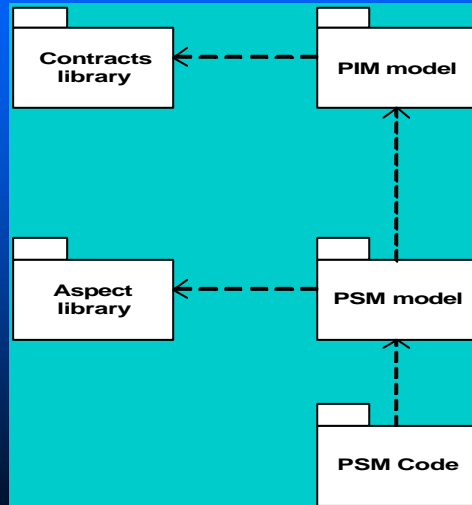
- Problem: it depends on the semantics of each contract type
 - QML does not capture the semantics
 - Sometimes quite complicated
 - » E.g. bounded throughput variation implies non-instantaneous monitoring and the collecting of statistics
 - » May heavily depends on the platform!
- There exist known solutions to these problems
 - Often semi-formalized as design patterns
 - Weave these solutions into the PSM model
- Model Transformations Needed
 - To go from PIM to PSM

© J.-M. Jézéquel, 2003

42

Contracts, Aspects and MDA

These ideas have been prototyped in the QCCS (Quality Controlled Component-based Software development) IST project.
cf. www.qccs.org



Conclusion

- Models can become organizations' main assets
 - Tautology if everything is a model
- Models capture aspects of reality
 - Semantics defined as several mappings to several semantic domains
 - » Not closed, because *one size fits all* not true in software
- Code generation is thus automated aspect weaving
 - But that's not the only interest of MDE
 - Connexion to semantic domains to leverage FM