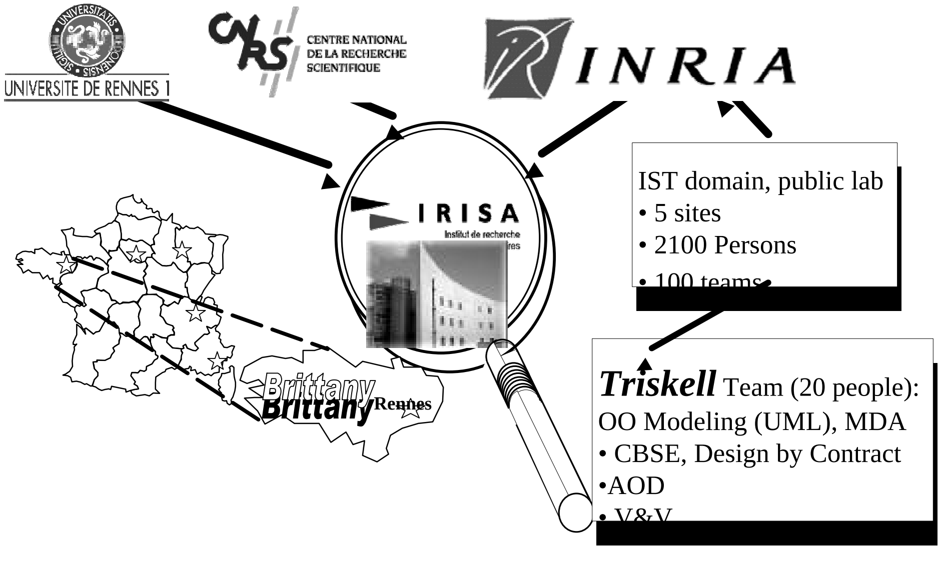
Model-Driven Engineering with Contracts, Patterns, and Aspects*

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

*This work has been partially funded by the European IST project QCCS (IST 1999 20122)

Irisa / Triskell

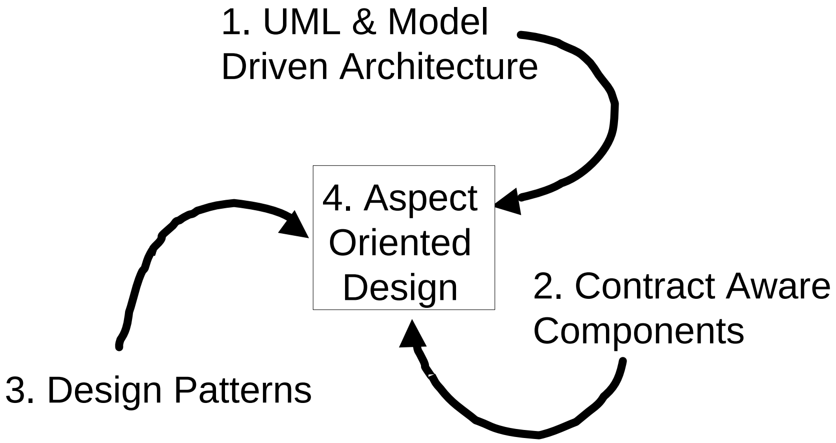
http://www.irisa.fr/triskell



IST domain, public lab
- 5 sites
- 2100 Persons
- 100 teams

Triskell Team (20 people):
OO Modeling (UML), MDA
- CBSE, Design by Contract
- AOD
- V&V

Tutorial Outline



1. UML & Model Driven Architecture

4. Aspect Oriented Design

2. Contract Aware Components

3. Design Patterns

IRISA

I. Model Driven Engineering

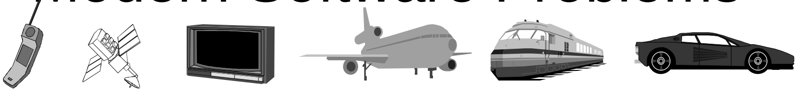
–Context: modeling component-based systems

–UML through one example

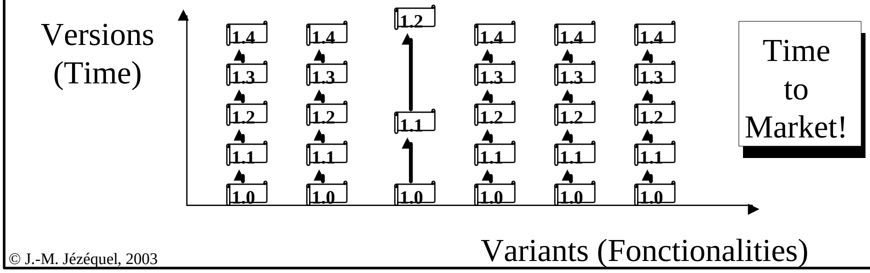
–Modeling is Aspect-Oriented (by definition)

–OMG's Model Driven Architecture as a limited version of AOM

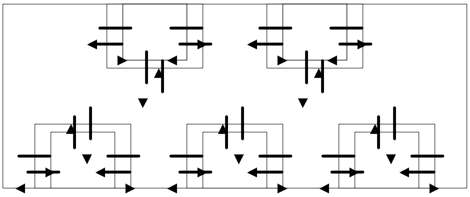
Modern Software Problems



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



OO approach:
Models and Components



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

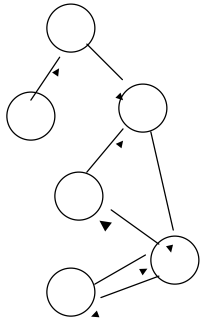
From the Object-Oriented Unification…

- From the object as the *only* one concept
  - As e.g. in Smalltalk
- To a multitude of concepts
  - Collaborations
  - Design patterns
  - Components
  - Middleware
  - Aspects

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

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)

COM+
DCOM

HTTP
HTML

Sun's
Java &
EJB

Microsoft
C# & .Net

CORBA
IIOP

XML
SOAP

Proprietary
Middleware
(eg. automotive)

- 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

+ until the next ultimate
middleware platform (~2005)

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 ?

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.

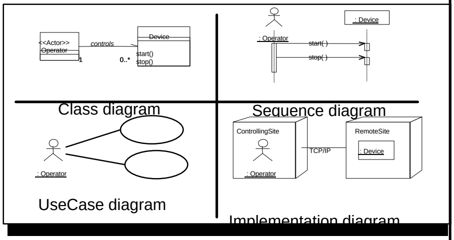
- A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality.

Jeff Rothenberg.

The World and the Model

A Model is a *simplified* representation of a subset of the World

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



M_1
(modeling space)

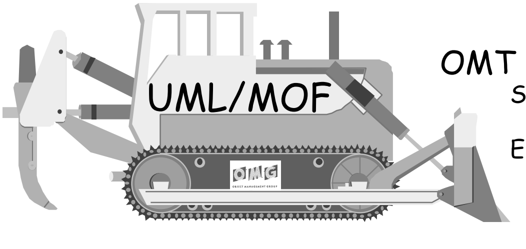
Is represented by

M_0
(the world)

UML paved the way…

From Object-Oriented Programming
to
Model-Based Software Engineering

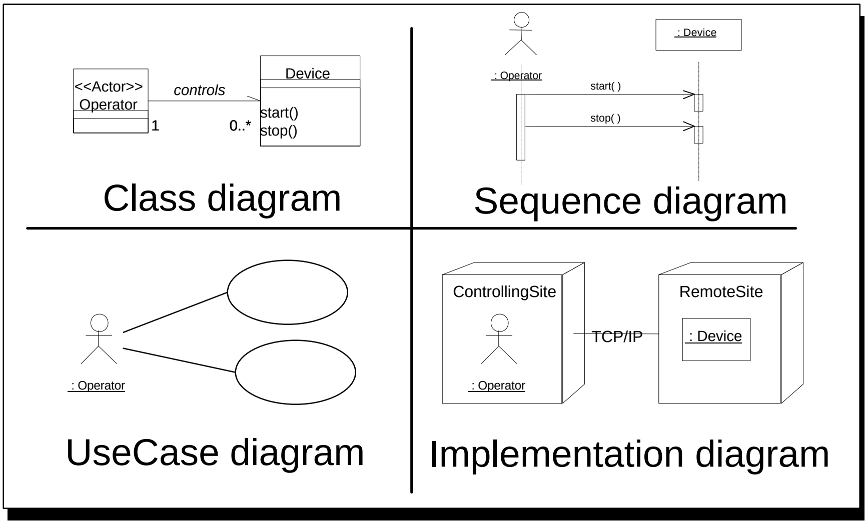
Model Driven Engineering



OMT
SA/RT
SADT
ERD
Merise
DFD
etc.
JSD

(From J. Bézivin)

UML: one model, 4 main dimensions, multiple views



Class diagram

Sequence diagram

UseCase diagram

Implementation diagram

The 9 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

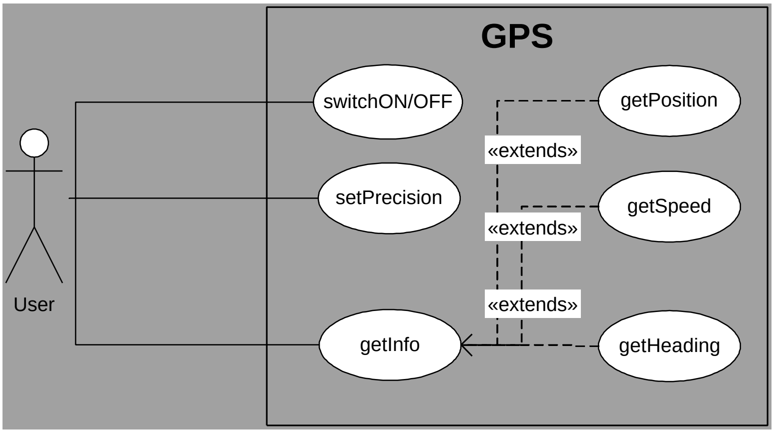
Example

- 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)

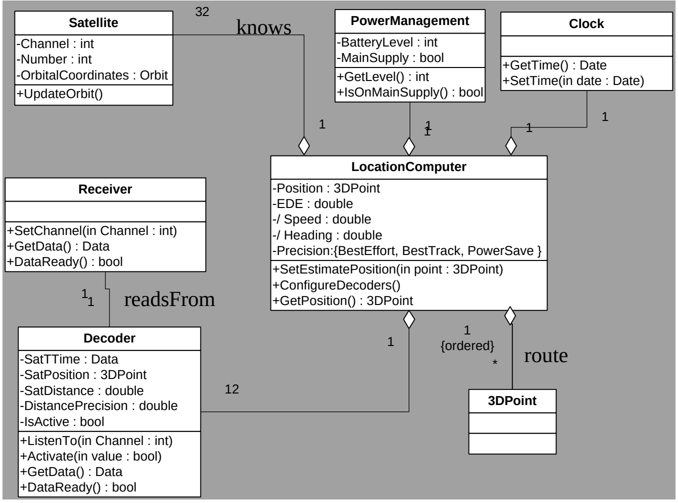
Modeling a (simplified) GPS device

- Use case diagram



Modeling a (simplified) GPS device

- Class diagram



Satellite
-Channel : int
-Number : int
-OrbitalCoordinates : Orbit
+UpdateOrbit()

PowerManagement
-BatteryLevel : int
-MainSupply : bool
+GetLevel() : int
+IsOnMainSupply() : bool

Clock
+GetTime() : Date
+SetTime(in date : Date)

LocationComputer
-Position : 3DPoint
-EDE : double
-/ Speed : double
-/ Heading : double
-Precision:{BestEffort, BestTrack, PowerSave }
+SetEstimatePosition(in point : 3DPoint)
+ConfigureDecoders()
+GetPosition() : 3DPoint

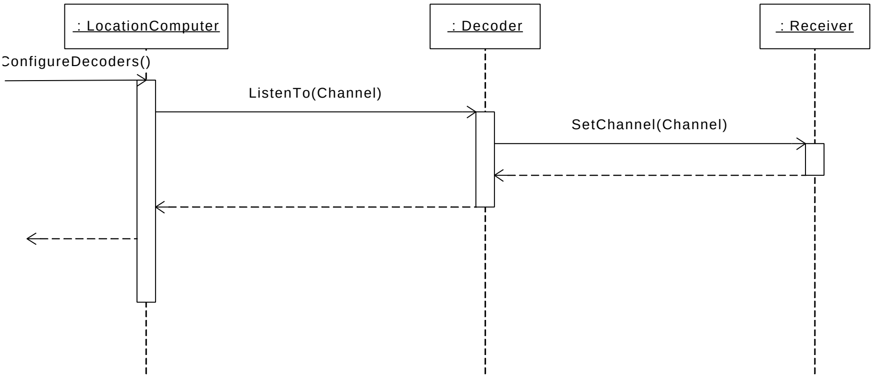
Receiver
+SetChannel(in Channel : int)
+GetData() : Data
+DataReady() : bool

Decoder
-SatTTime : Data
-SatPosition : 3DPoint
-SatDistance : double
-DistancePrecision : double
-IsActive : bool
+ListenTo(in Channel : int)
+Activate(in value : bool)
+GetData() : Data
+DataReady() : bool

3DPoint

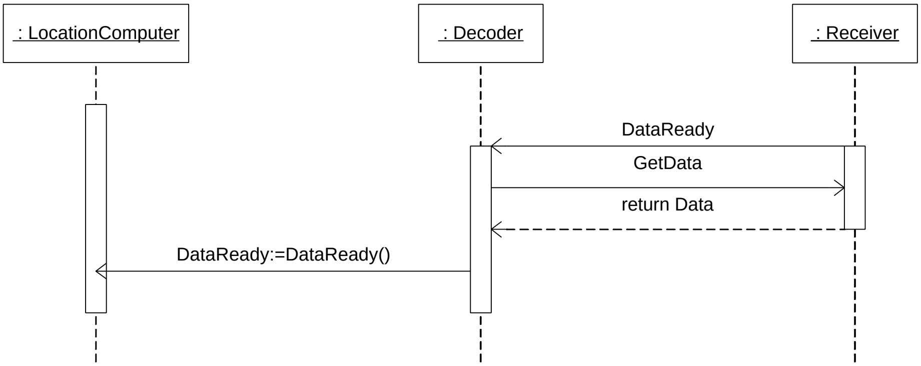
Modeling a (simplified) GPS device

- Sequence diagram: configuring decoders



Modeling a (simplified) GPS device

- Sequence diagram: interrupt driven architecture



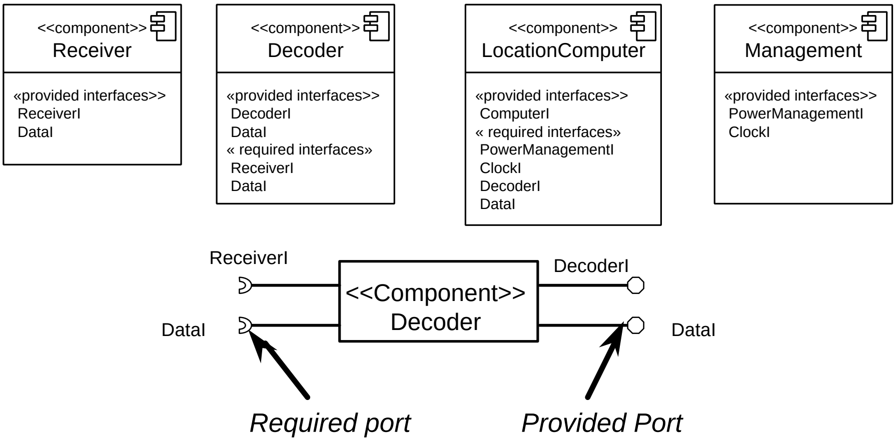
- Many more sequence diagrams needed…

Modeling a (simplified) GPS device

- Targeting multiple products with the same (business) model
  - Hand held autonomous device
  - Plug-in device for PalmTop
  - Plug-in device for laptop (PCMCIA)
  - 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

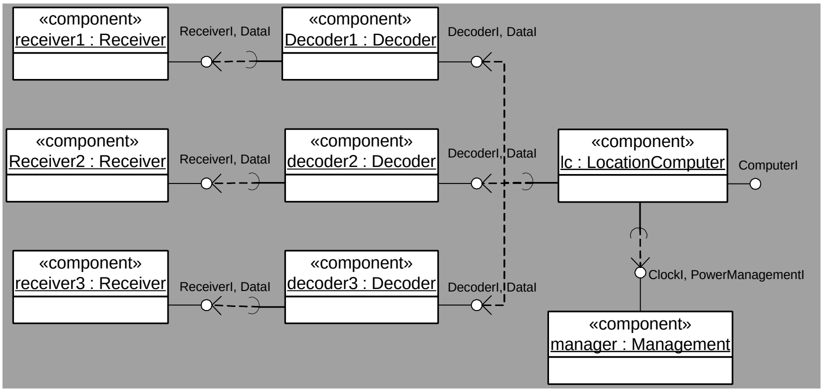


Required port

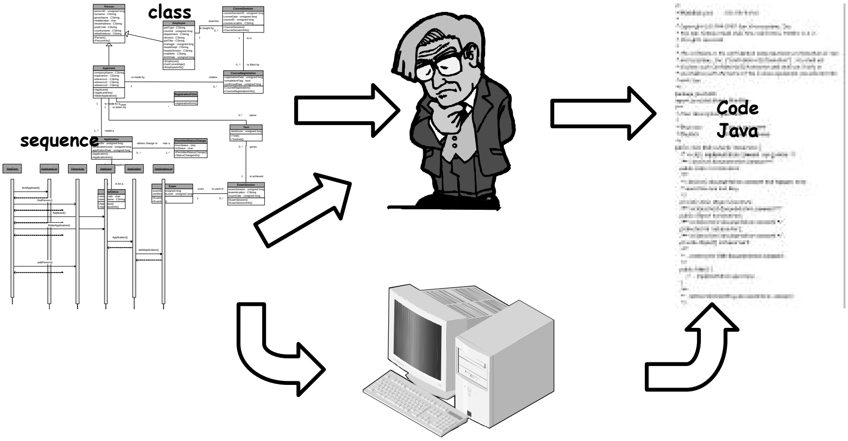
Provided Port

Modeling a (simplified) GPS device

- Deployment diagram



Models: from contemplative to productive

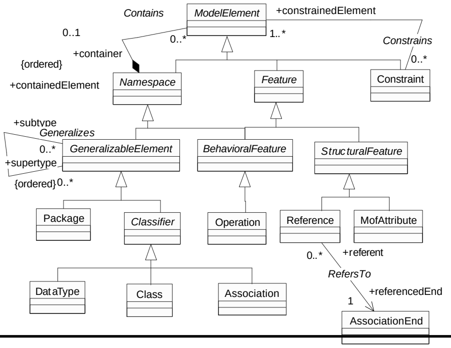


"from human-readable to computer-understandable"

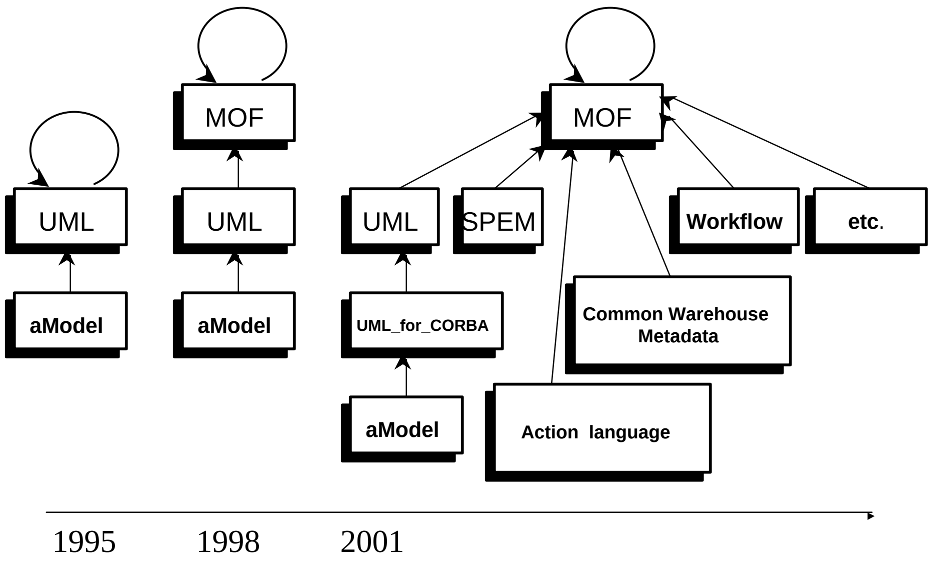
From J. Bézivin

Assigning Meaning to Models

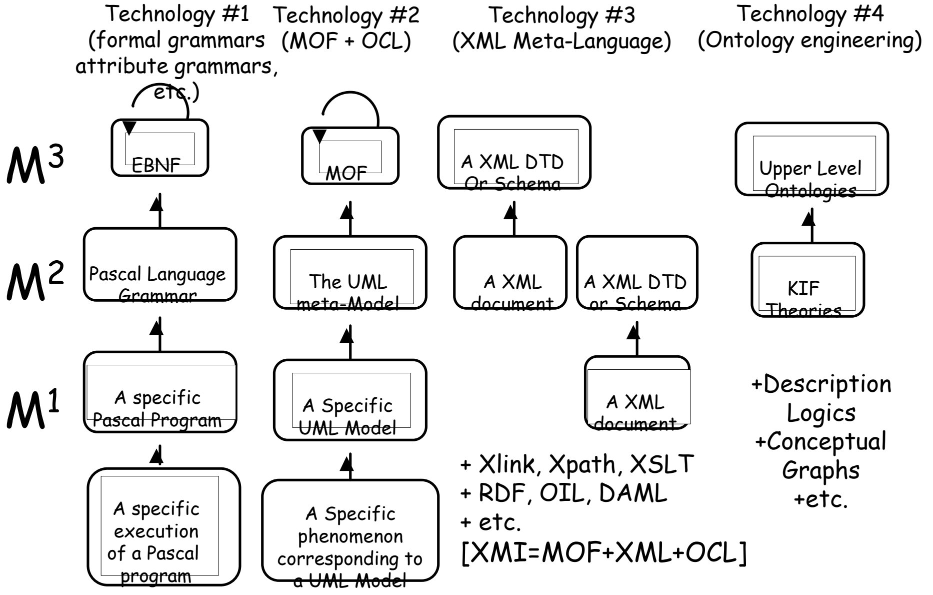
- If a UML model *is no longer* just
  - – fancy pictures to decorate your room
  - – a graphical syntax for C++/Java/C#/Eiffel...
- Then tools must be able to manipulate models
  - – Let's make a model of what a model is!
  - – => ***meta-modeling***
    - » & meta-meta-modeling..



Modeling techniques at OMG: 3 steps



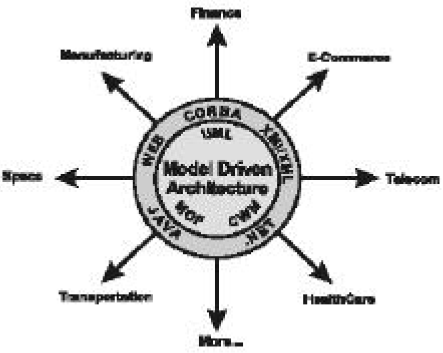
Comparing Abstract Syntax Systems



(From J. Bézivin)

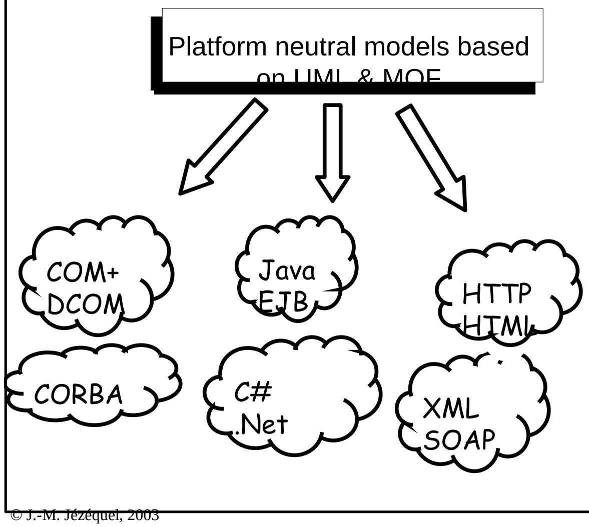
MDA: the OMG new vision

"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 platforms



Platform neutral models based on UML & MOF

COM+ DCOM

Java EJB

HTTP HTML

CORBA

C# .Net

XML SOAP

⌘ MOF & UML as the core

⌘ Organization assets expressed as models

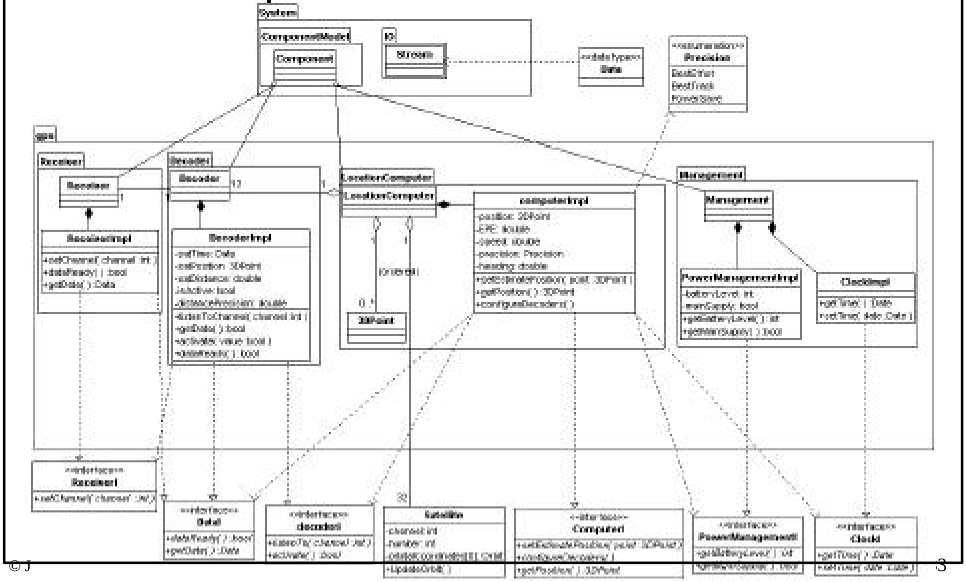
⌘ Model transformations to map to technology specific platforms

The core idea of MDA: PIMs & PSMs

- MDA models
  - **PIM**: Platform Independent Model
    - » Business Model of a system abstracting away the implementation 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

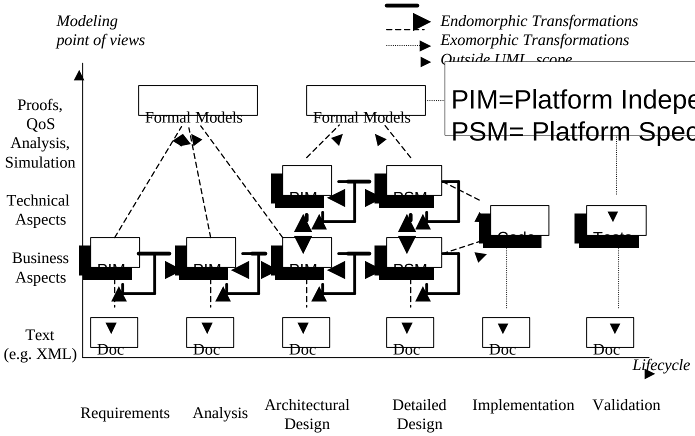


How to go From PIM to PSM?

- "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!



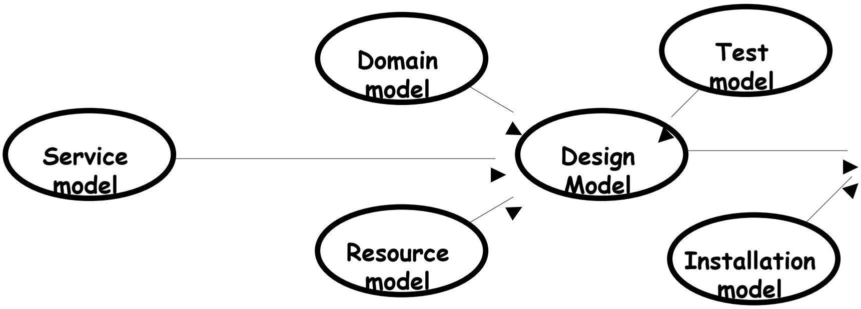
PIM=Platform Independent Model
PSM= Platform Specific Model

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.

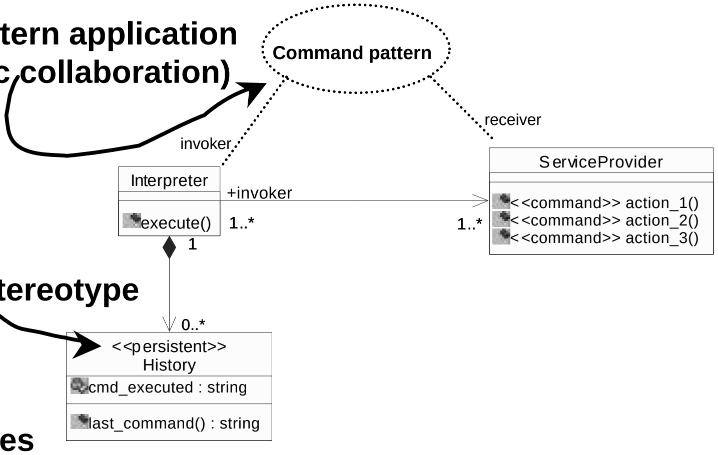
How to take these dimensions into account?

- Within UML, use built-in extension mechanisms to link with other semantic domains
- Weave all these aspects into a design model



Embedding implicit semantics into a model

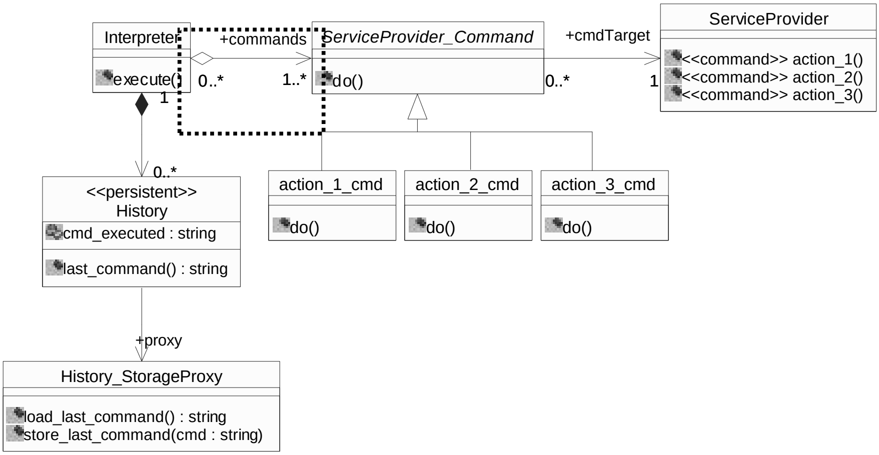
Design pattern application (parametric collaboration)



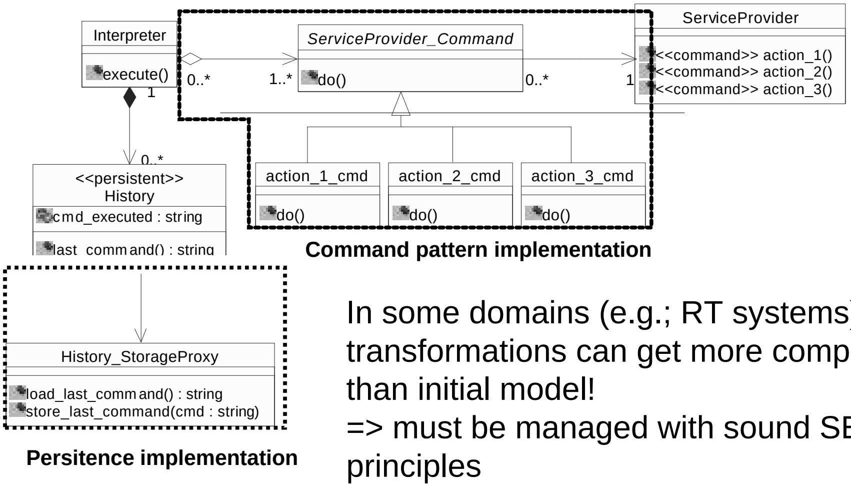
Element stereotype

…and also
Tagged values
& Contracts

…and the result we want...

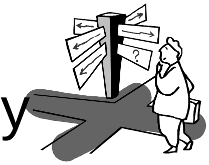


How To: Automatic Model Transformations



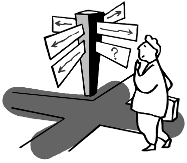
In some domains (e.g.; RT systems) transformations can get more complex than initial model!
=> must be managed with sound SE principles

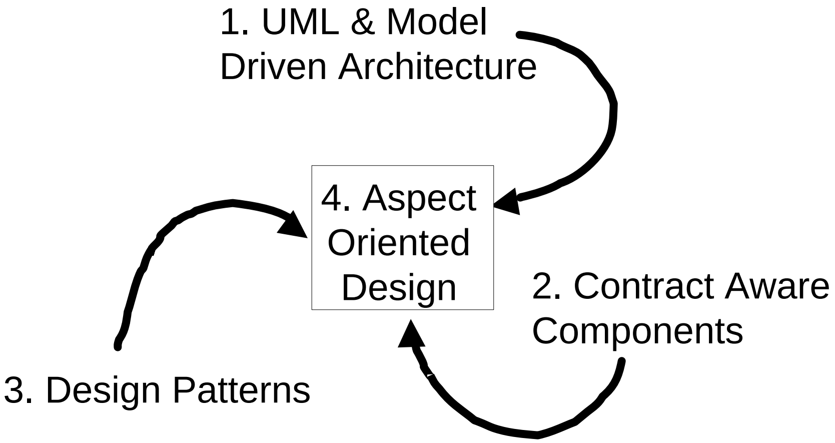
UML & Model Driven Architecture: Summary



- 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

Tutorial Outline





1. UML & Model Driven Architecture

4. Aspect Oriented Design

2. Contract Aware Components

3. Design Patterns

I R I S A

II. Contracts

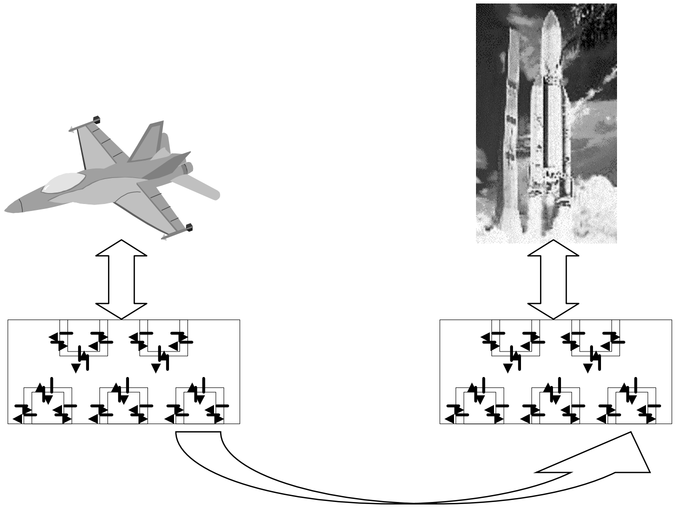
–Origin & interest, various levels of software contracts

–OCL for level 2 contracts

–QoS contracts in QCCS

Validity of component integration

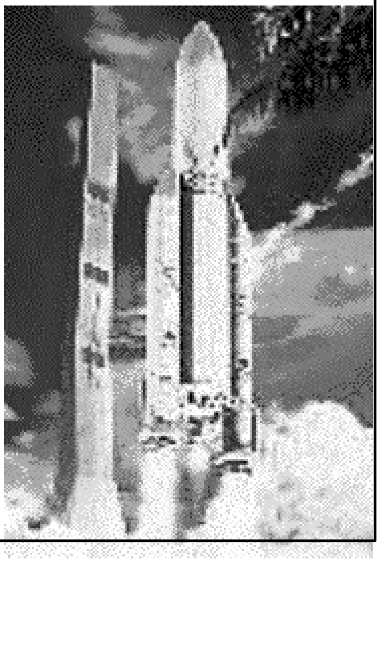
How can we (re)use a component?



Ariane 501 Maiden Launch

Kourou, ELA3 -- June, 4 1996,12:34 UT

- H0 -> H0+37s : nominal
- Within SRI 2:
  - BH (Bias Horizontal) > 2^15
  - convert_double_to_int(BH) fails!
  - exception SRI -> crash SRI2 & 1
- OBC disoriented
  - Angle > 20°, huge aerodynamics constraints
- boosters separating...



Ariane 501 Maiden Launch

Kourou, ELA3 -- June, 4 1996,12:34 UT

- H0 + 39s: Self destruction (cost: € 500M)

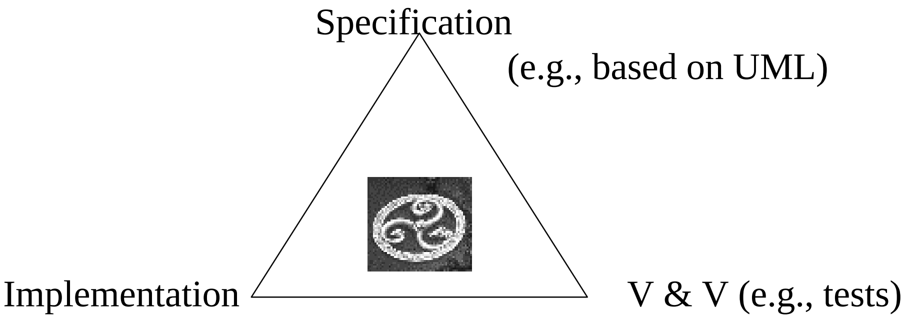
Why? (cf. IEEE Comp. 01/97)

- Not a programming error
  - unprotected conversion = design decision (~1980)
- Not a design error
  - Justified vs. Ariane 4 trajectory & RT constraints
- Problem with integration testing
  - As always, could have theoretically been caught. But huge test space vs. limited resources
  - Furthermore, SRI useless at this stage of the flight!

Why: (cf. *IEEE Comp. 01/97*)

- Reuse of a component with a hidden constraint!
  - Precondition : abs(BH) < 32768.0
  - Valid for Ariane 4, but no longer for Ariane 5
    - » *More powerful rocket*

How Can You Build Trust into a Component?



Specification

(e.g., based on UML)

Implementation

V & V (e.g., tests)

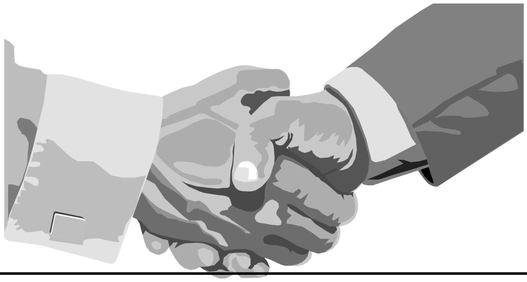
Check Consistency between these 3 Aspects

- Inherent (test & resistance to mutations)

- Composability inter-components

Specification = Contract between the client and the component

- In real life, many kinds of contracts
  - *From Jean-Jacques Rousseau’s “Social Contract” to “cash & carry”*
- Likewise, many issues for software contracts in a distributed setting



Four levels of Software Contracting

- Basic (syntactic)
  - the program compiles…
- Behavioral
  - Eiffel like pre/post conditions
- Synchronization
  - e.g. path expressions, etc. [McHale]
- Quality of service (quantitative)
  - Possible dynamic negotiation

Cf. IEEE Computer July 1999

Level 2 Contracts in UML: OCL (Object Constraint Language)

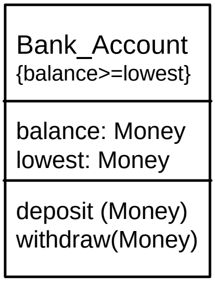
- Constraint = Boolean expression (no side effect) on
  - Usual operations on basic types (Boolean, Integer...)
  - attributes of class instances
  - « query » operation (functions side-effect free)
  - associations from the UML class diagram
  - States from StateCharts

Behavioral Contracts

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

Class invariants in UML

- Contraints can be added to UML model
  - notation: between { }
- Invariant = Boolean expression
  - True for all instances of a class in stable states...
  - Expressed with the OCL (Object Constraint Language)
    - e.g. {balance >= lowest}
    - Can also navigate the associations



Bank_Account
{balance>=lowest}

balance: Money
lowest: Money

deposit (Money)
withdraw(Money)

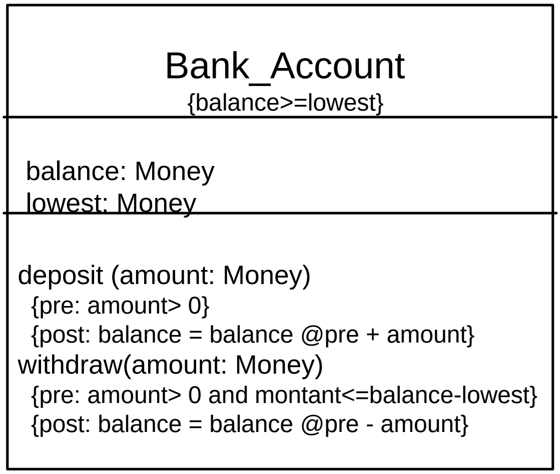
Precondition:
Burden on the client

- Specification on what must be true for a client to be allowed to call a method
  - example: amount > 0
- Notation in UML
  - {«precondition» OCL boolean expression}
  - Abbreviation: {pre: OCL boolean expression}

Postcondition: *Burden on the implementor*

- Specification on what must be true at completion of any successful call to a method
  - example: balance = balance @pre + amount
- Notation in UML
  - {«postcondition» *OCL boolean expression*}
  - Abbreviation: {post: *OCL boolean expression*}
  - Operator for previous value (idem old Eiffel):
    - » *OCL expression* @pre

To be Abstract <u>and</u> Precise with the UML



Bank_Account
{balance>=lowest}

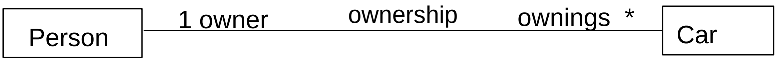
balance: Money
lowest: Money

deposit (amount: Money)
  {pre: amount> 0}
  {post: balance = balance @pre + amount}
withdraw(amount: Money)
  {pre: amount> 0 and montant<=balance-lowest}
  {post: balance = balance @pre - amount}

- •In memory implementation
  - •straightforward
  - •list of transactions
- •Data base implementation
- •etc.

Non-local contracts: navigating associations

- Each association is a navigation path
  - The context of an OCL expression is the starting point
  - Rolenames are used to select which association is to be traversed (or target classname 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

Basic operations on collections

- *isEmpty*
  - *true* if collection has no element

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

- *notEmpty*
  - *true* if collection has at least one element
- *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)
```

Other OCL Operations

- exists (expr)
  - true if expr holds for at least one element of the collection
- includes(elem), excludes(elem)
  - True if elem belongs (resp. does not belong) to the collection
- includesAll(coll)
  - True if all elements from coll are also here
- union (coll), intersection (coll)
  - Classical set operation
- asSet, asBag, asSequence
  - Type conversion

Interest of Behavioral Contracts

- Specification, documentation
  - Not a software fault tolerance gadget
  - Might help system fault tolerance...
- Help V&V
  - When assertions are monitored
    - » Must go from model to instrumented code: *Transformations*
  - Never doing debugging again
- Help allocate responsibilities during integration
  - No longer have to find a scapegoat ;-)

Contract Violations: *Preconditions*

- The client broke the contract.
  - The provider does not have to fulfill its part of the contract.
  - If contracts are monitored, an exception should be raised
    - » making it easy to identify the exact origin of the fault.

```
Unhandled exception: Routine failure. Exiting program.
Exception history:
===============================================================================
Object Routine
Type of exception          Description               Line
===============================================================================
#<BANK_ACCOUNT5f0c0>                                 BANK_ACCOUNT:deposit
precondition violated      positive_amount           63
-------------------------------------------------------------------------------
#<USER 5f000>                                        USER:test
Routine failure                                      90
-------------------------------------------------------------------------------
#<DRIVER 5f010>                                      DRIVER:make
Routine failure                                      18
-------------------------------------------------------------------------------
```

Contract violations: *Postconditions*

- The implementation of a method did not comply with its promise: This is a bug

```
Unhandled exception: Routine failure. Exiting program.
Exception history:
==============================================================================
Object Routine
Type of exception              Description                    Line
==============================================================================
#<BANK_ACCOUNT5f0c0>                                           BANK_ACCOUNT:deposit
postcondition violated         deposited                        70
--------------------------------------------------------------------------------
#<USER 5f000>                                                  USER:test
Routine failure                                                90
--------------------------------------------------------------------------------
#<DRIVER 5f010>                                                DRIVER:make
Routine failure                                                18
--------------------------------------------------------------------------------
```

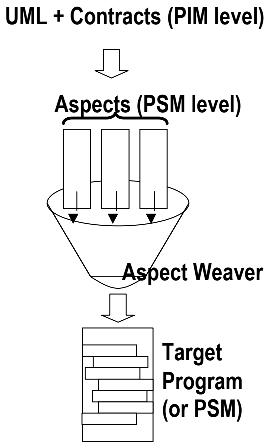
- Again, easy to trace...(between lines 63-70)

Application to Component Testing *(Self-Testable Components)*

- Embed the test suite inside the component
  - Implements a SELF_TESTABLE interface
  - Component Unit Test suite =
    - » Test data + activator
    - » Oracle (mostly executable assertions from the component specification)
- Useful in conjunction with
  - Estimating the Quality of the Component
  - Integration Testing

QCCS: an IST Project

- QCCS = Quality Controlled Component-based Software development
- Contract Aware Components
  - Including QoS
- Aspect Weaver for Implementing Contracts
- Apply methodology and tools to 3 case studies
- Partners:
  - INRIA, TU Berlin, Univ. Cyprus
  - SchlumbergerSema, KD Soft



UML + Contracts (PIM level)

Aspects (PSM level)

Aspect Weaver

Target Program (or PSM)

QCCS Challenges

- Model for QC Component Specification
  - accounting for the various levels of contracts in UML
- Infrastructure for QC Components
  - runtime contract management
  - integration into standard component technology: CORBA/EJB and .NET
- QCCS-specific development process
  - methodology and tool support based on AOSD

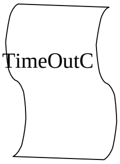
QoS Contracts in QCCS (Level 4)

- QoS Dimension (from QML)
  - Name (responseTime, throughput…)
  - Type (float, int, bool, enum…)
  - Direction (up, down)
  - Unit (seconds, bytes, none …)
- QoS Categories
  - To group a set of QoS Dimensions
- Contracts
  - Inherit one or more QoS categories
  - Bound to ports

QML: A Language for Quality of Service Specification
HP Labs Technical Reports
http://www.hpl.hp.com/techreports/98/HPL-98-10.html

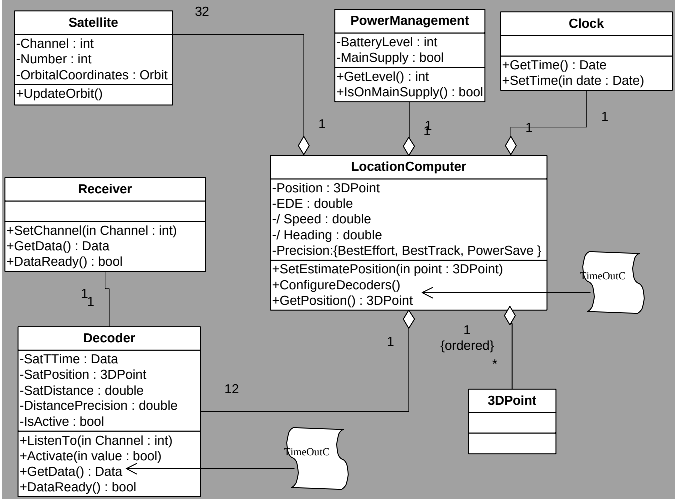
Example for the GPS

- 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



Example

- Adding QoS contracts to our GPS device



Motivations to go beyond atomic contracts => λ-Contracts

- Contracts can provide *trust*
  - *But you cannot (completely) hide the platform*
  - abstraction ? hiding
- Components have offered *and* required interfaces
  - need to express dependencies between interfaces
- Component contracted interfaces:
  - Implies dependencies between offered and required contracts

Component contracts

- Component Based Systems are not layers of functionalities
  - networks of interdependent pieces
- *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

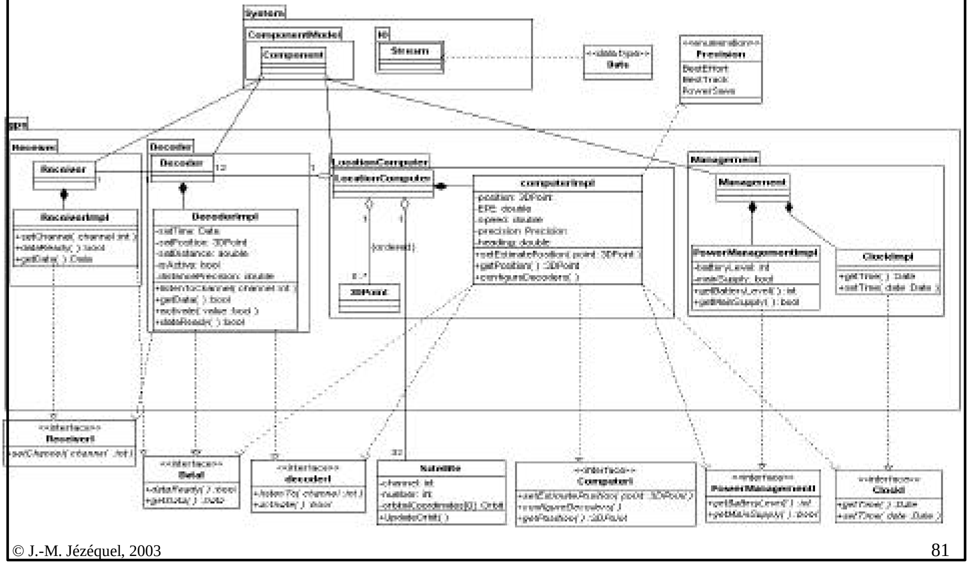
Contract space

- 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 compute:
  - Logical deduction
  - Functionally dependent parameters
  - ...

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 ...
- Model Transformations Needed
  - To go from PIM to PSM

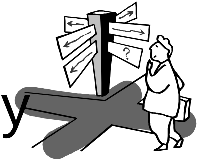
How to implement these contracts for this .NET PSM?



Weave contract management

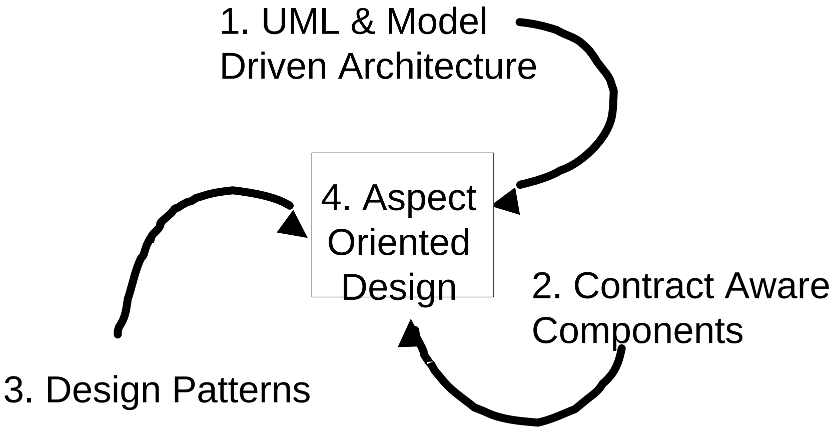
- Problem: it depends on the semantics of each contract type
  - – QML does not capture the semantics
  - – Sometimes quite complicated
    - » E.g. bounded throughput variation implies non-instantaneous monitoring and the collecting of statistics
    - » May heavily depends on the platform!
- There exist known solutions to these problems
  - – Apply design patterns?
  - – Weave design pattern applications into the PSM model

Contract Aware Components: Summary



- Components must have explicitly defined contracts
  - Four levels of contracts
  - Modeling in UML based on e.g. QML
  - Reasoning on models with components & contracts
    - Bottom-up or top-down
- Contracts to declaratively express non-functional aspects
  - Dependencies between contracts
- Monitoring of contracts is
  - Complex, Cross-cutting, Platform dependant

Tutorial Outline



1. UML & Model Driven Architecture

2. Contract Aware Components

3. Design Patterns

4. Aspect Oriented Design

III. Design Patterns

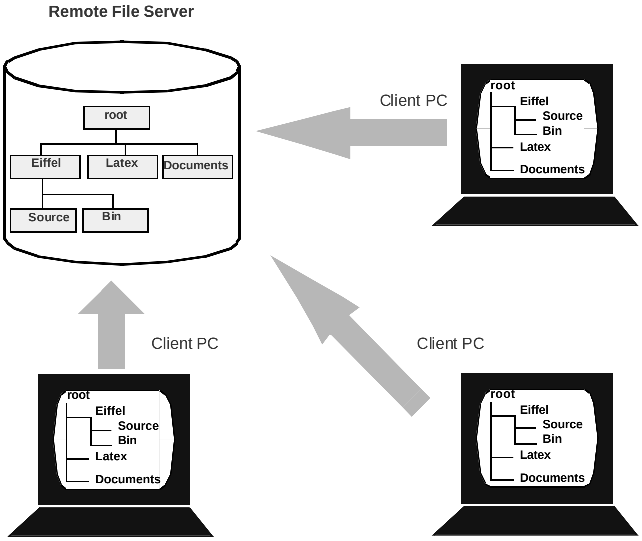
– Origin & interest

– Precise modeling with UML and Meta-level OCL

Origin of Design Patterns

- GoF's Book: A catalog
  - *Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma, Helm, Johnson, Vlissides). Addison Wesley, 1995
- Earlier works by Beck, Coplien and others...
- Origin of Patterns in Architecture (C. Alexander)
  - *Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to this problem in such a way that you can use this solution a million times over, without ever doing it the same way twice.*

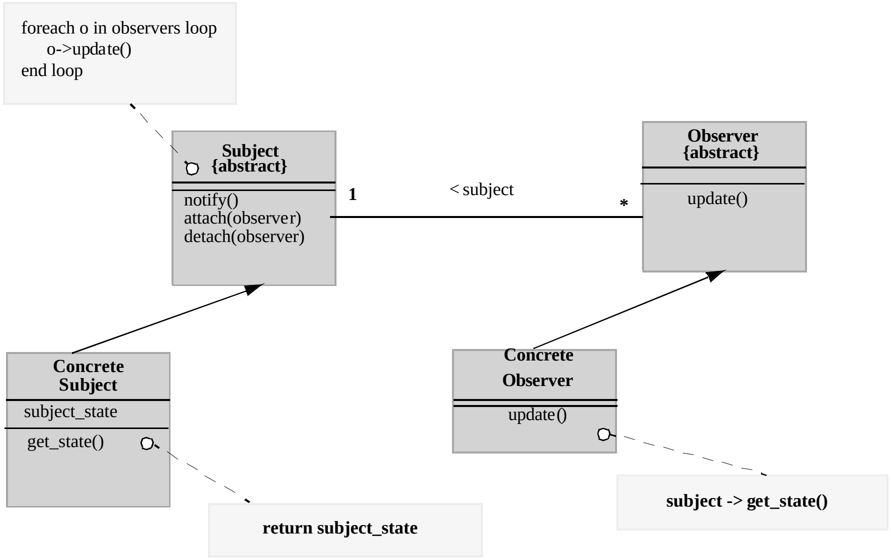
Example: A Distributed File System



The Observer Pattern

- Intent
  - *Dependency from a subject to observers so that when the subject changes state, observers are notified*
- Key constraints
  - *Any number of observers*
  - *Each observer can react specifically to the notification of change*
  - *The subject should be decoupled from the observers (dynamic add/remove of observers)*

Structure of the Observer Pattern



foreach o in observers loop
    o->update()
end loop

Subject
{abstract}

notify()
attach(observer)
detach(observer)

1

< subject

*****

Observer
{abstract}

update()

Concrete Subject

subject_state

get_state()

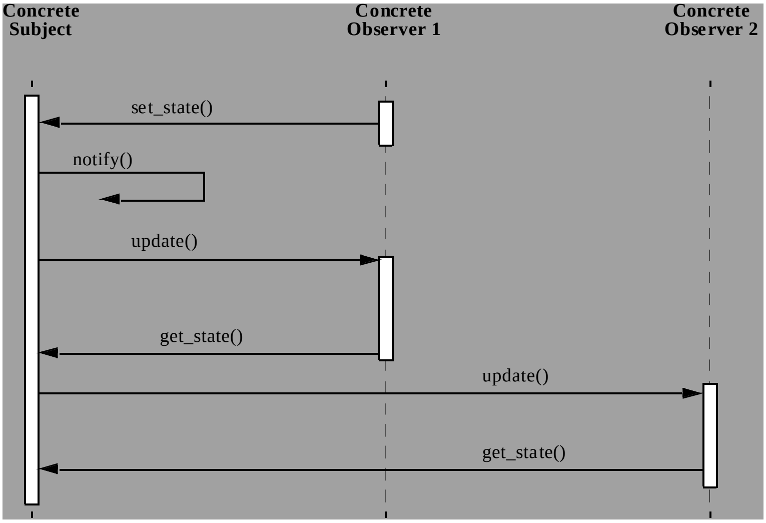
Concrete Observer

update()

return subject_state

subject -> get_state()

Collaborations in the Observer Pattern



Concrete Subject

Concrete Observer 1

Concrete Observer 2

set_state()

notify()

update()

get_state()

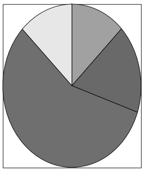
update()

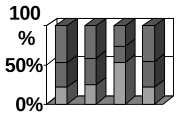
get_state()

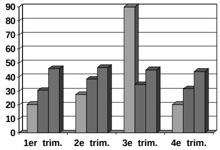
Another Problem...

- Any number of views on a Data Table in a windowing system…
  - close, open views at will…
  - change the data from any view
    - » … and the other are updated

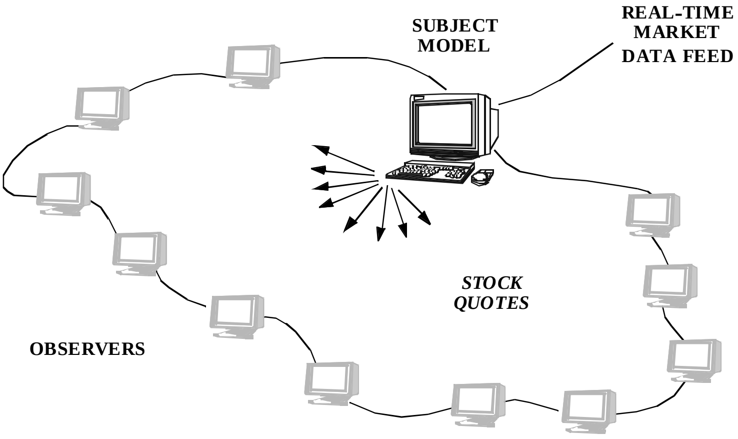
	1er trim.	2e trim.	3e trim.	4e trim.
Est	20,4	27,4	90	20,4
Ouest	30,6	38,6	34,6	31,6
Nord	45,9	46,9	45	43,9







Yet Another Problem...



SUBJECT MODEL

REAL-TIME MARKET DATA FEED

STOCK QUOTES

OBSERVERS

What Design Patterns are all about

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

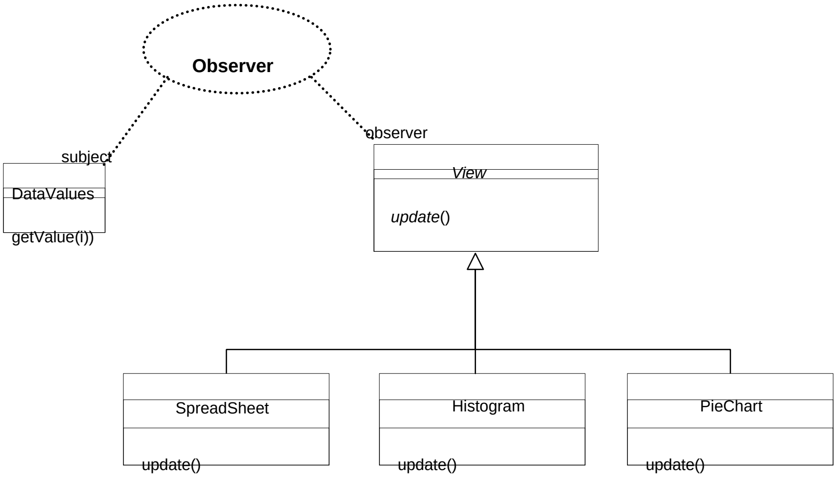
Interest of Documenting Design Patterns

- Communication of architectural knowledge among developers
- Provide a common vocabulary for common design structures
  - Reduce complexity
  - Enhance expressiveness, abstractness
- Distill and disseminate experience
  - Avoid development traps and pitfalls that are usually learned only by experience

Precise Modeling of Design Pattern Applications

- Go beyond mere documentation
- Specifying reusable applications of design patterns
  - Structural properties
  - Behavioral properties
- Using design patterns in a model
  - Pointing out or detecting pattern occurrences
  - Checking for missing structural properties

Example in UML



UML Current Solution (UML 1.4)

- Patterns in UML rely on collaborations
  - That is, sets of collaborating roles
  - A role in a collaboration is a placeholder for objects conforming to the role’s base classifier
  - Additional constraining elements can be used
- Collaboration diagrams or sequence diagrams are used to represent expected interactions among participant objects

UML Current Solution (continued)

- Reusability of the pattern is obtained by turning the bases of roles into formal template parameters
- A pattern occurrence is then a template instantiation (a.k.a. binding) providing the actual participants for each template base
- The binding is represented using the intuitive ellipse notation

Using Templates to Express Structural Constraints?

- Using templates as “prototypical” structural constraints was a good idea:
  - Placeholders share the same notation as the “real” modeling elements
  - No need to introduce M2 (Meta-Model) level entities
- But...

Limitations of the approach (1/2)

- Template parameters provide only a fixed number of placeholders for modeling elements
  - Problem in Composite, Visitor, etc.
- Almost everything must be parameterized (including so-called constraining elements)...
- … which leads to numerous parameters.
- Some parameters are “compound”.

Limitations of the approach (2/2)

- Template expansion is only used to link and check the conformance of the actual modeling elements to this set of "prototypical" structural constraints.
- Moreover, no conformance rules are specified in the UML documentation.

Patterns as Meta-Level Constraints

- Use explicit constraints at the M2 level
  - instead of implicit template constraints
- A pattern is modeled a set of constraints
  - similar to UML Well-Formedness Rules, but with
    - » Pre-conditions stating the initial situation
    - » Post-conditions to describe the result of the pattern application
  - These supplementary constraints apply only to the participants in the pattern occurrences
- The profile mechanism could be used as a way to build a repository for pattern definitions

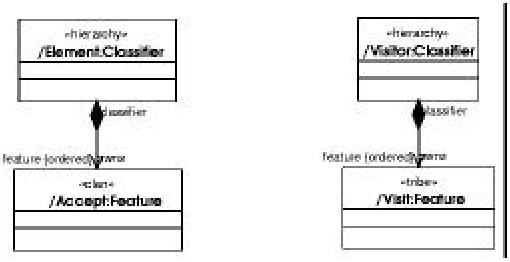
About Collaborations and Constraints

- Collaborations as contexts for OCL expressions
  - Some constraints involve several elements
  - The context of an OCL expression
    is normally made of a single element - “self”
  - Collaborations and their roles help describe complex contexts

Collaborations of modeling elements

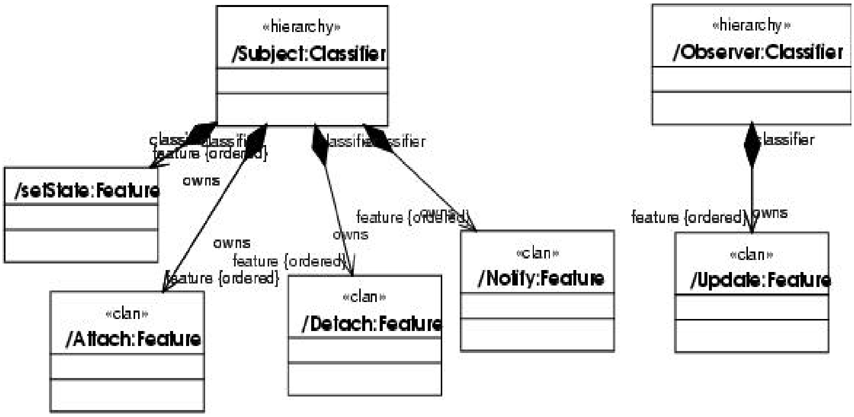
- A pattern can be thought of as a constrained collaboration of UML modeling elements.
- Refinements can be specified by specializing the collaboration and adding new constraints
- Each occurrence of the pattern in the model corresponds to a M2 collaboration occurrence

Model of the Visitor Pattern



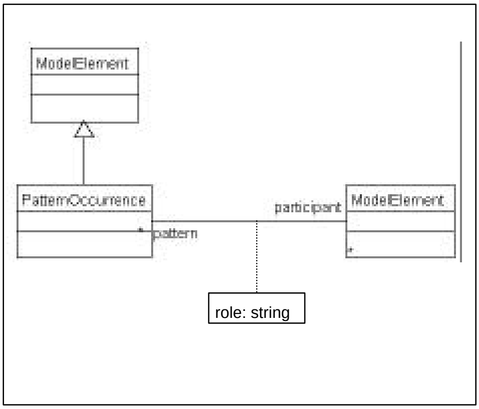
-
  ```
  visit->size() = element->size()
  and visit->forall(v |        visitor->feature->includes(v)
                           and v.parameter->size() = 1
                           and element->exists(e | v.parameter.type = e)
                           and …
                       )
  ```

Model of the Observer Pattern

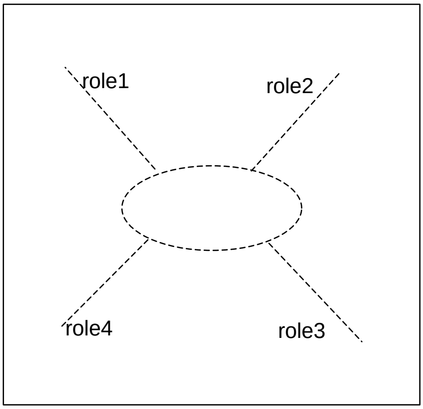


Meta-Model for Pattern Occurrences

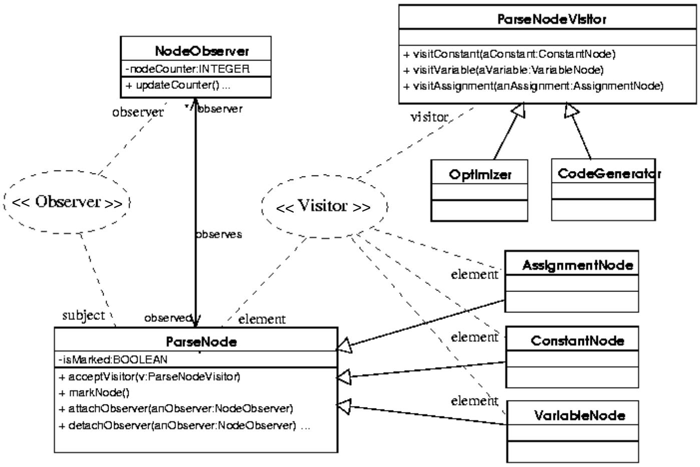
Meta-Model



Notation



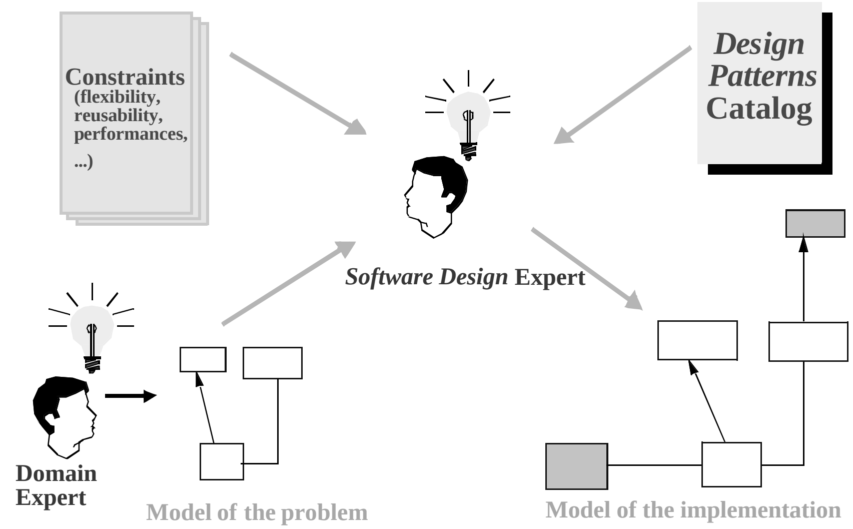
Pattern Occurrences



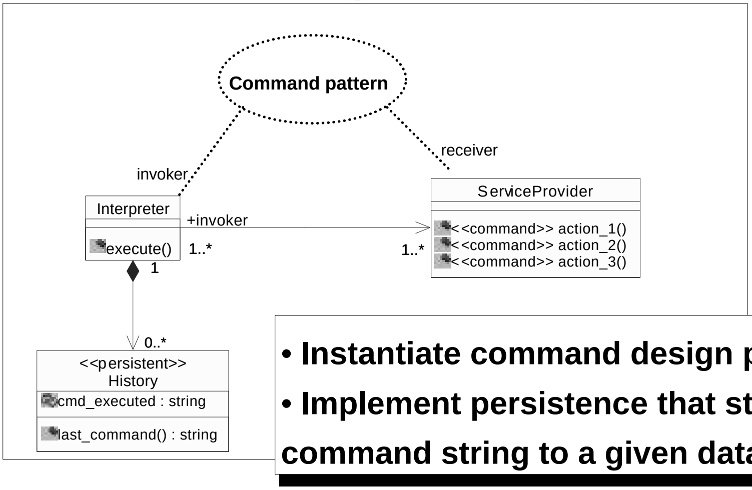
About Behavioral Properties

- Interactions (sequence diagrams) are representations of expected behavior
  - They are to be interpreted as properties
- Precise specification requires a model of the execution semantics
  - HMSCs, Action Semantics
- Behavioral properties should be constraints restraining the set of possible executions

Design Patterns in the Model Driven Architecture

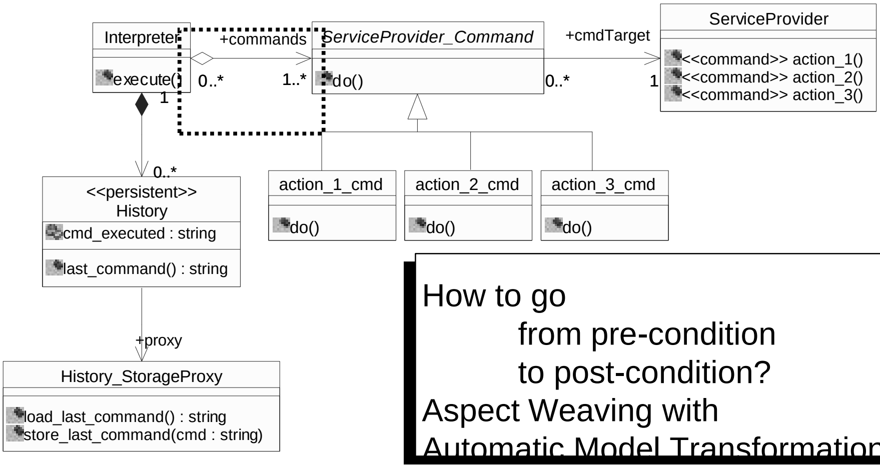


Let’s look at a business model... (pre-state)



- **Instantiate command design pattern.**
- **Implement persistence that stores the command string to a given database.**

…and the wanted design model... (post-state)

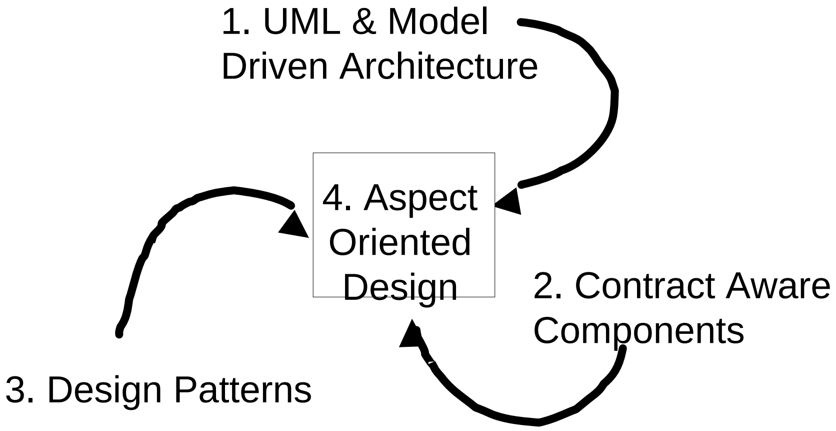


How to go
from pre-condition
to post-condition?
Aspect Weaving with
Automatic Model Transformations!

Design Patterns: Summary

- Design Pattern applications as constrained collaborations
- Many different variants of applications
  - E.g. observer push or pull
- Identification of occurrence
  - UML ellipse notation
- Weave the pattern application through model transformations
  - Correspondance with PIM vs PSM

Tutorial Outline



1. UML & Model Driven Architecture

2. Contract Aware Components

3. Design Patterns

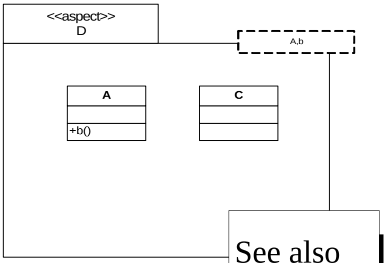
4. Aspect Oriented Design

IV. Aspect-Oriented Design & Model Transformations

– Modeling Aspects in UML
– Weaving Static Aspects
– Weaving Dynamic Aspects
–The Grand Unification of Contracts/Patterns/Aspects
– Through an OCL2 Meta-level interpreter

Modeling Aspects

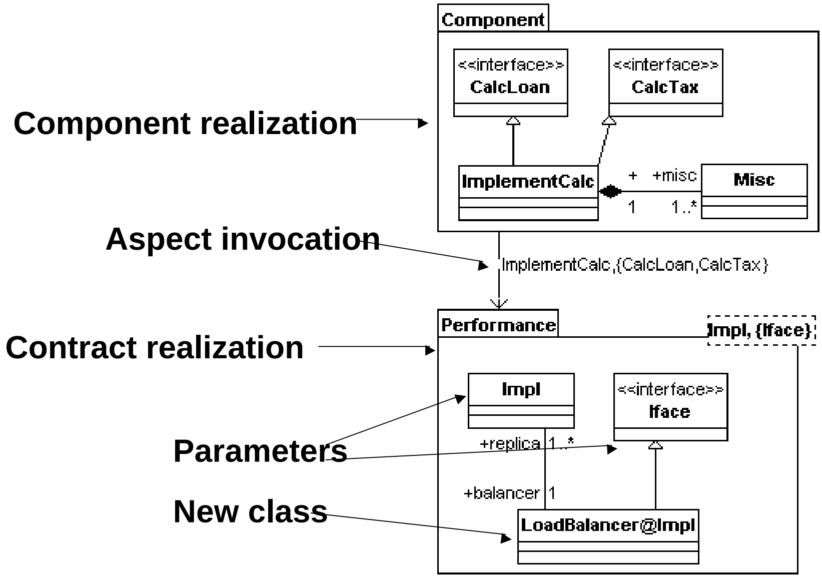
- Aspects are defined separately from any given model (=> reuse across models)
  - Stereotyped packages
  - Well defined interfaces
  - Have (M2 level) formal parameters
    - » parameters may be typed or constrained (with OCL)
    - » cardinality on a per parameter basis



See also S. Clark's works

- Join-points can be bound (with standard extension mechanisms) to *any* element in a UML model
  - class, method, state,…
  - event occurrence, method call,...

Weaving Aspects in UML



Component realization

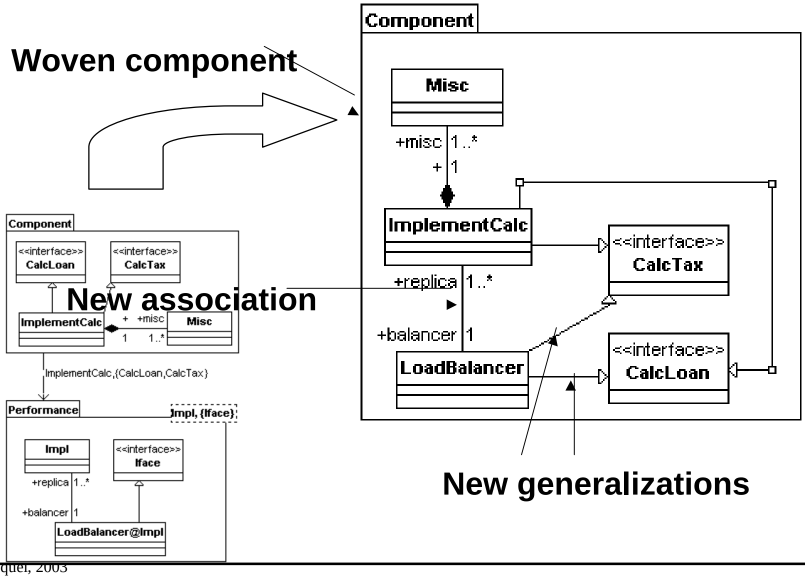
Aspect invocation

Contract realization

Parameters

New class

Woven Aspects in UML



Woven component

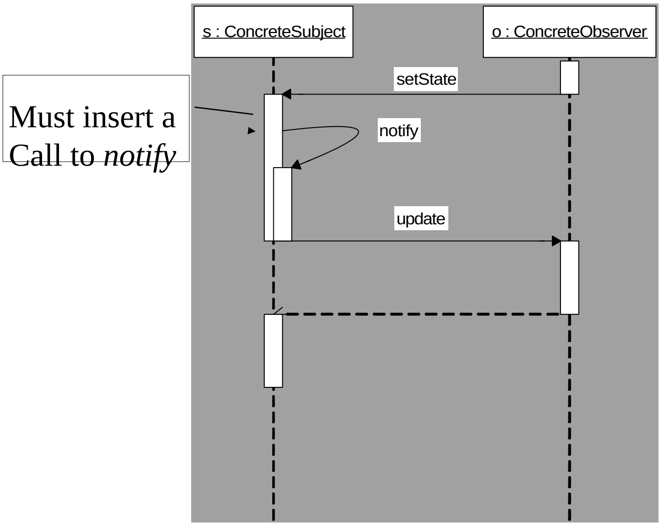
New association

New generalizations

Dynamic Aspects

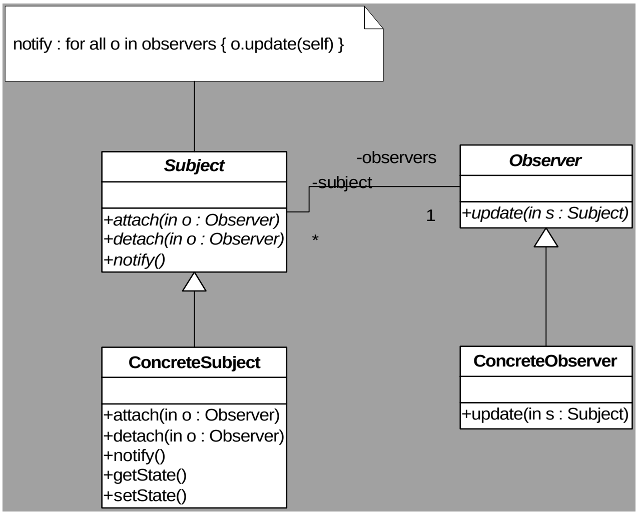
- Templates not enough for dealing with dynamic aspects
  - Those represented by (H)MSC, etc.
  - Useful for e.g. behavioral patterns
- Must deal with interaction protocols among objects
  - If new operations added on target object, must add calls to these operations at relevant places in the client objects.

Observer Revisited: The Observer Protocol



Must insert a Call to *notify*

Observer again



notify : for all o in observers { o.update(self) }

Subject

+attach(in o : Observer)
+detach(in o : Observer)
+notify()

-observers

-subject

1

*

Observer

+update(in s : Subject)

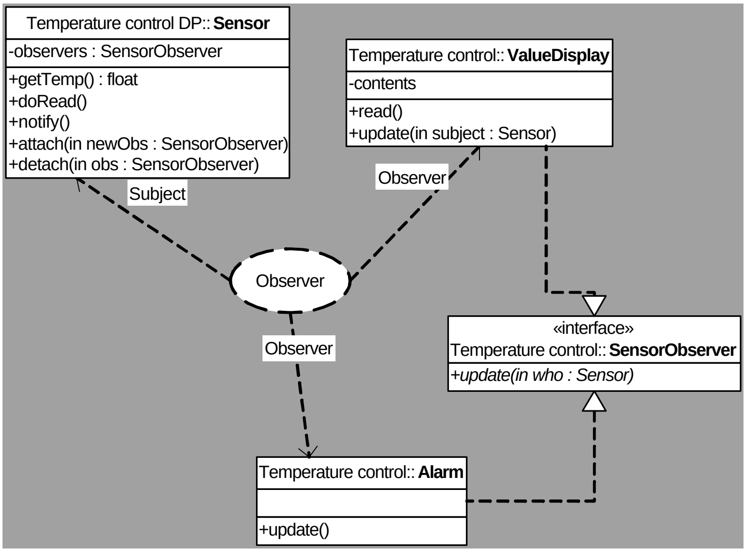
ConcreteSubject

+attach(in o : Observer)
+detach(in o : Observer)
+notify()
+getState()
+setState()

ConcreteObserver

+update(in s : Subject)

Observer pattern reference



Temperature control DP::**Sensor**

-observers : SensorObserver

+getTemp() : float
+doRead()
+notify()
+attach(in newObs : SensorObserver)
+detach(in obs : SensorObserver)

Temperature control::**ValueDisplay**

-contents

+read()
+update(in subject : Sensor)

Subject

Observer

Observer

Observer

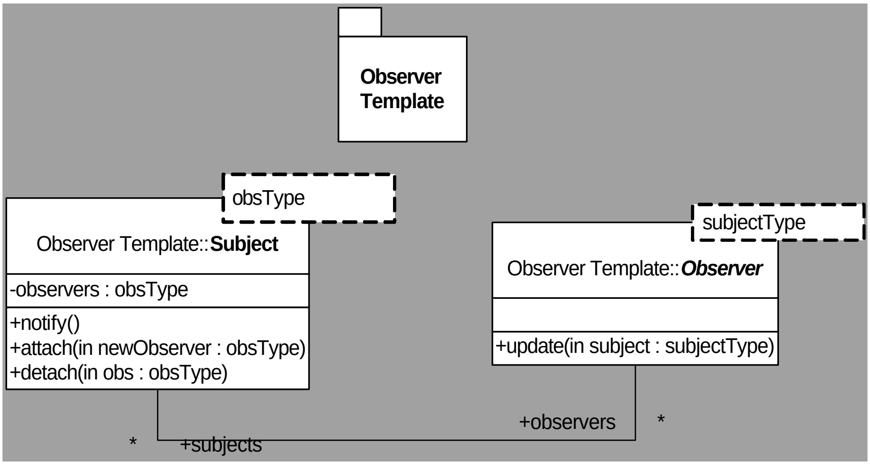
«interface»
Temperature control::**SensorObserver**

+update(in who : Sensor)

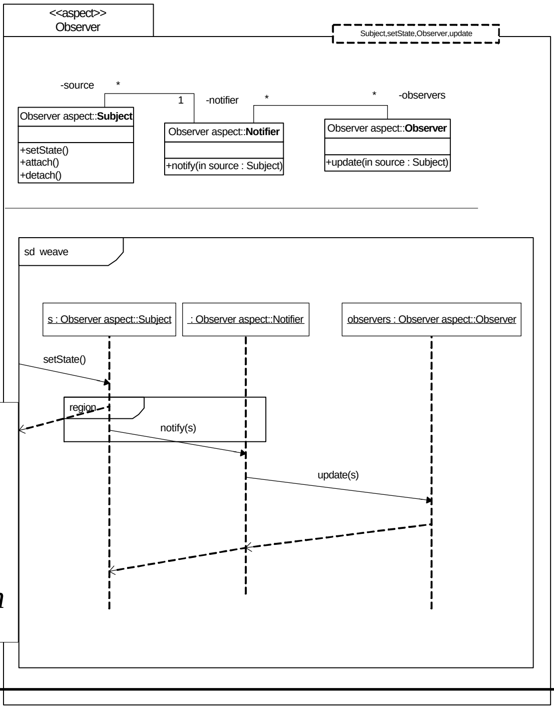
Temperature control::**Alarm**

+update()

Template for Observer

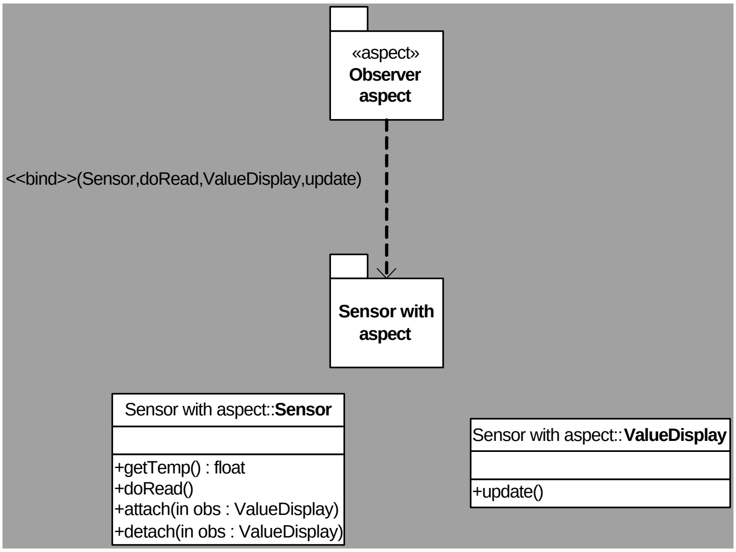

Observer Template
obsType
Observer Template::Subject
-observers : obsType
+notify()
+attach(in newObserver : obsType)
+detach(in obs : obsType)
subjectType
Observer Template::Observer
+update(in subject : subjectType)
+observers
*
*
+subjects

Observer Definition as an Aspect

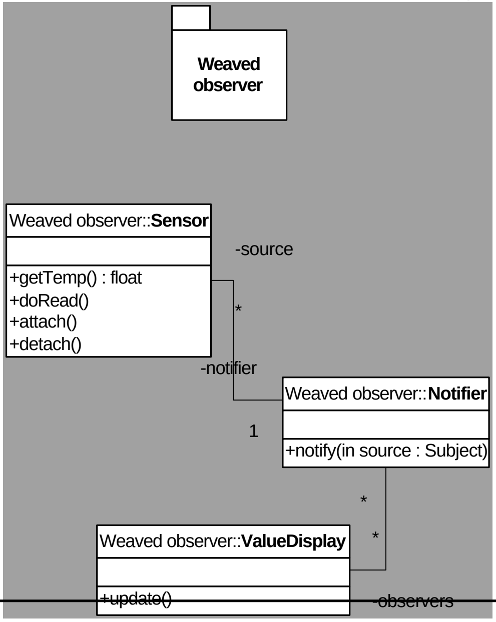

<<aspect>>
Observer
Subject,setState,Observer,update
-source
*
1
-notifier
*
*
-observers
Observer aspect::Subject
+setState()
+attach()
+detach()
Observer aspect::Notifier
+notify(in source : Subject)
Observer aspect::Observer
+update(in source : Subject)
sd weave
s : Observer aspect::Subject
: Observer aspect::Notifier
observers : Observer aspect::Observer
setState()
region
notify(s)
update(s)

However, automatic processing of these dynamic aspects specified with such a declarative formalism is still under research

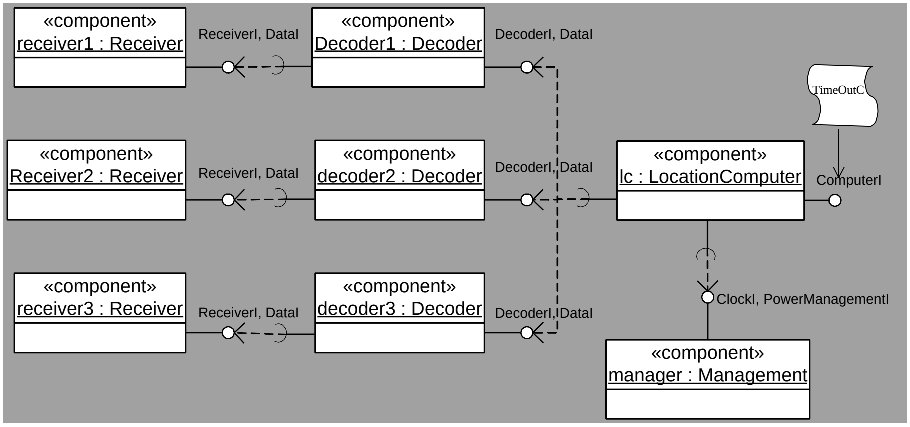
Observer Aspect Instantiation


«aspect»
Observer aspect
<<bind>>(Sensor,doRead,ValueDisplay,update)
Sensor with aspect
Sensor with aspect::Sensor
+getTemp() : float
+doRead()
+attach(in obs : ValueDisplay)
+detach(in obs : ValueDisplay)
Sensor with aspect::ValueDisplay
+update()

Result of Weaving



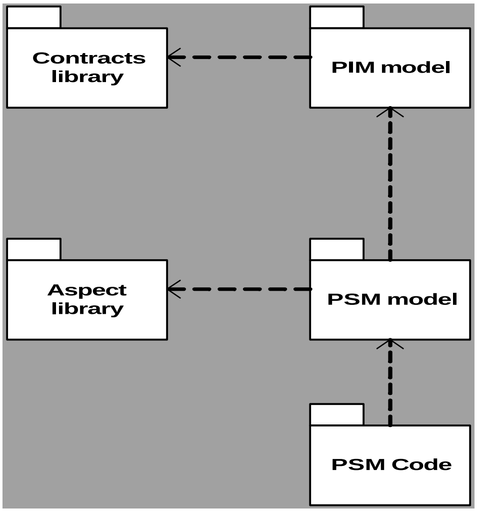
Back to the GPS example



Implementation of Contract Checking

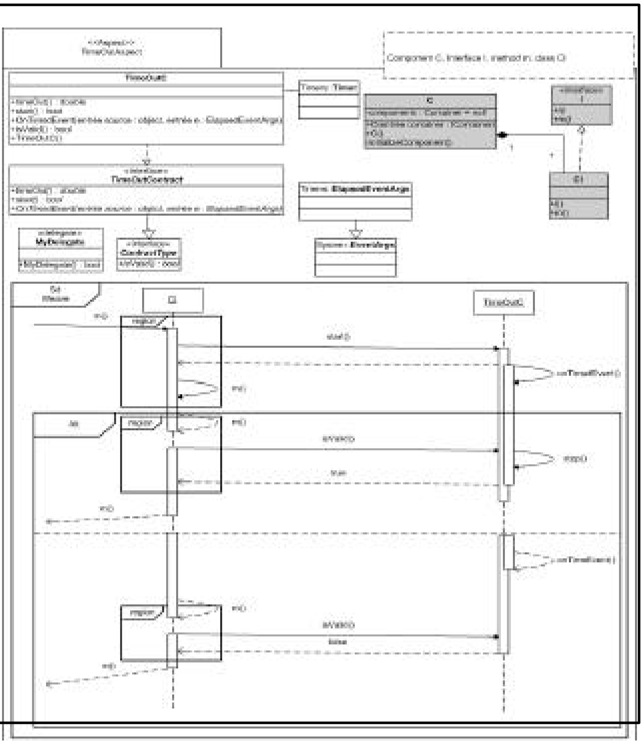
- How to graft checking code onto existing application code?
- For real-time related contracts, contract checking code can be tricky & tedious
  - Specialist task, hard to devise general purpose solutions
  - Platform dependent
- Definition as an aspect

Contracts, Aspects and MDA



Timeout Aspect Definition

Practical solution: Resort to some meta-level explicit weaving code…

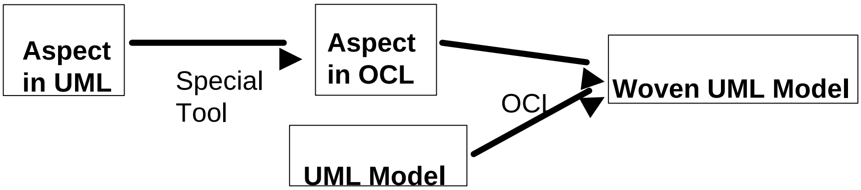


Aspect weaving in UML

- Generic aspect weaver
  - Interprets the UML model, looking for
    - » Aspect invocation
    - » Aspect signature
    - » Multiplicities, etc.
- Using aspects
  - Aspect developer ?Aspect user
  - Using an aspect means running the aspect weaver
  - Problem: Special tool support needed

Aspect Weaving Based on OCL2 Meta-Level Interpretation

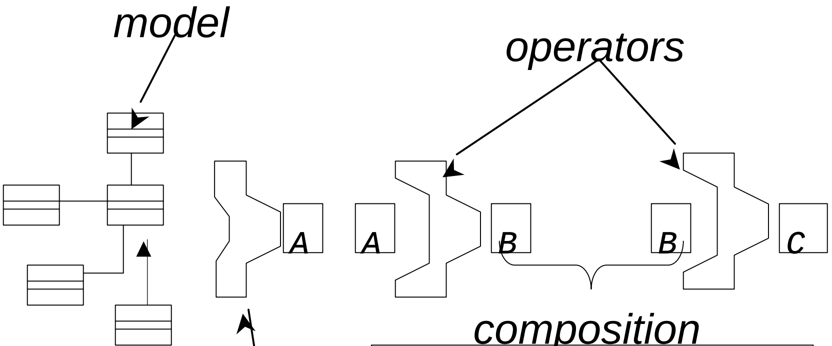
- Solution: Use “standard” UML tools for weaving
  - OCL 2.0
  - Action Semantics Language
- Transform an aspect into an OCL 2.0 expression
  - Weaving the aspect = Executing the OCL expression



Implementation (1998-2001): Using UMLAUT transformation engine

- Semantics of annotation interpreted by transformation rules
  - The same annotation can be interpreted differently depending on context
- Extensible framework
  - transformations = reusable components
  - expressed as compositions of other transformations down to primitive operations

Implementation of the engine



model

operators

A A B B C

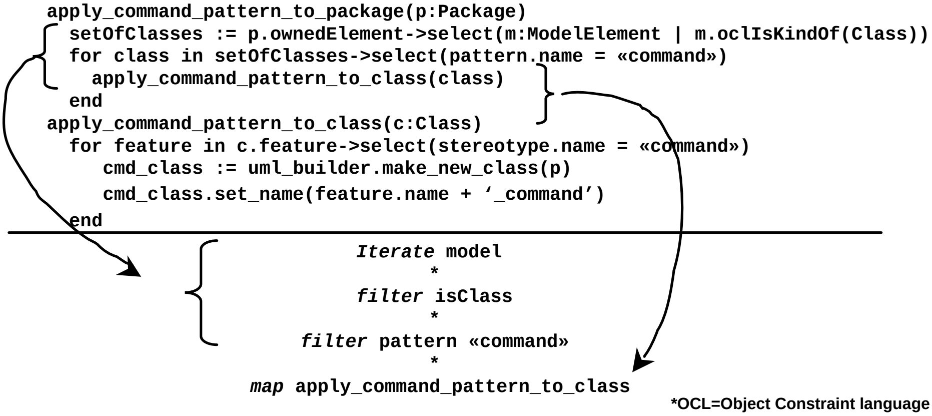
composition

iterator

`map, filter, reduce, …`
`(borrowed from BMF)`

Ongoing work (2002-?): Integration with OCL / AS*

- Transformation engine driven by OCL / AS.
  - User-defined M2-level (Meta-Model) manipulations



```
apply_command_pattern_to_package(p:Package)
  setOfClasses := p.ownedElement->select(m:ModelElement | m.oclIsKindOf(Class))
  for class in setOfClasses->select(pattern.name = «command»)
    apply_command_pattern_to_class(class)
  end
apply_command_pattern_to_class(c:Class)
  for feature in c.feature->select(stereotype.name = «command»)
     cmd_class := uml_builder.make_new_class(p)
     cmd_class.set_name(feature.name + '_command')
  end
```

```
            Iterate model
                  *
            filter isClass
                  *
        filter pattern «command»
                  *
map apply_command_pattern_to_class
```

*OCL=Object Constraint language
AS=Action Semantics

Tiny Example: Transform public attributes into private ones and add accessors

```
processPackage(p:Package)
-- For each public Attribute of each Class of the Package,
-- we apply the privatizeAttribute transformation
 forAll attribute in
      p.ownedElement->select(m:ModelElement |
            m.oclIsKindOf(Attribute)
            and m.visibility = #public
            and m.owner.oclIsKindOf(Class)
            ) {
                  privatizeAttribute(attribute)
 }
```

Example (cont.): privatizeAttribute

```
privatizeAttribute(a:Attribute)
   -- Set attribute a to private & create public setter/getter
   a.visibility := #private
   -- create setter & link it
   add_link(a.owner.feature, -- the enclosing class’ features
           newSetter(a))
  -- create getter & link it
   add_link(a.owner.feature, -- the enclosing class’ features
           newGetter(a))
```

Example (cont.): newSetter

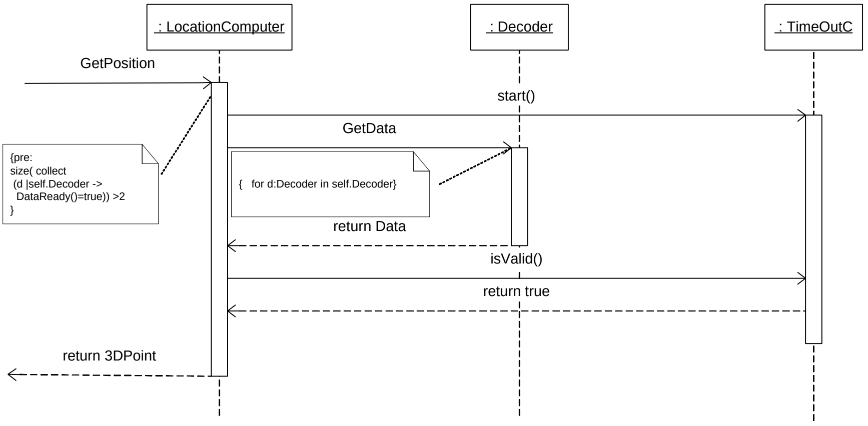
```
newSetter(a:Attribute) : Operation
   -- Creates a setter for attribute a
  Operation result := new Operation()
  result.visibility := #public
  result.name := 'set_' + a.name
  -- We prepare the input parameter
  Parameter newName := new Parameter()
  newName.name := 'new_' + a.name
  newName.kind := #in
  add_link(newName.type, a.type)
  add_link(result.parameter, newName)
  -- return result
```

Example (cont.): newGetter

```
newGetter(a:Attribute) : Operation
   -- Creates a getter for attribute a
  Operation result := new Operation()
  result.visibility := #public
  result.name := ‘get_' + a.name
  -- We prepare the return parameter
  Parameter returnParam := new Parameter()
  returnParam.kind := #return
  add_link(returnParam.type, a.type)
  add_link(result.parameter, returnParam)
```

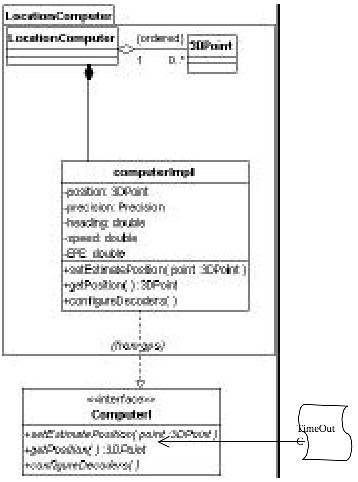
Weaving TimeOut Contracts into our GPS device

- E.g. Sequence diagram for nominal behavior



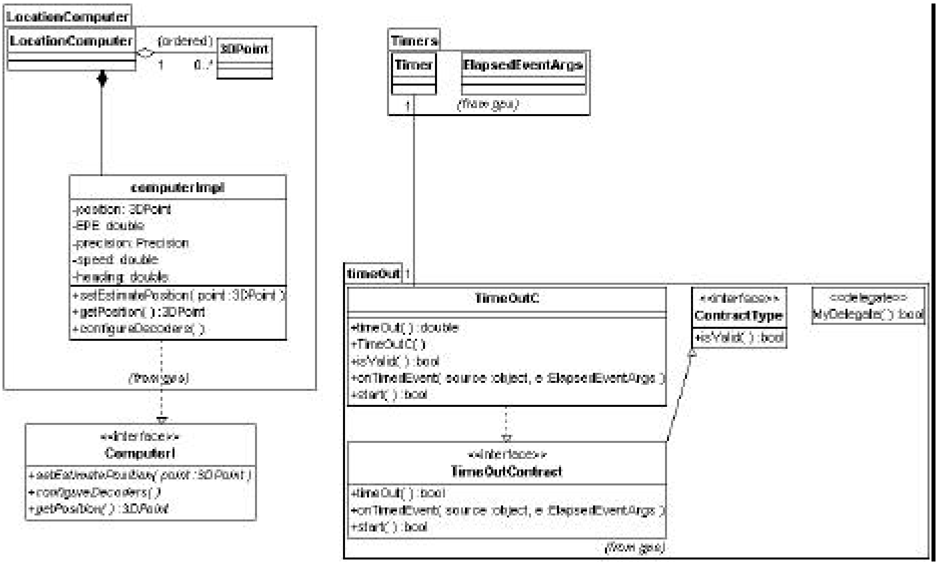
Step 1

- Check that the contract can be applied here



With a lot of boring OCL code

Step 2: add required packages

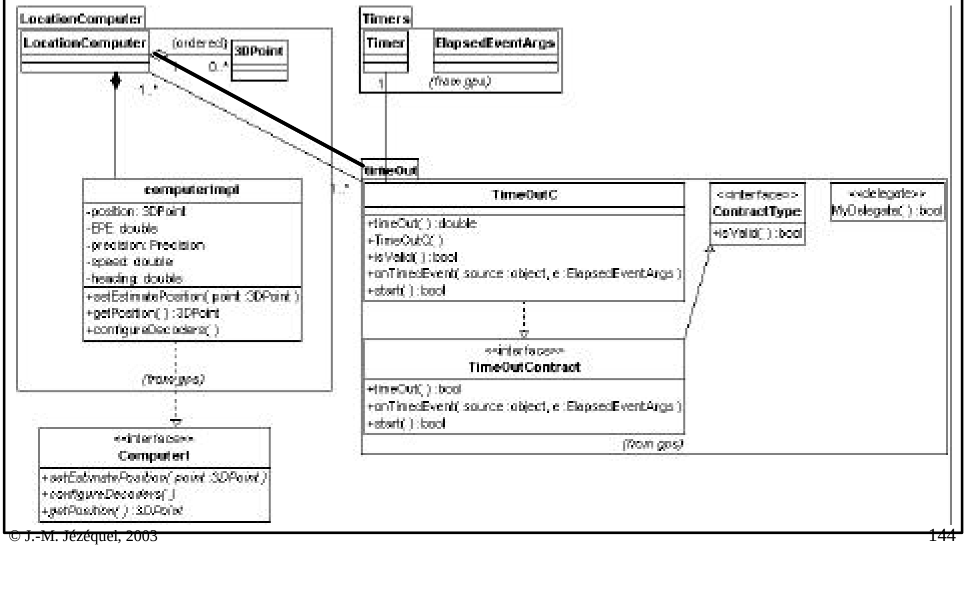


Step 2 code

```
class Contract {
    attribute:
        # reference : Class;

    constructor:
        + (name : String) {
                importContractModel;
                self(Component::Core::Classes::Class.allInstances->any(i |
    i.qualifiedName = name));
        }

        + (reference : Class) {
                self.reference := reference;
        }
    }
```

Step 3

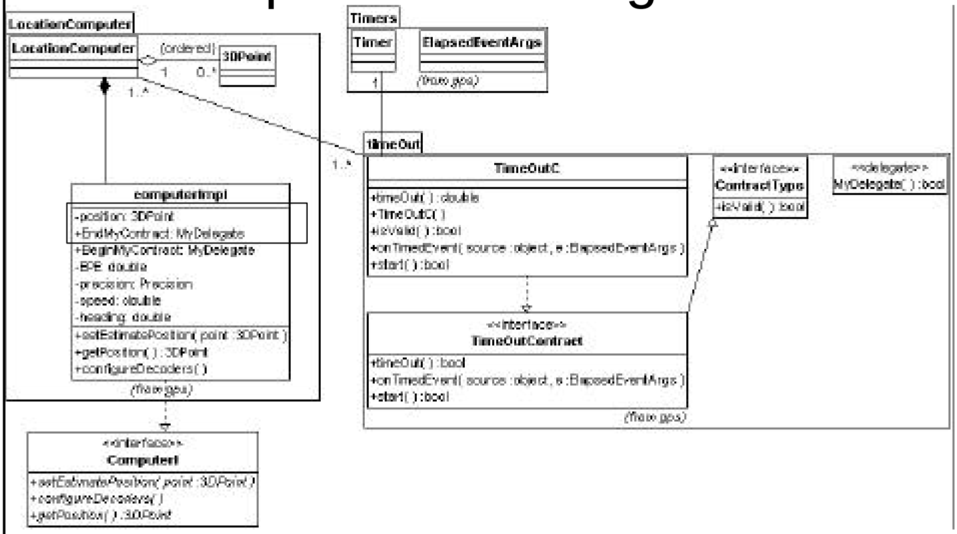
- Add association LocationComputer-TimeOutC



Step 3 code

```
class Contract {
// cont. From previous slide
    if not component.getBaseClass.feature->
          select(oclIsKindOf(EndPoint)).otherEnd.featuringClassifier->
                                          includes(self.reference) then

          UMLManager.associate(Sequence{component.getBaseClass,
    self.reference}, Sequence{1, 1}, Sequence{1, 1}, Sequence{true, true},
    Sequence{'',''})
    endif
    …
```

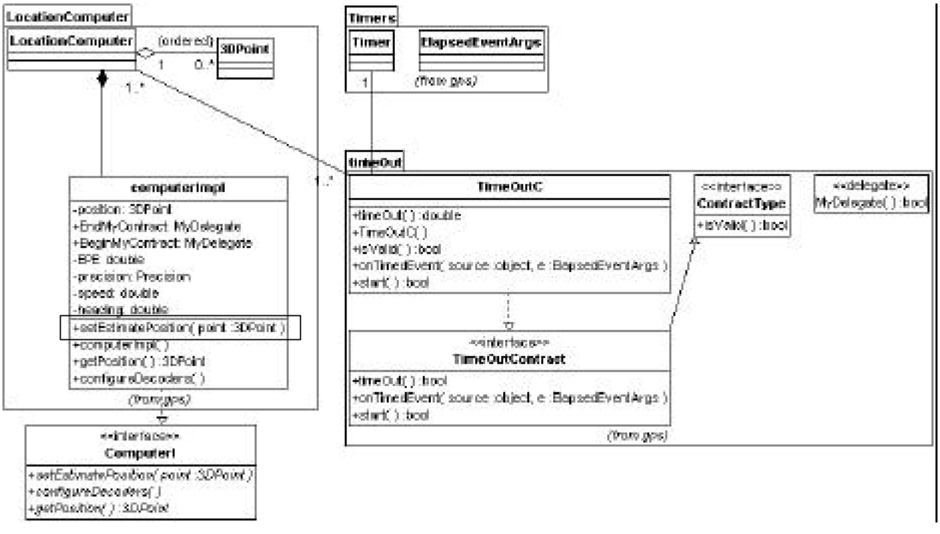
Step 4: add "delegates"



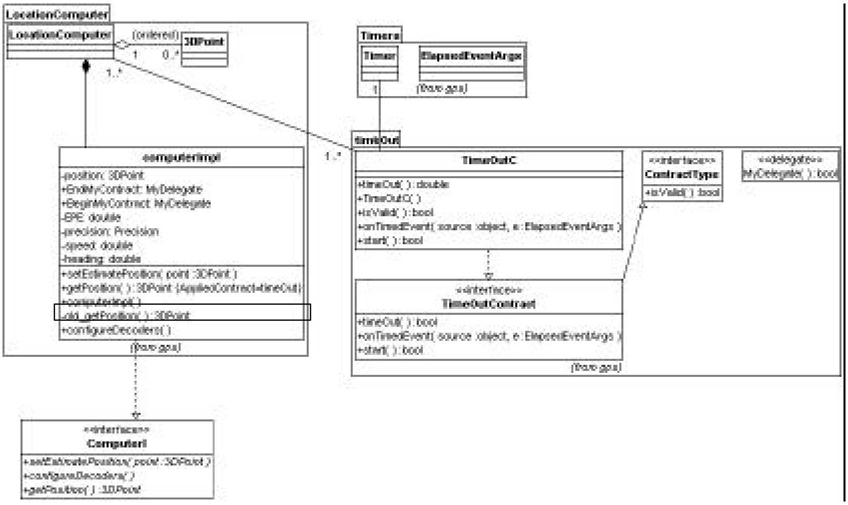
Step 4 code

```
UMLManager.associate(Sequence{subComponent.getReference, getMyDelegate}, Sequence{1, 1}, Sequence{0, 0}, Sequence{false, true}, Sequence{'','BeginMyContract'})

UMLManager.associate(Sequence{subComponent.getReference, getMyDelegate}, Sequence{1, 1}, Sequence{0, 0}, Sequence{false, true}, Sequence{'','EndMyContract'})

UMLManager.setTaggedValue(subComponent.getReference, getTagDefinitionIdentifier, self.reference.name)
```

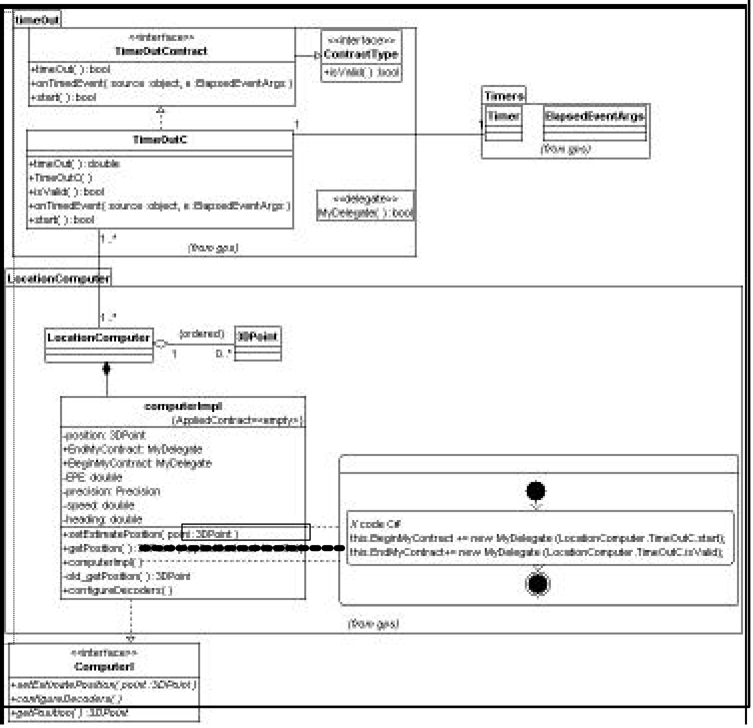
Step 5: create constructor



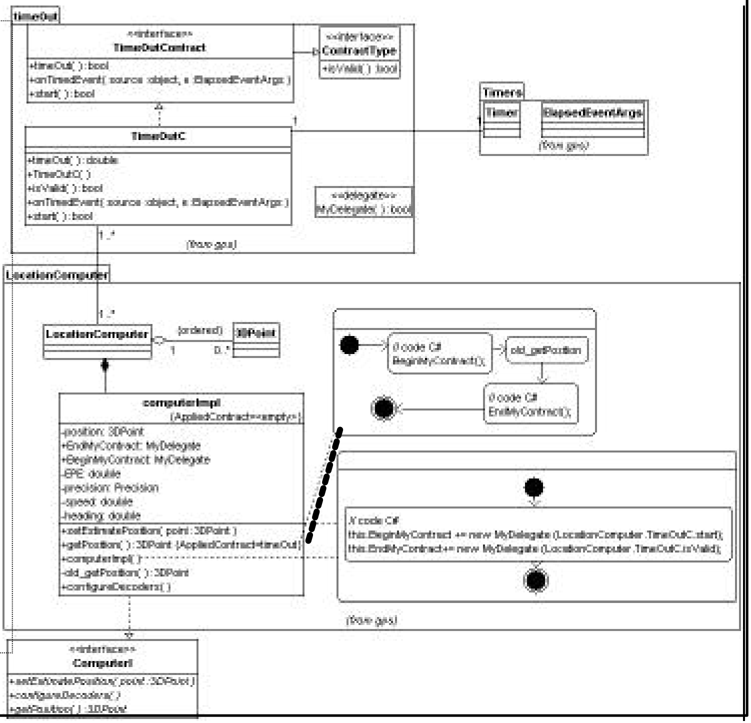
Step 6: rename setPrecision, duplicate its prototype



[redacted]



Step 8: associate Activity to *getPosition*



IRISA

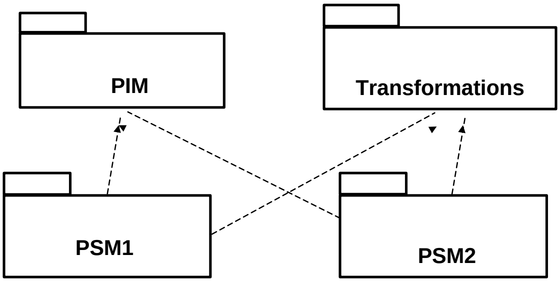
V. Wrap up

Transformations are Assets

- Must be Modeled
  - with the UML, using the power of OO
- Must be Designed
  - Design by Contract (of course), 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

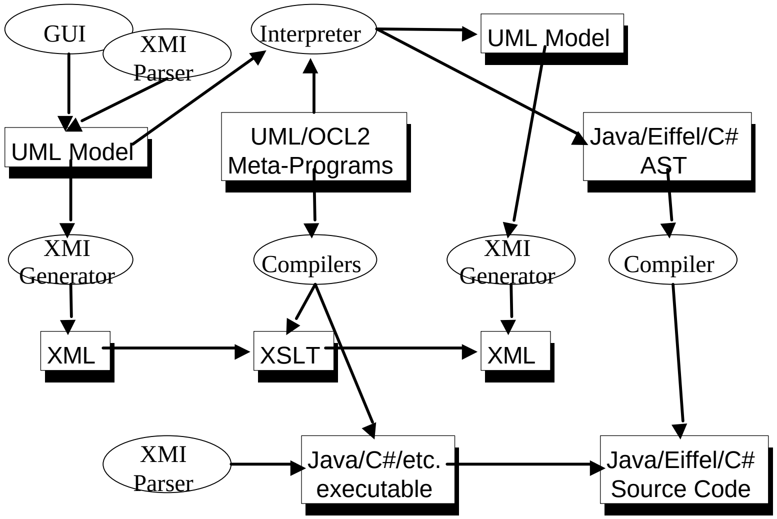
Looking into the future

- Model of PIM and Model of Transformation side by side on the CASE tool



- Rely on
  - Libraries & Frameworks at M2 level
  - The unifying notion of *aspect* to give the power of meta-modeling & model transformation to the masses

Our UMLAUT New Generation



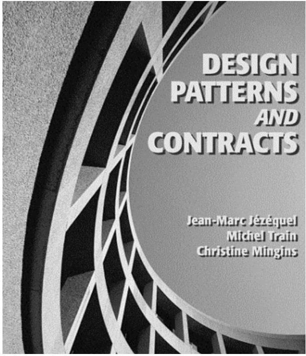
© J.-M. Jézéquel, 2003

Conclusion

- Model Driven Engineering is really about weaving Aspects at model level
  - MDA focuses on PIM->PSM
- Contracts & Patterns can be used to abstract Aspects within UML
  - Aspect Oriented Design ?
  - Designing with aspects: still a research avenue (cf. AOSD)
- UMLAUT: an OO Framework
  - for working at meta-model level
  - with operators combined in BMF style
- Towards a Model Transformation Language
  - RFP Q/V/T

© J.-M. Jézéquel, 2003

References



- “Design Patterns and Contracts”
  - *Addison-Wesley*, *1999*. ISBN 0-201-30959-9
- “Making Components Contract Aware”, *IEEE Computer, July 1999*
  - A. Beugnard, J.-M. Jézéquel, N. Plouzeau, D. Watkins
- “Precise modeling of design patterns’’, - In Proc. UML2000, LNCS 1939
  - A. Le Guennec, G. Sunyé, and J.-M. Jézéquel
- “A toolkit for weaving aspect oriented UML designs.’’ – In Proc. of AOSD 2002,
  - W.M. Ho, J.-M. Jézéquel, F. Pennaneac'h, and N. Plouzeau.