A Unified Approach to Modeling and Programming

Ole Lehrmann Madsen
The Alexandra Institute
& Aarhus University

Co-author
Birger Møller-Pedersen
University of Oslo

Dahl & Nygaard: ACM Turing Award Winners

Dahl & Nygaard at the time of Simula’s development

“for their role in the invention of object-oriented programming, the most widely used programming model today.” (ACM press release)

More than 30 years of collaboration

Kristen Nygaard

Birger Møller-Pedersen

Co-author of accompanying paper

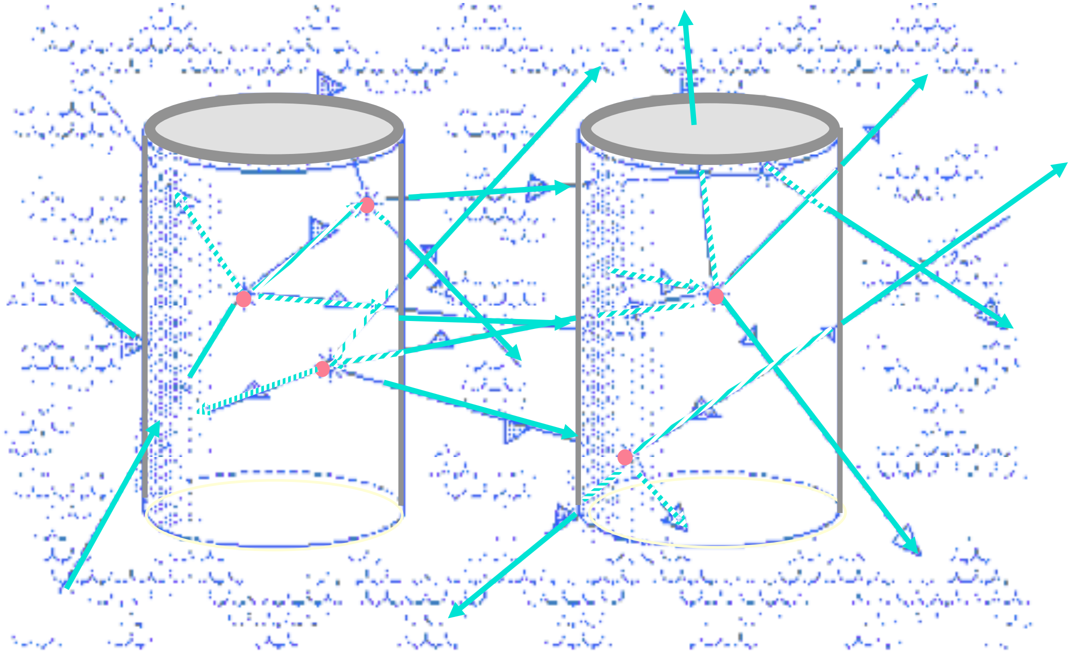
Bent Bruun Kristensen

Contents

- SIMULA: back to the future
- What is a model?
- The conceptual framework of OO
- Language issues

SIMULA: back to the future

The Atom:
From Mathematics to Monte Carlo



$$f(x) = g(x) + \lambda \int_{\Omega} K(x, y) f(y) dy$$

The SIMULA I language report from 1965 opens with these sentences:

"The two main objects of the SIMULA language are:

To provide a language for a precise and standardised description of a wide class of phenomena, belonging to what we may call "discrete event systems".

To provide a programming language for an easy generation of simulation programs for "discrete event systems"."

SIMULA I opening sentences (98-08-11)

The past

NΣR

- SIMULA -

A LANGUAGE FOR PROGRAMMING AND
DESCRIPTION OF DISCRETE EVENT SYSTEMS.
INTRODUCTION AND USER'S MANUAL
BY
OLE-JOHAN DAHL AND KRISTEN NYGAARD

SIMULA I

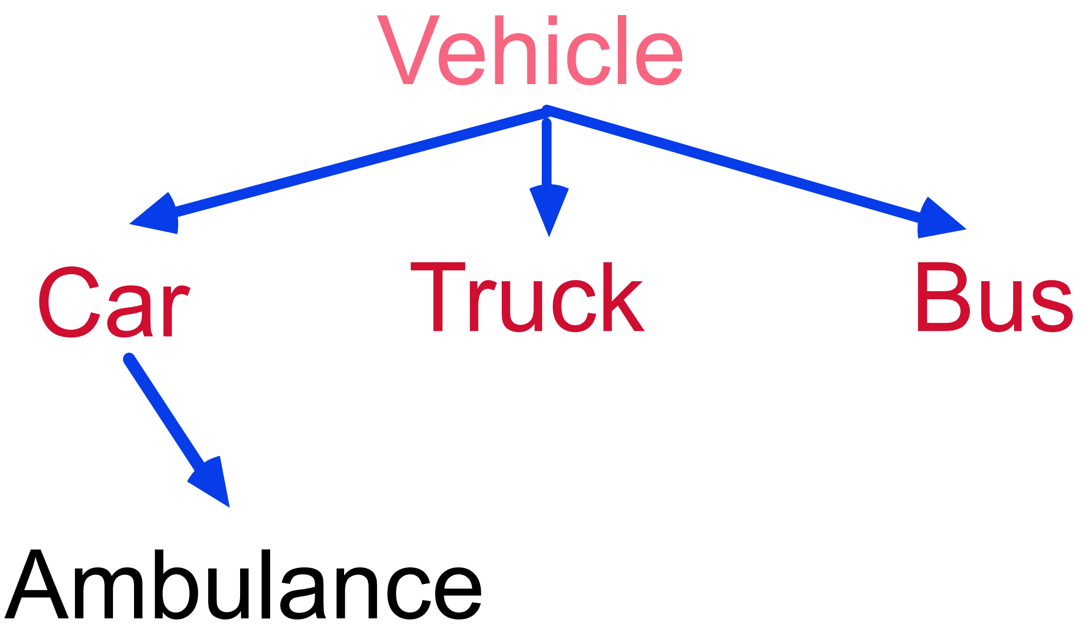
was a simulation programming language
that turned out to become
a powerful general programming language.

SIMULA 67

is a general programming language
that also is a powerful platform
for other, specialised programming languages,
as e.g. simulation languages.

SIMULA I and SIMULA 67

SIMULA 67 was triggered off by the invention of inheritance in January 1967.



Vehicle

Car

Truck

Bus

Ambulance

Virtual properties

The contributions of Simula .

- Object
- Class
- Subclass – single inheritance
- Virtual procedure (method)
- Nested/inner classes/procedures
- Action combination
  - inner

The contributions of Simula ..

- Active objects
  - Quasiparrallel systems
  - Coroutines
- Processes and schedulers
  - Used for defining process scheduling in class Simulation
  - Later extended to concurrency
  - Necessary to be able to control scheduling
- Concurrency abstractions

The contributions of Simula ...

- The first examples of application frameworks
  - Class Simulation
  - A domain specific language with special syntax

- Automatic memory management
  - Garbage collection

Some lessons from SIMULA

- SIMULA was created as a means for making simulation models – motivated by operations research
- SIMULA was a general purpose programming language
- SIMULA was used for implementation ***and*** analysis and design
- The application domain was reflected in the programs
- No need for structured analysis and structured design
    - SIMULA was a programming language with a built-in method

The BETA language

- For modeling and programming
- *Bent Bruun Kristensen, Ole Lehrmann Madsen, Birger Møller-Pedersen, Kristen Nygaard*
- Generalized abstraction mechanisms
  - Pattern, virtual class, etc.
- Concurrency, alternation, etc.
- An associated conceptual framework
- *History of Programming Languages III, San Diego 2007*

Benefits of object-orientation

- Unifying perspective on most phases of the software life cycle
  - Common language core:
    class, subclass, virtual, method, etc.
  - Common conceptual framework
- Good support for modeling
  - Programs reflect reality (the application domain) in a natural way
- Good support for programming
  - Extensibility & reuse

The situation today

- Divergence of modeling and programming
- Object-oriented programming:
  - Very little attention to modeling
- Mainstream modeling:
  - Is widening the gap to programming
- Executable models:
  - The right direction
  - A long way to go

The first duty of a programmer

"The first duty of a revolutionary
is to get away with it."
Abbie Hoffmann

"The first duty of a programmer
is to get away with it."
Hackers' Credo

To program is to understand
Kristen Nygaard

While programmers don’t use UML

- Large and incomprehensible
- All you need is code (Nierstrasz)
  - The code is the real thing
- “Very few real-world objects in my application”
- When time becomes critical, the model is dropped
  - Difficult to keep the model consistent with the code
  - Impedance mismatch, as with OOA and OOD

The model should be the code

- “There similarly appears to be something fundamentally wrong with model-driven development as it is usually understood — instead of generating code from models, the model should be the code.”

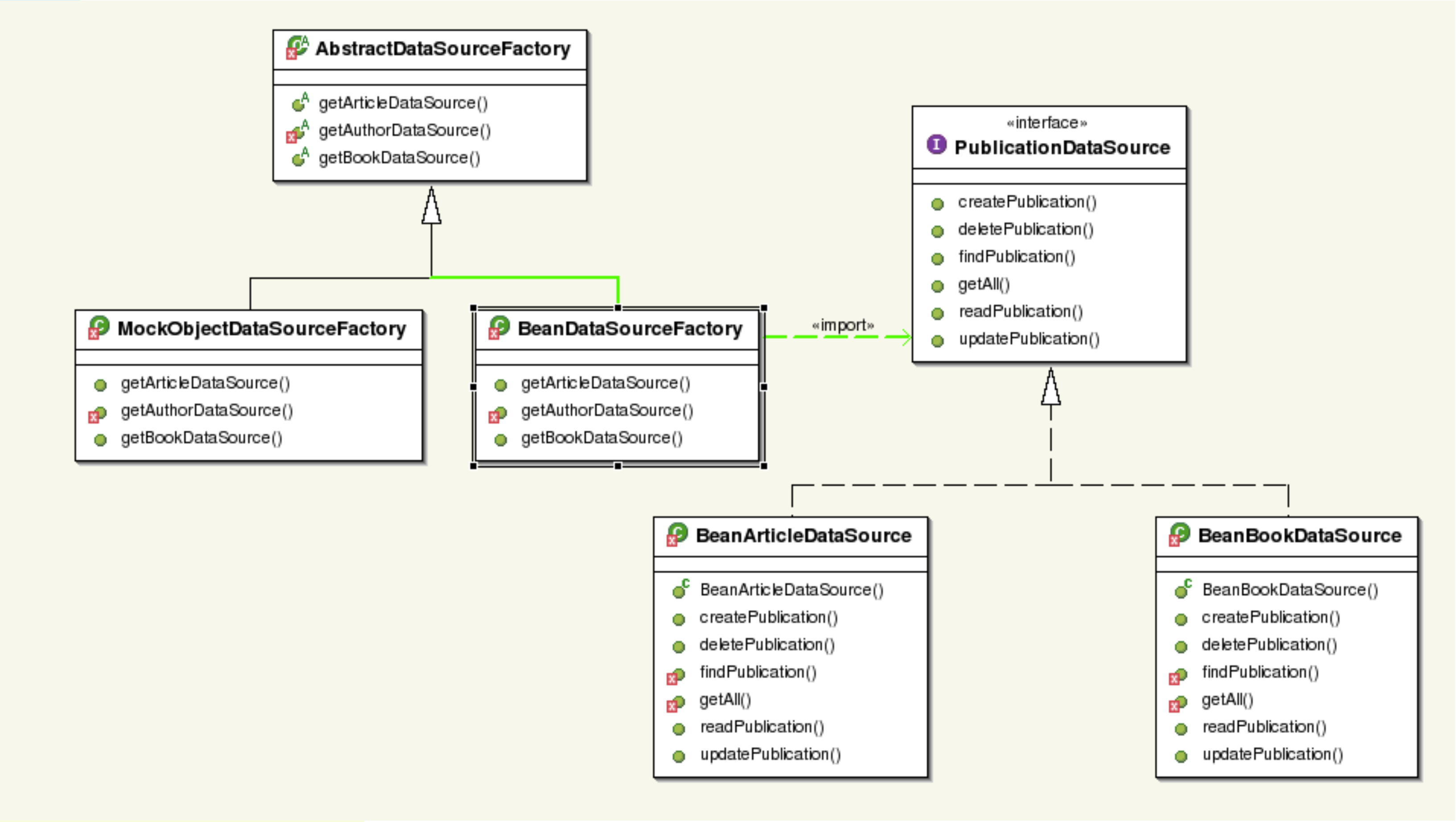
Oscar Nierstrasz:

Ten things I hate about object-oriented programming

Banquet speech, ECOOP 2010

What is a model?

Is a UML diagram a model?



Classes drive me crazy ...

“There is a complete disconnect in OOP between the source code and the runtime entities. Our tools don’t help us because our IDEs show us classes, not objects.

I think that’s probably why Smalltalkers like to program in the debugger. The debugger lets us get our hands on the running objects and program them directly.

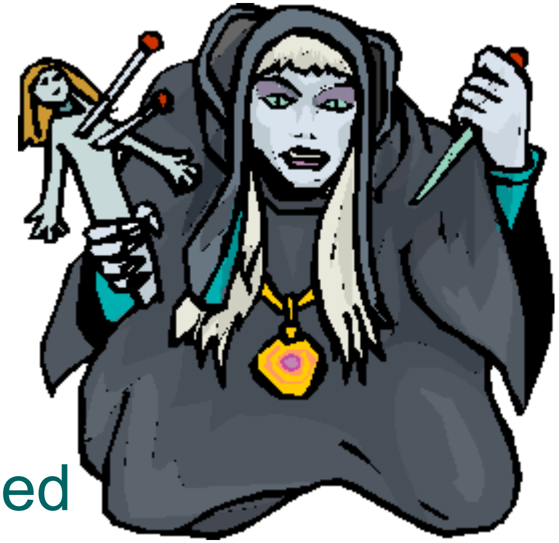
Here is my message for tool designers: please give us an IDE that shows us objects instead of classes!”

Oscar Nierstrasz:
Ten things I hate about object-oriented programming
Banquet speech, ECOOP 2010

The use of modeling

- Used in many disciplines:
  - Science, engineering, architecture, entertainment, software development, ...
- For communication of properties of systems
- For understanding properties of existing systems
- For analyzing properties before building new systems
- As toys

Primitive societies



- Models are sometimes believed to be useful in their own right

- Manipulation of the model might itself cause corresponding changes in the real world
- Sticking pins into wax models of enemies …
- Example due to Tony Hoare

Simulation metaphor

- SIMULA describes real systems and generated a simulation
- The simulation is the dynamic structure of objects evolving during execution of the program
- The simulation (**program execution**) is considered to be the **model**
- Alan Kay has described object-oriented programming as a view on **computation as simulation**

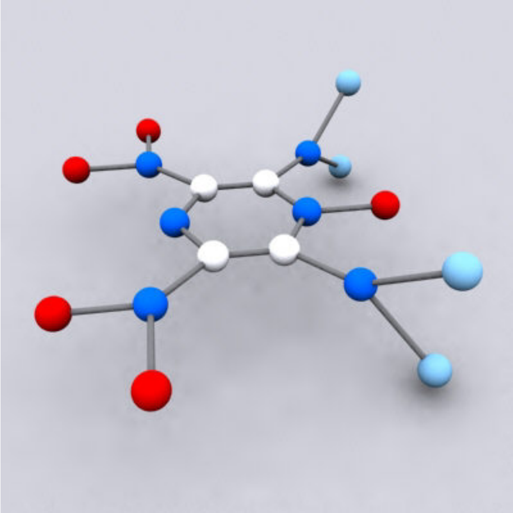
Webster:

- A model refers to a small, abstract or actual representation of a planned or existing entity or system from a particular viewpoint.

Abstract or actual

- Abstract: mathematical description
- ***Newton's laws of motion***

- Actual: something physical
- A molecule:



Planned or existing system

Model of planned building

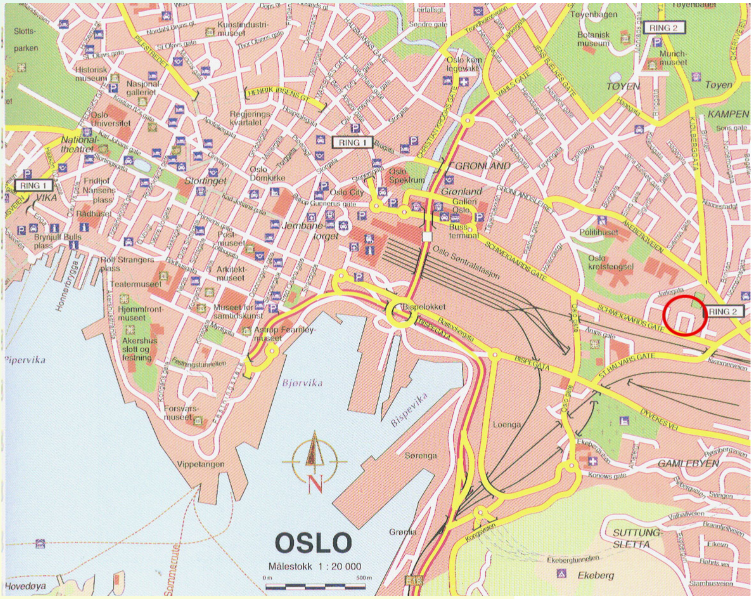
Model of (existing) solar system

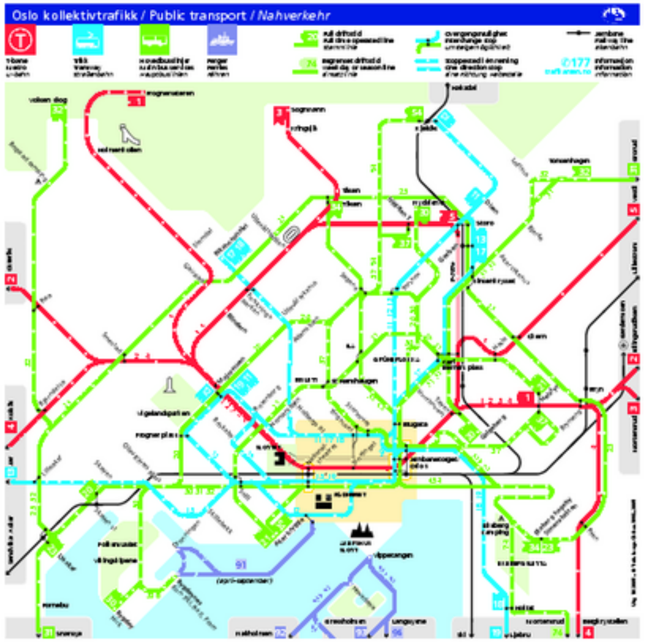
From a particular viewpoint

- A good map provides the information we need for a particular purpose – or the information the mapmaker wants us to have

Peter Turchi:

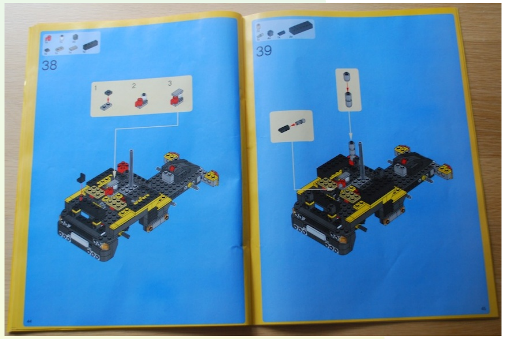
Maps of the Imagination – the writer as a cartographer





Description versus model

Description:



Model:

The (physical) material:

Object-oriented programming

A program execution is regarded as a
physical model
simulating the behavior of a
real or imaginary system

Description versus model

- The languages for making descriptions are important
- More attention to the model
  - Objects, their properties, and actions
- We need tools showing the program execution
  - More than debuggers
- The semantics of the language

Conceptual framework

The basis for modeling

- We must understand and develop the conceptual means for understanding and organizing knowledge
- Guidelines for identifying
  - Objects, properties of objects, classes, class hierarchies, methods, activities (tasks), …
- For understanding the limitations of programming languages
- The theory and semantics of OO

Current situation of modeling

- Mainstream programming languages:
- Modeling was never considered seriously
- In Smalltalk many classes are not substitutable
- Subclassing: inheritance of code
- Multiple inheritance: often complicated technicalities
- Languages with both types and classes

Status of conceptual framework

- SIMULA
  - Modeling central, but conceptual framework was implicit
- BETA:
  - The conceptual framework was developed together with the language
  - Test: programming and modeling
- Modeling languages:
  - I am sure that conceptual framework is considered important

Abstraction

- In the development of our understanding of complex phenomena, the most powerful tool available to the human intellect is abstraction

(Tony Hoare, Notes on Data Structuring.
In Dahl, Dijkstra, Hoare:
Structured Programming,
Academic Press, 1972)

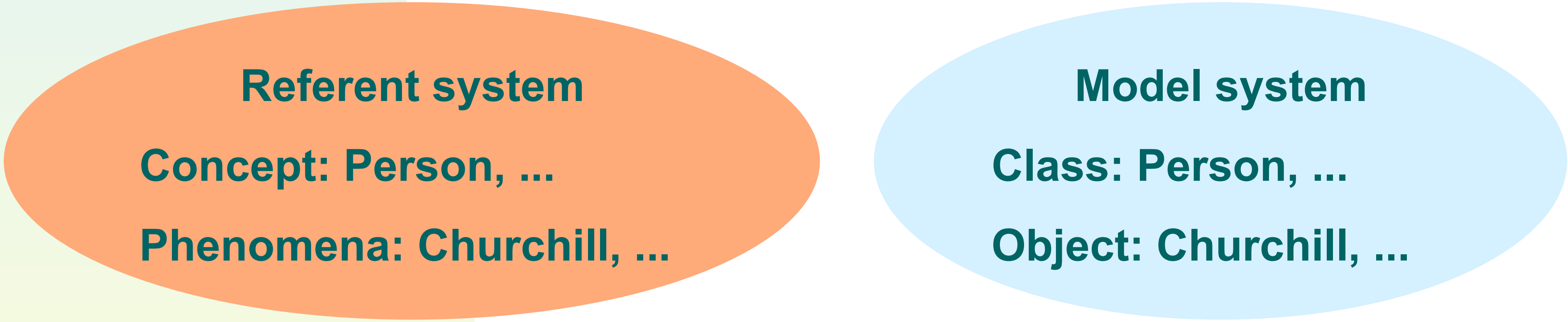
- Identification of phenomena and their properties
- Formation of concepts

Conceptual means

- Identification of objects
- Classification
  - Clustering
  - Generalization/specialization
- Composition/aggregation
  - Whole/part
  - Reference composition
  - Concept composition
- Association

Class and concept

- A concept is represented as a class
- Intension: attributes and methods
- Extension: the instances of the class



Referent system

Concept: Person, ...

Phenomena: Churchill, ...

Model system

Class: Person, ...

Object: Churchill, ...

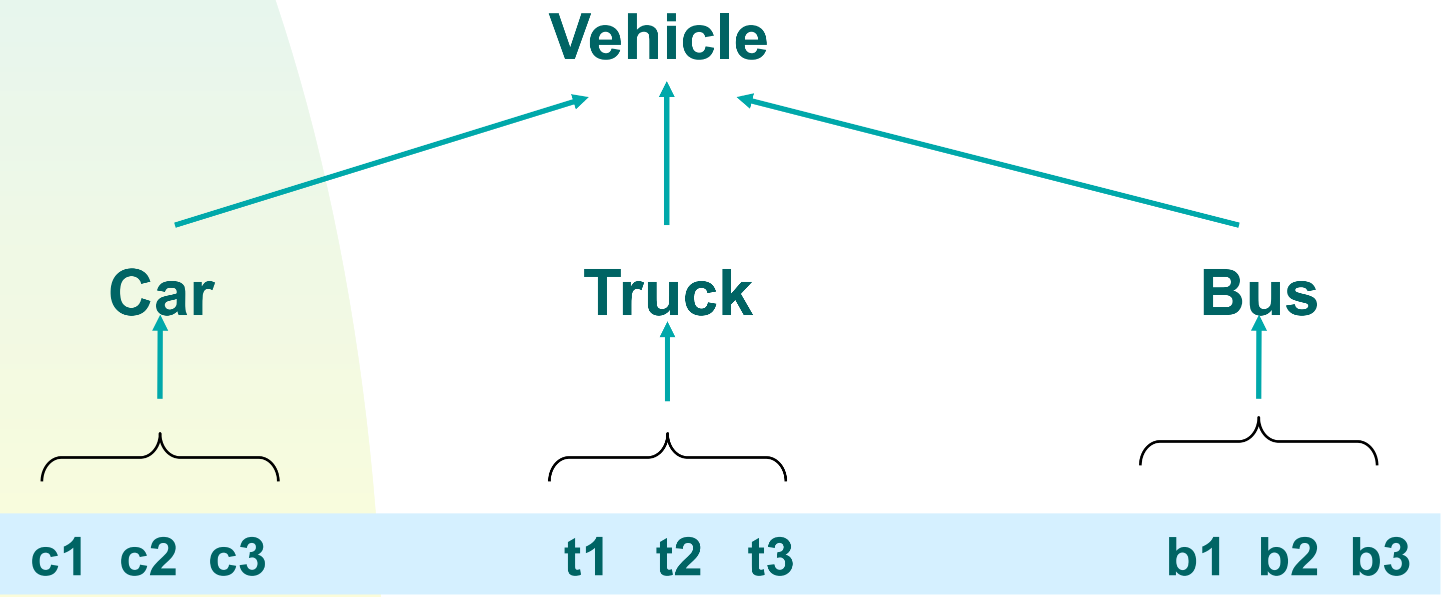
Concept hierarchies

- Subclass

  The extension of Car is a subset of the extension of Vehicle

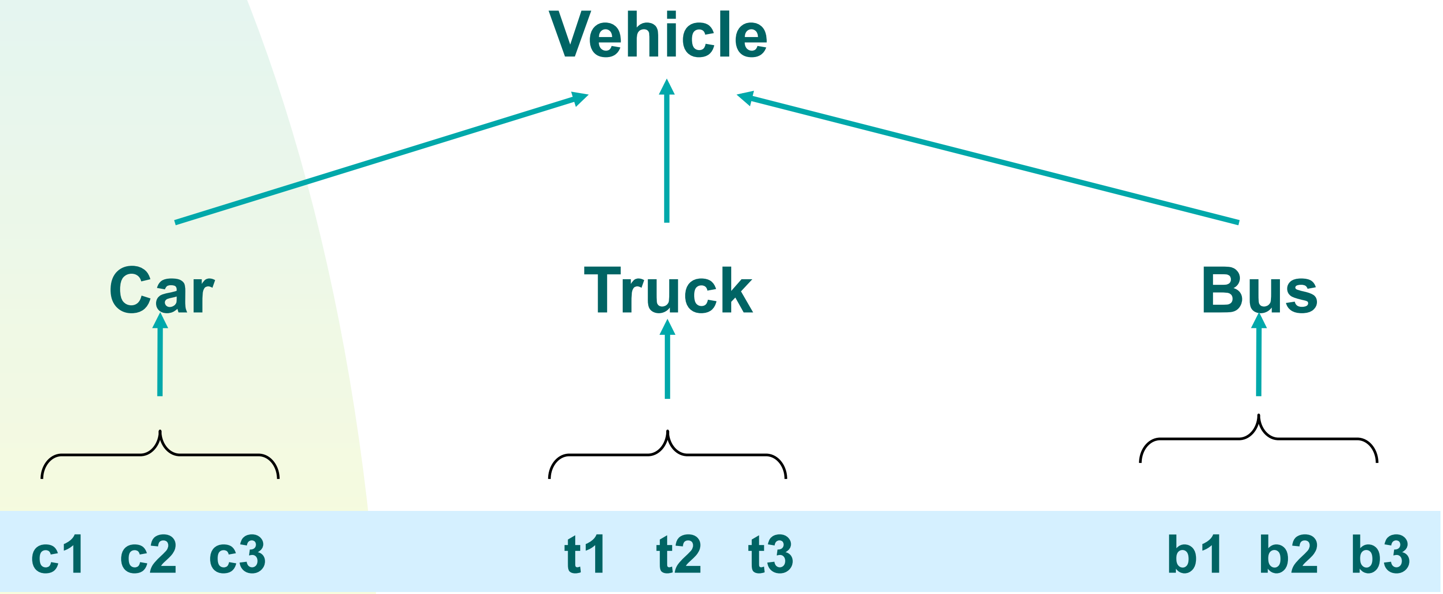
- Extends

  The intension of Car extends the intension of Vehicle



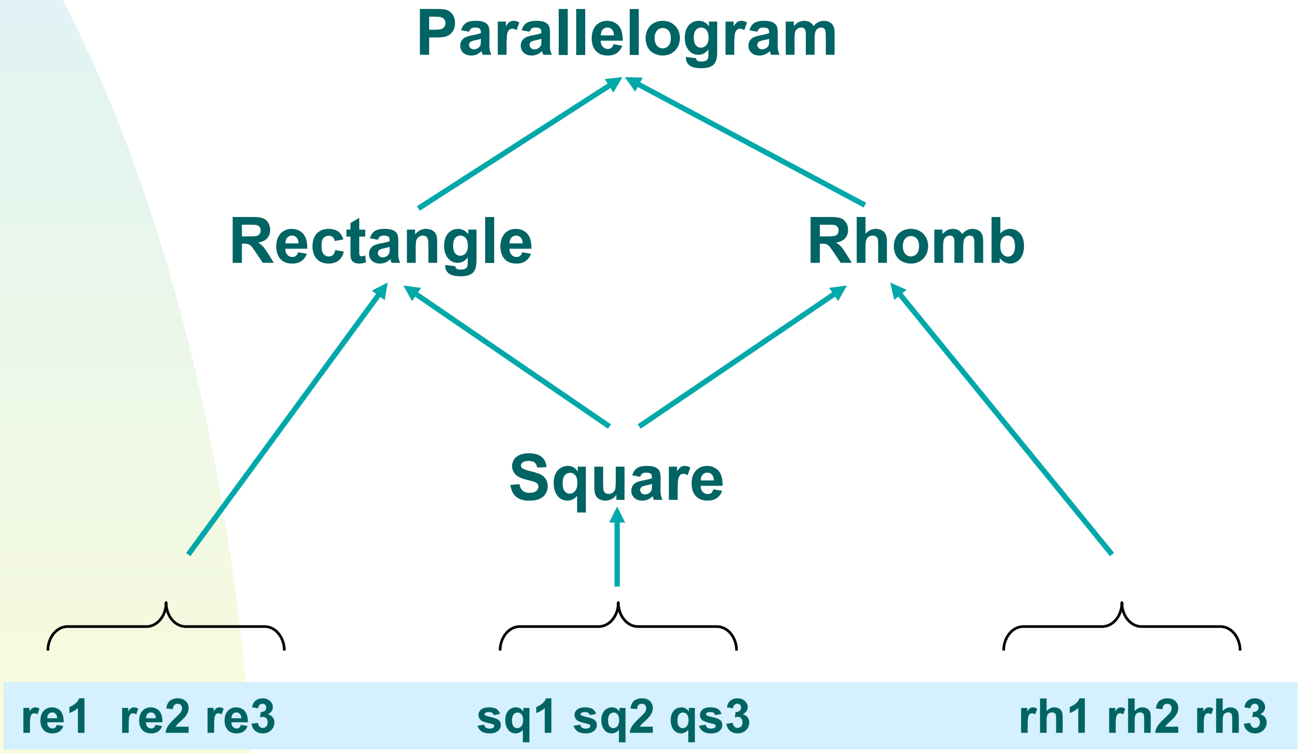
Tree structured classification

Subclasses with disjoint extensions



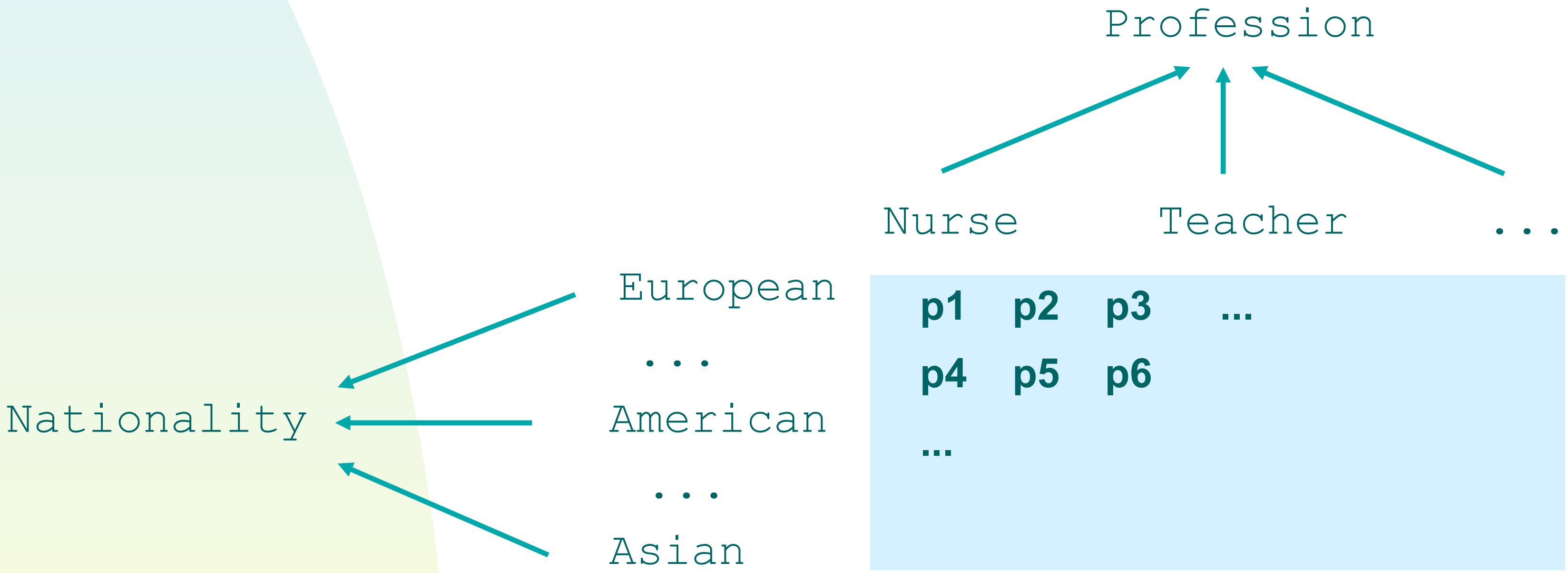
Non-tree structured classification

Subclasses with overlapping extensions



Several classification hierarchies

Independent tree-structured classifications of the same objects



Profession

Nurse Teacher ...

Nationality

European ... American ... Asian

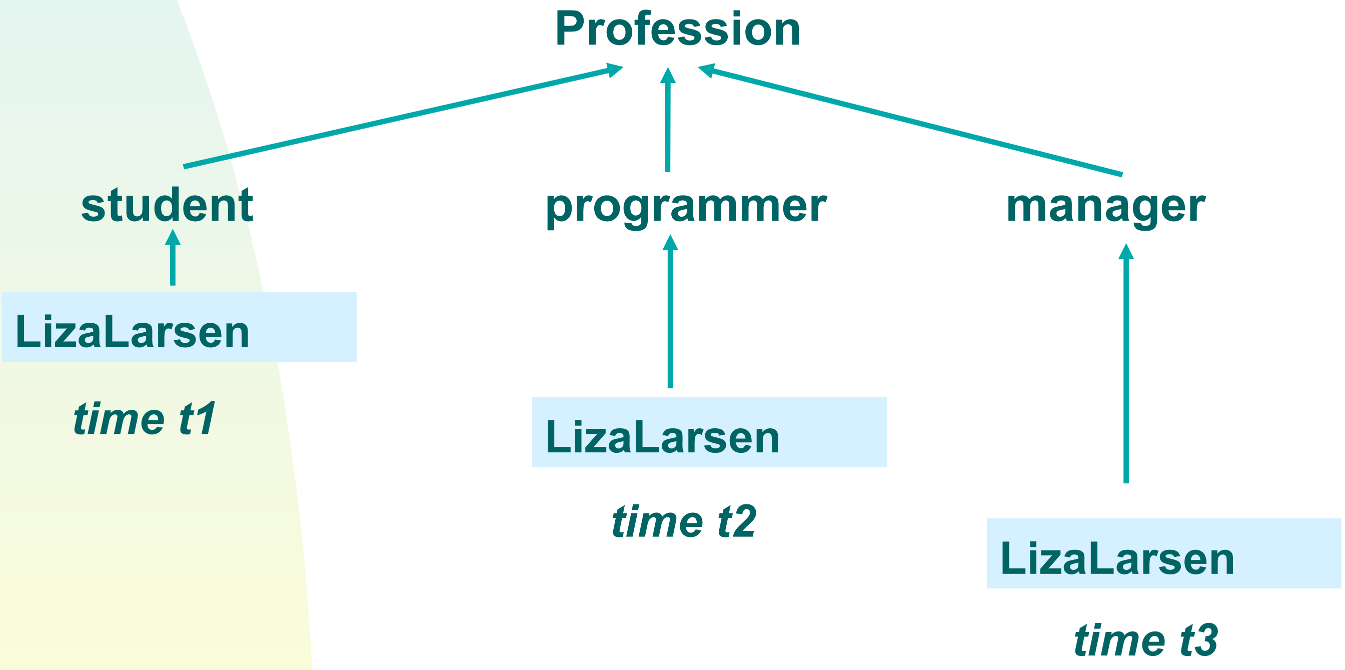
p1 p2 p3 ...

p4 p5 p6

...

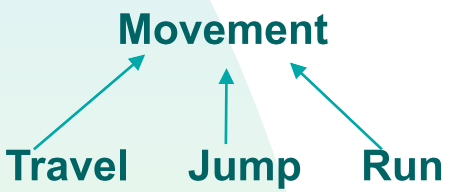
Dynamic classification

Objects may change class membership during their existence



Behavior hierarchies

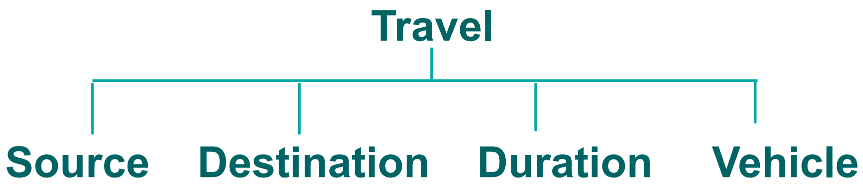
Generalization/specialization



Movement

Travel Jump Run

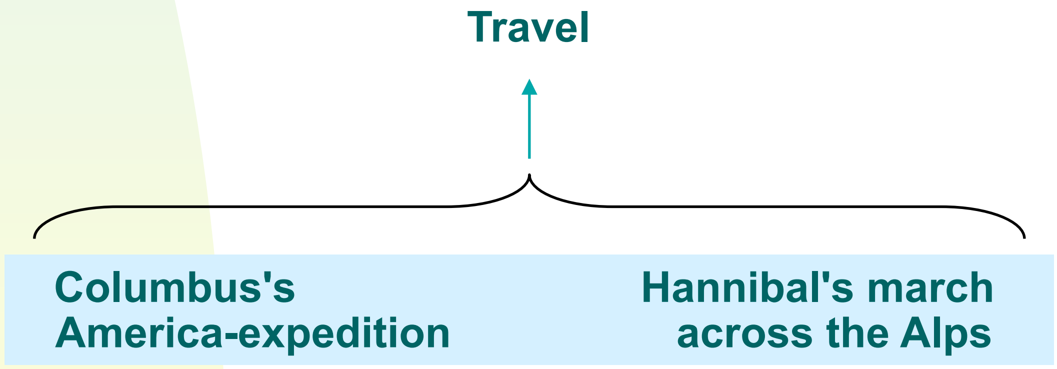
Part/whole composition



Travel

Source Destination Duration Vehicle

Clustering



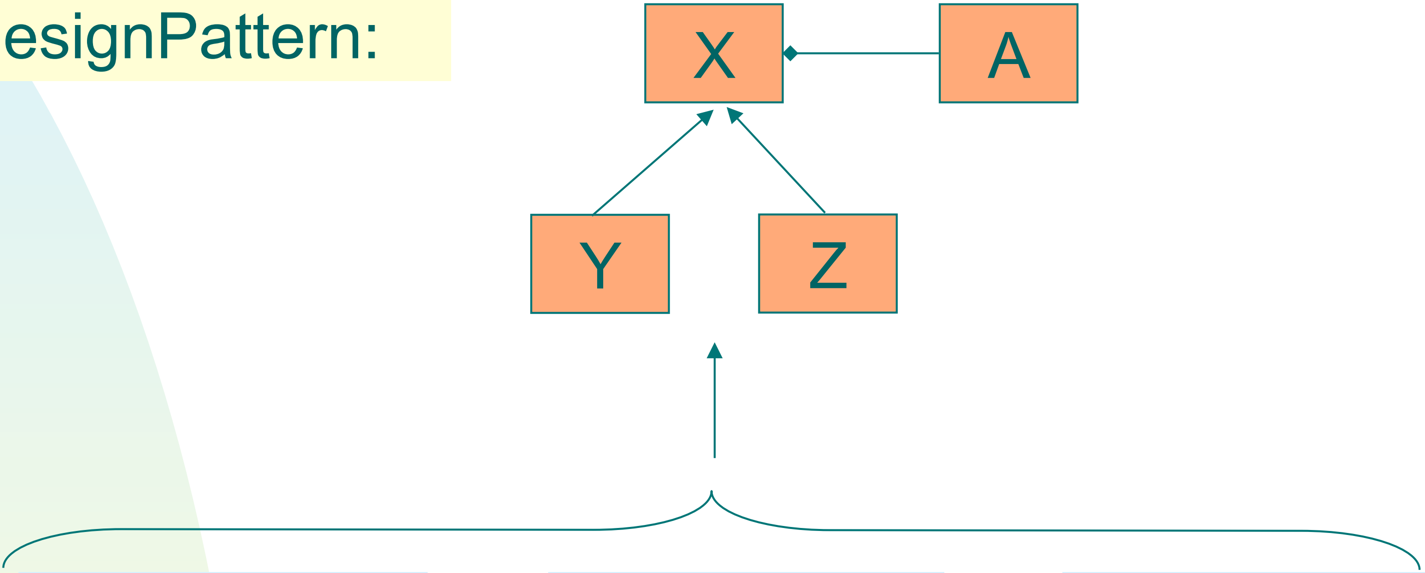
Travel

Columbus's America-expedition

Hannibal's march across the Alps

Design patterns: a challenge for abstraction

myDesignPattern:



Program fragment 1

Program fragment 2

Program fragment 3

Abstraction mechanisms

- The conceptual framework shows limitations of current abstraction mechanisms:
  - Class, type, virtual, procedure, generic, etc.
- We should derive new abstraction mechanisms

Properties of phenomena in information systems

- Physical material
- Properties of material
- Actions

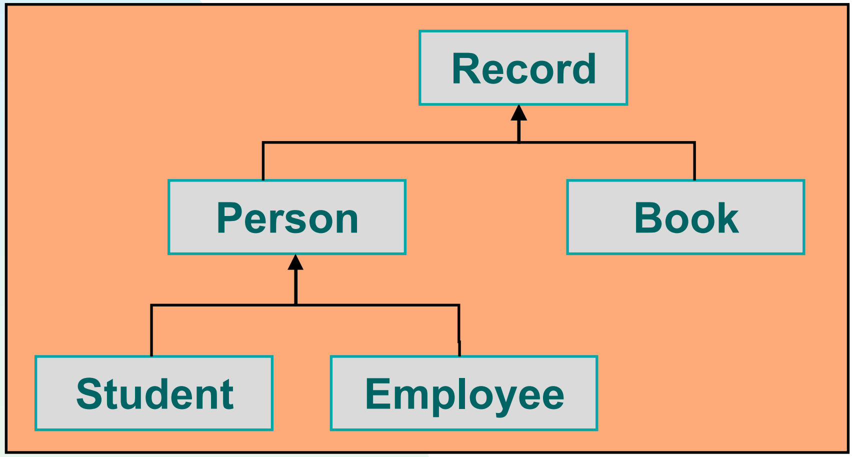
Language issues

Issues

- Syntax
- Constraints
- Domain specific languages
- Scenario descriptions
- Programming by examples

- State machines
- Associations
- Asynchronous events
- Action sequences
- Other constructs

Diagrams and code: different views



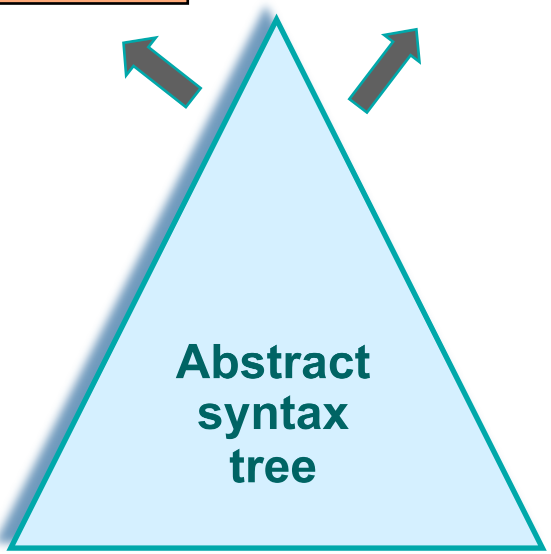
```
Record: { … };

Person: Record { … }

Student: Person{ … }

Employee: Person{ … }

Book: Record{ … }
```



Common

Representation

- One to one correspondence
- No code generation

A picture says more than a thousand words

- A word says more than a thousand pictures:

Vehicle:

Constraints

- Class invariants,
- Pre- and post- conditions
- Assertions
- Constraint-oriented programming
- Useful for programming as well
- Modeling/specification work could drive this

Scenario descriptions

- Emphasis on run-time – the model
- UML sequence diagrams, object diagrams, etc., are steps in the right direction
- Should be a part of any programming language definition
- We need a notation for the whole program execution

Programming by examples

- Derive descriptions / programs from scenarios / examples
- From interaction diagrams to state-machines
- Derivation of programs from examples

State-machines & associations

- The justifying examples for modeling languages
- Issues:
  - Precise semantics?
  - Programming language abstraction mechanisms
  - Design patterns
  - Textual or graphical notation
  - Embedding of graphical language

Asynchronous events and actions

- Relevant for programming as well as modeling
- Sequential actions
  - The same for programming and modeling
- Concurrency
  - We need better concurrency abstractions

Concurrency

- Modeling of concurrent processes from the real world was essential for SIMULA
- SIMULA had quasi-parallel processes
- Major difference between modeling languages and programming languages
- Challenge:
  - Strong need for better mechanisms for modeling and implementing concurrent processes

Executable modeling language

- Must subsume state-of-art from programming languages
- Must be just as efficient as programming languages
- No need for modifying the generated code
- Just a new higher level programming language
- A long way to go

Summary

A unified approach

- Language for modeling and programming
  - Executable
  - Non-executable
- Model: program execution
  - Properties, notation, tools
- Conceptual framework

Back to the future

- SIMULA:
  the strength of a unified approach to
  modeling and programming is fading away
- Re-unite the forces
  - Don’t let programmers get away with it
  - Don’t let modelers just create bubbles