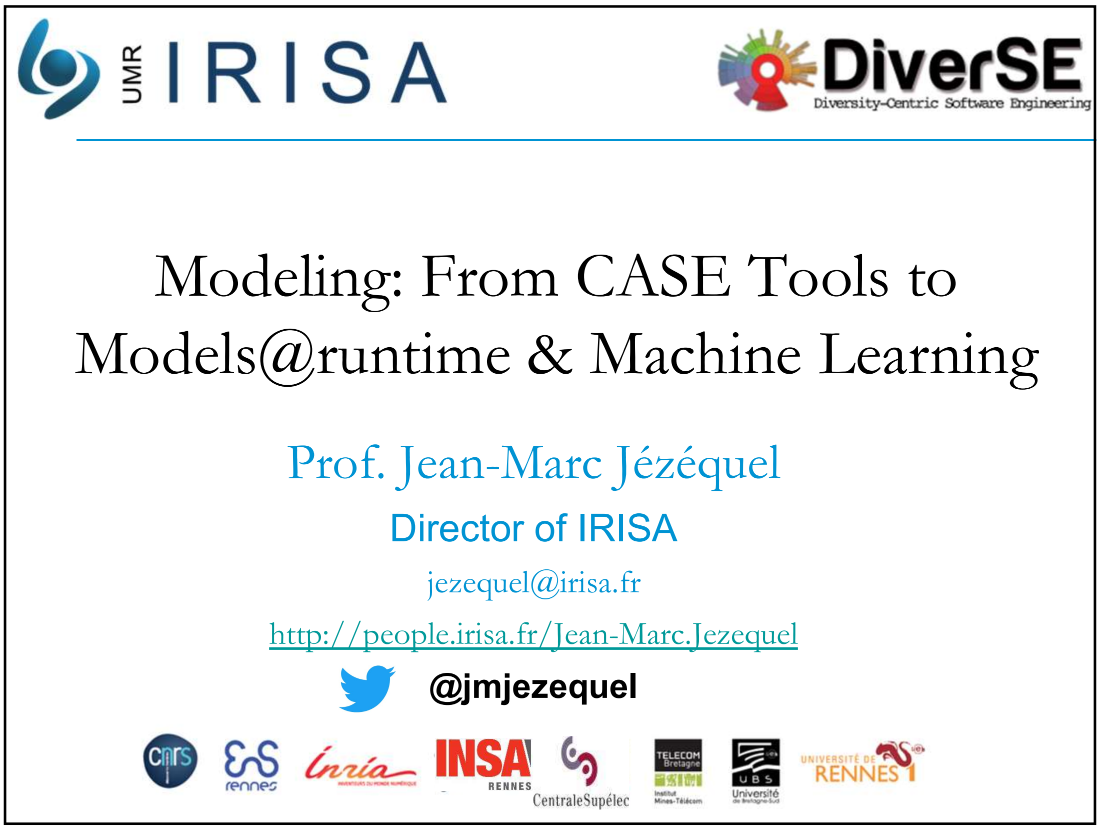


UMR IRISA

DiverSE
Diversity-Centric Software Engineering

Modeling: From CASE Tools to Models@runtime & Machine Learning

Prof. Jean-Marc Jézéquel

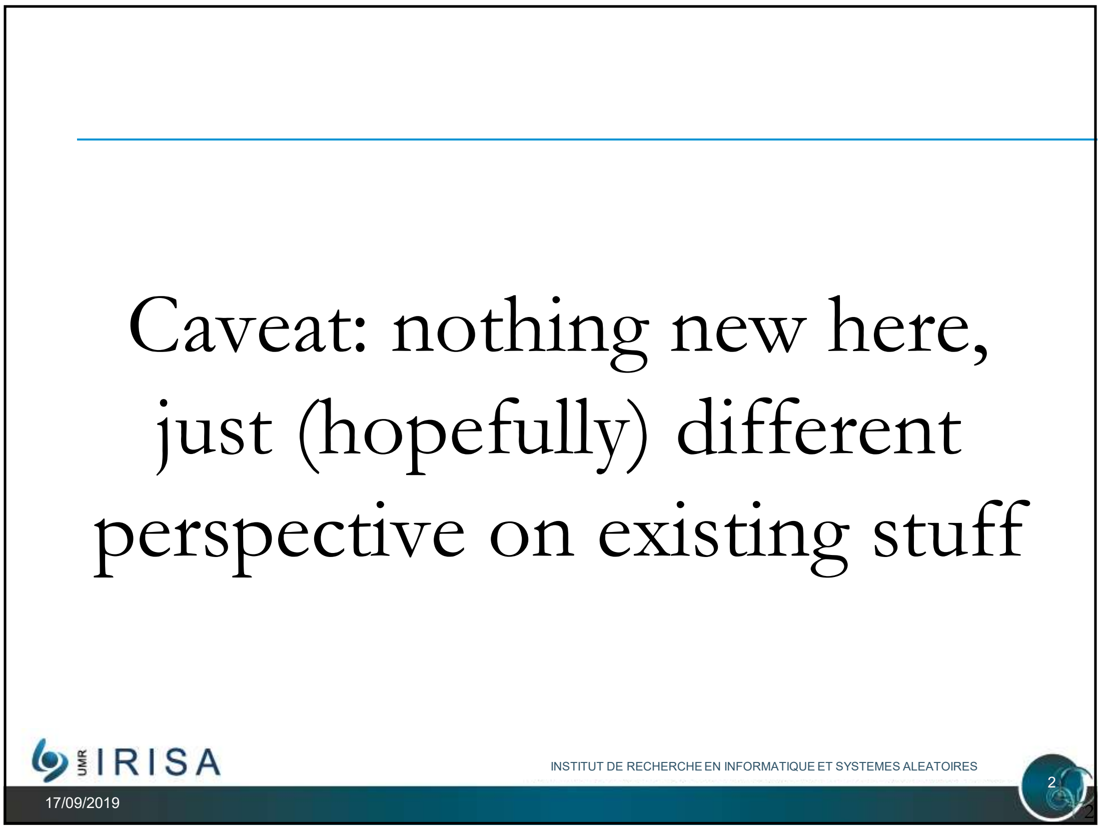
Director of IRISA

jezequel@irisa.fr

http://people.irisa.fr/Jean-Marc.Jezequel

@jmjezequel

cnrs ENS rennes Inria INSA RENNES CentraleSupélec TELECOM Bretagne UBS Université de Bretagne-Sud UNIVERSITÉ DE RENNES 1



Caveat: nothing new here, just (hopefully) different perspective on existing stuff

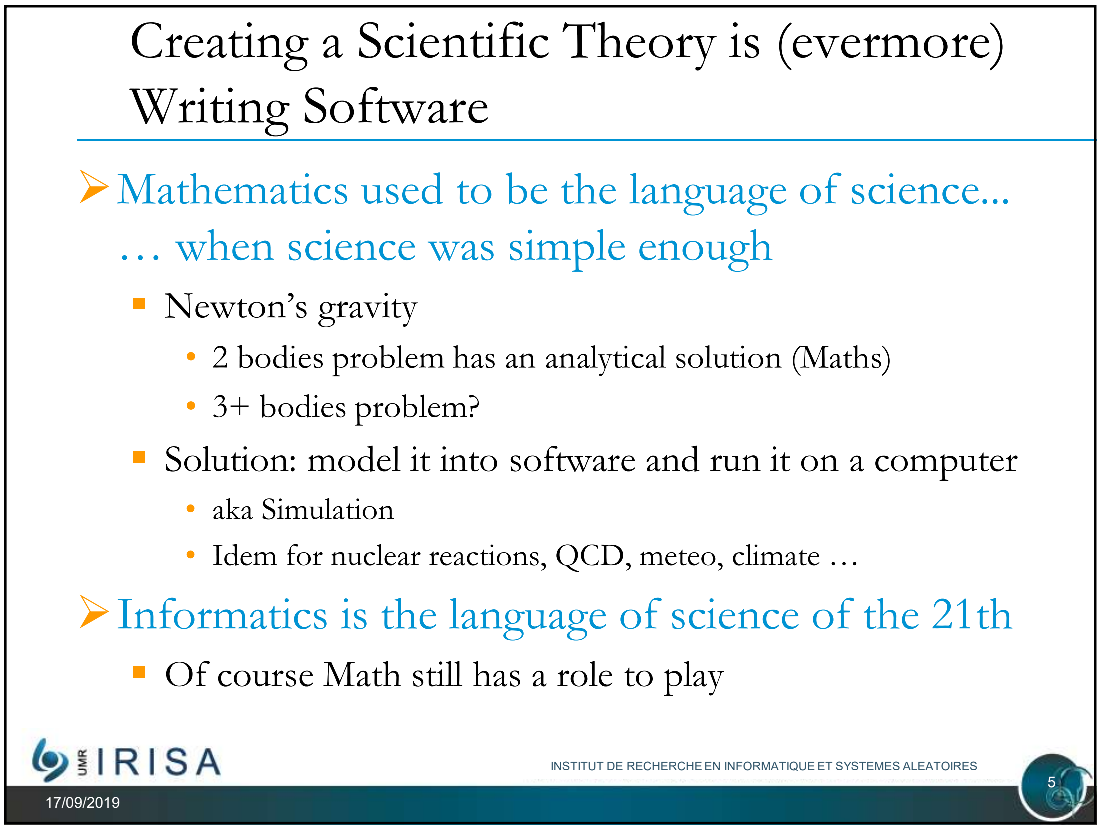
UMR IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

17/09/2019 2

Writing Software Models *vs.* Creating a Scientific Theory

Scientific Theories & Models

- A scientific theory is an explanation of an aspect of the natural world that can be repeatedly tested, in accordance with the scientific method, using a predefined protocol of observation and experiment
    - "The Structure of Scientific Theories" in The Stanford Encyclopedia of Philosophy
- The scientific method involves
    - the proposal and testing of hypotheses,
        - by deriving predictions from the hypotheses about the results of future experiments
    - then performing those experiments to see whether the predictions are valid
- A Model is an abstraction of an aspect of the world for a specific purpose. Therefore a Scientific Theory is a Model.
    - But a Model is not always a Scientific Theory

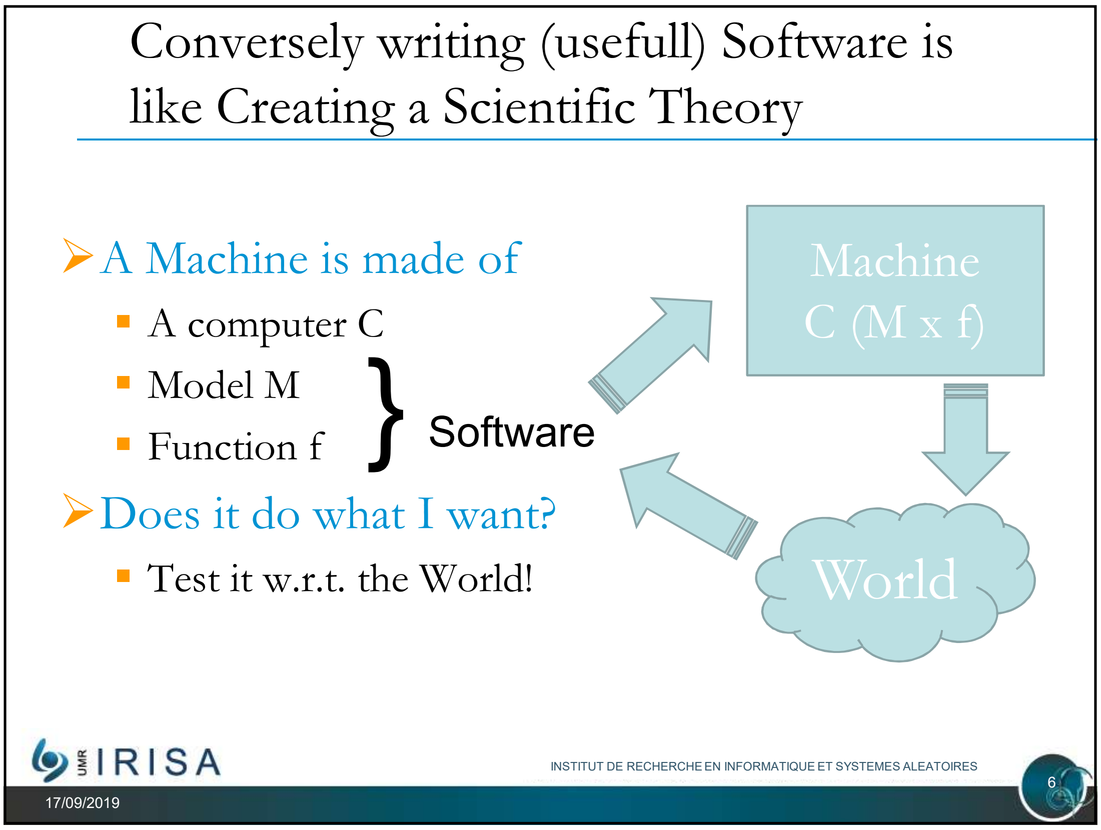


Creating a Scientific Theory is (evermore) Writing Software

- Mathematics used to be the language of science… … when science was simple enough
  - Newton's gravity
    - 2 bodies problem has an analytical solution (Maths)
    - 3+ bodies problem?
  - Solution: model it into software and run it on a computer
    - aka Simulation
    - Idem for nuclear reactions, QCD, meteo, climate …
- Informatics is the language of science of the 21th
  - Of course Math still has a role to play

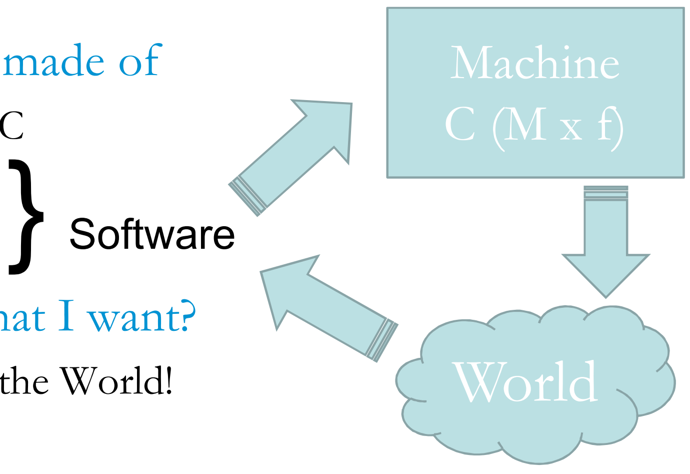
IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

17/09/2019 5



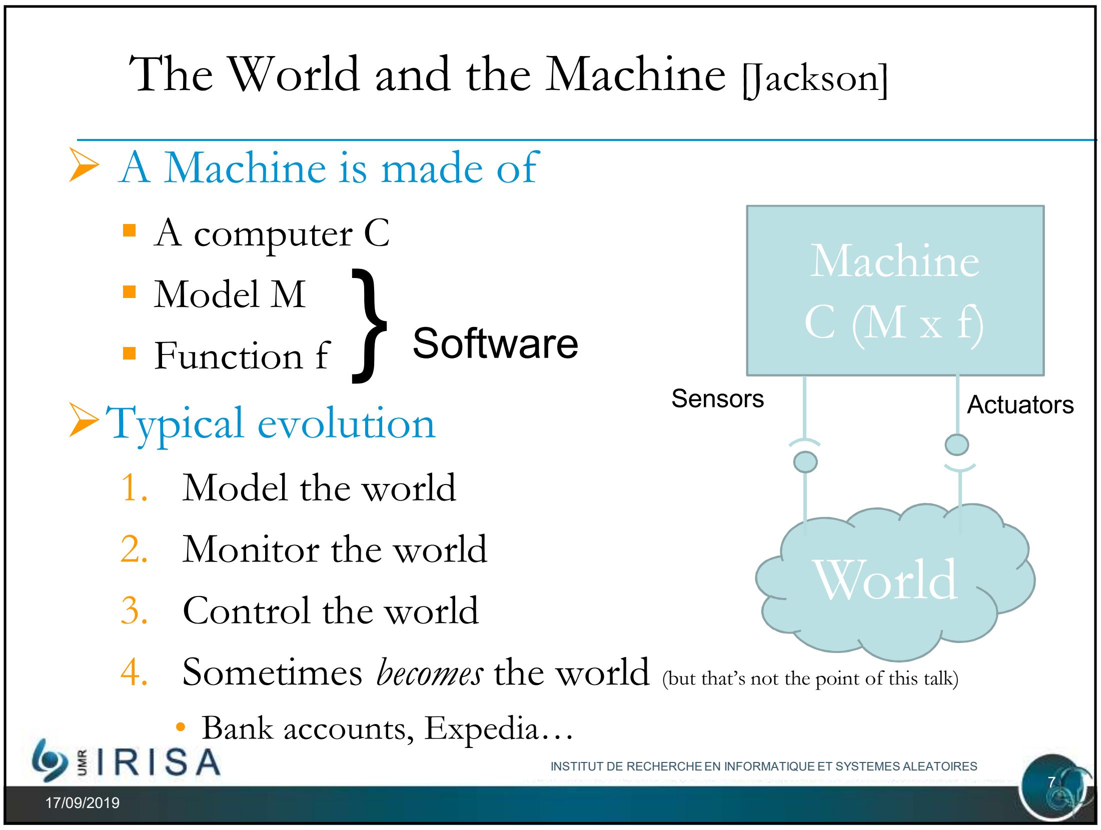
Conversely writing (usefull) Software is like Creating a Scientific Theory

- A Machine is made of
  - A computer C
  - Model M
  - Function f
- Does it do what I want?
  - Test it w.r.t. the World!



IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

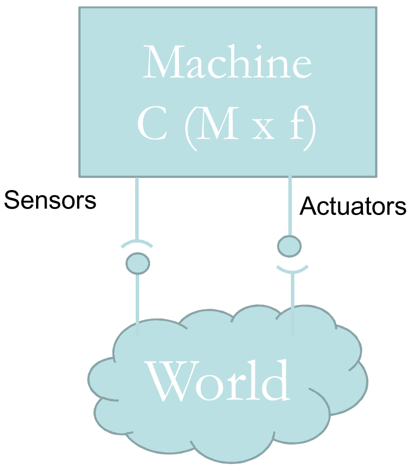
17/09/2019 6

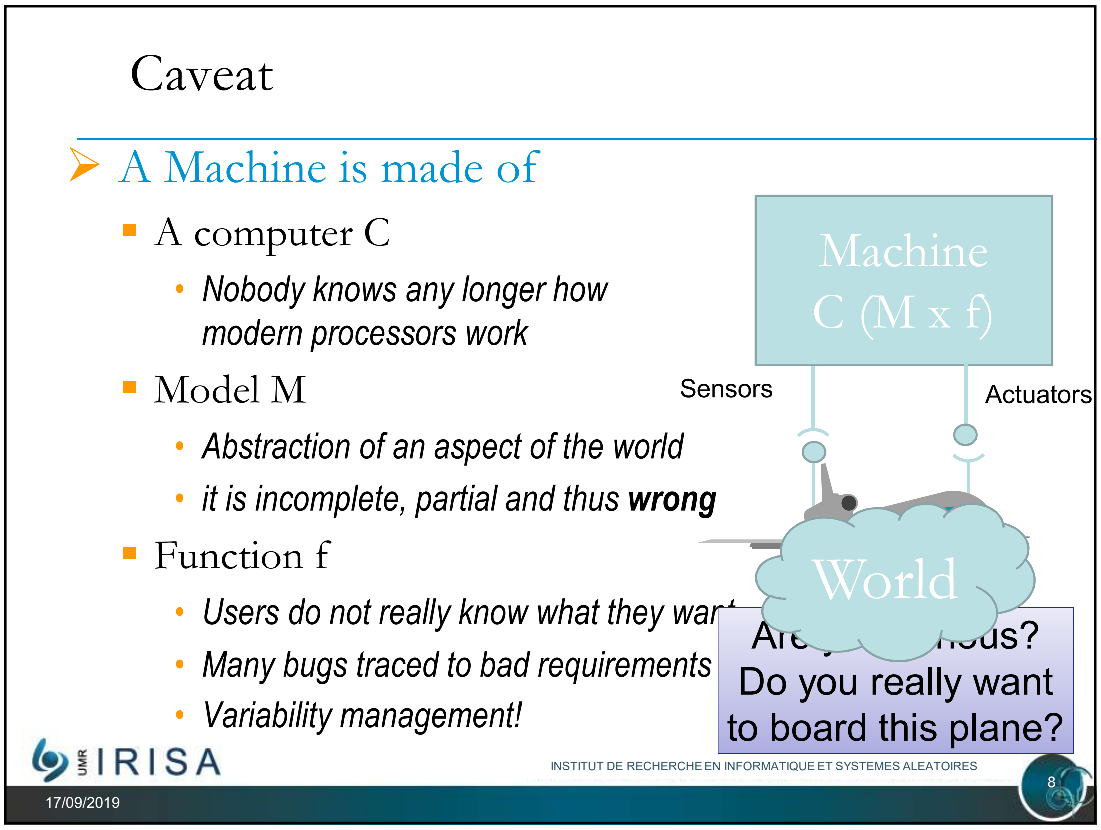


The World and the Machine [Jackson]

- A Machine is made of
  - A computer C
  - Model M
  - Function f

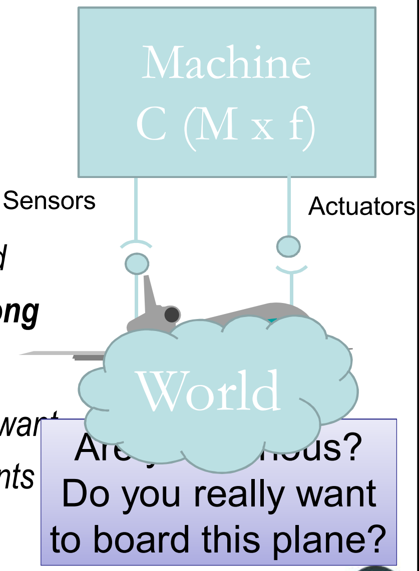
  } Software
- Typical evolution
  1. Model the world
  2. Monitor the world
  3. Control the world
  4. Sometimes *becomes* the world (but that's not the point of this talk)
     - Bank accounts, Expedia…

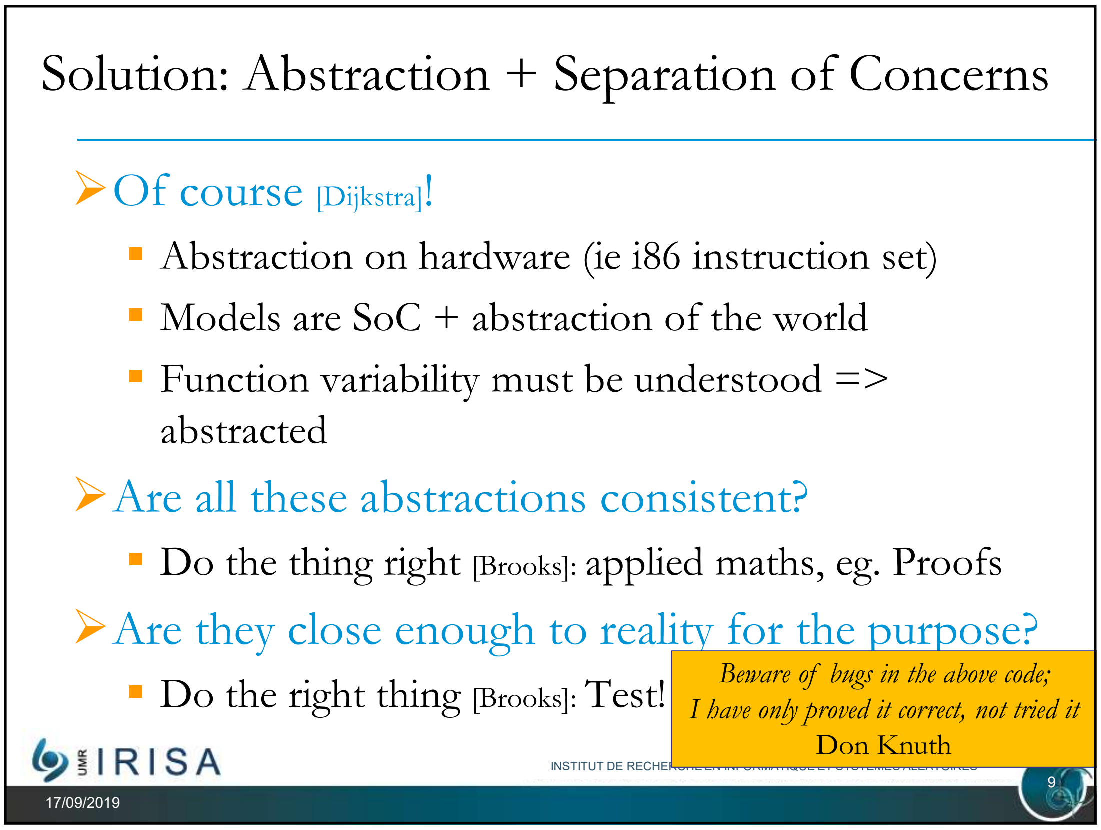




Caveat

- A Machine is made of
  - A computer C
    - *Nobody knows any longer how modern processors work*
  - Model M
    - *Abstraction of an aspect of the world*
    - *it is incomplete, partial and thus **wrong***
  - Function f
    - *Users do not really know what they want*
    - *Many bugs traced to bad requirements*
    - *Variability management!*





Solution: Abstraction + Separation of Concerns

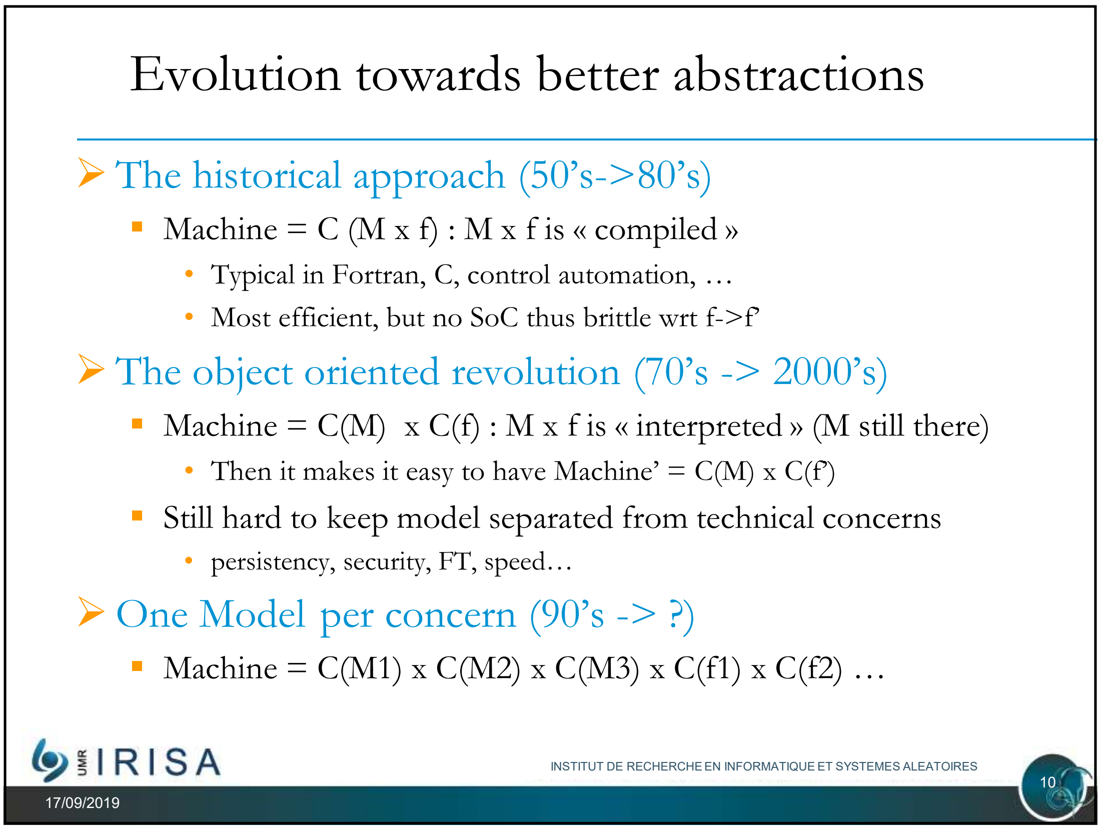
- Of course [Dijkstra]!
  - Abstraction on hardware (ie i86 instruction set)
  - Models are SoC + abstraction of the world
  - Function variability must be understood => abstracted
- Are all these abstractions consistent?
  - Do the thing right [Brooks]: applied maths, eg. Proofs
- Are they close enough to reality for the purpose?
  - Do the right thing [Brooks]: Test!

Beware of bugs in the above code;
I have only proved it correct, not tried it
Don Knuth

IRISA

INSTITUT DE RECHE

17/09/2019 9



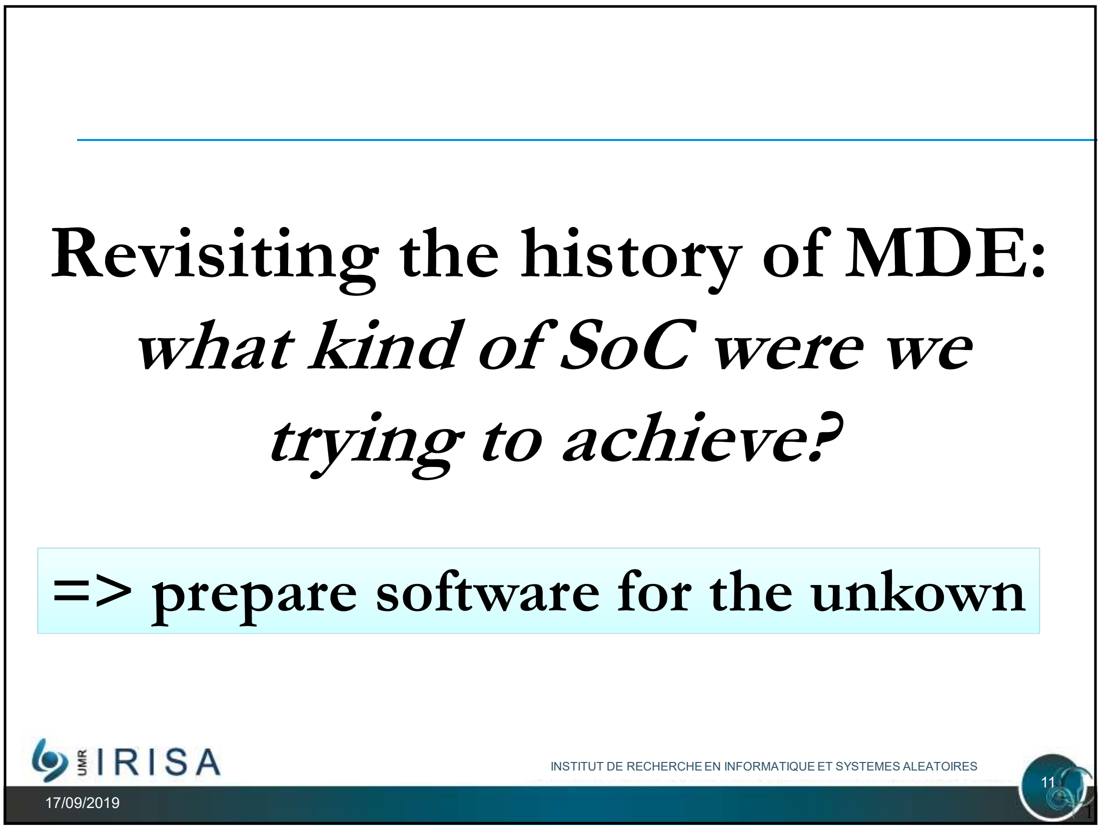
Evolution towards better abstractions

- The historical approach (50's->80's)
  - Machine = C (M x f) : M x f is « compiled »
    - Typical in Fortran, C, control automation, …
    - Most efficient, but no SoC thus brittle wrt f->f'
- The object oriented revolution (70's -> 2000's)
  - Machine = C(M) x C(f) : M x f is « interpreted » (M still there)
    - Then it makes it easy to have Machine' = C(M) x C(f')
  - Still hard to keep model separated from technical concerns
    - persistency, security, FT, speed…
- One Model per concern (90's -> ?)
  - Machine = C(M1) x C(M2) x C(M3) x C(f1) x C(f2) …

IRISA

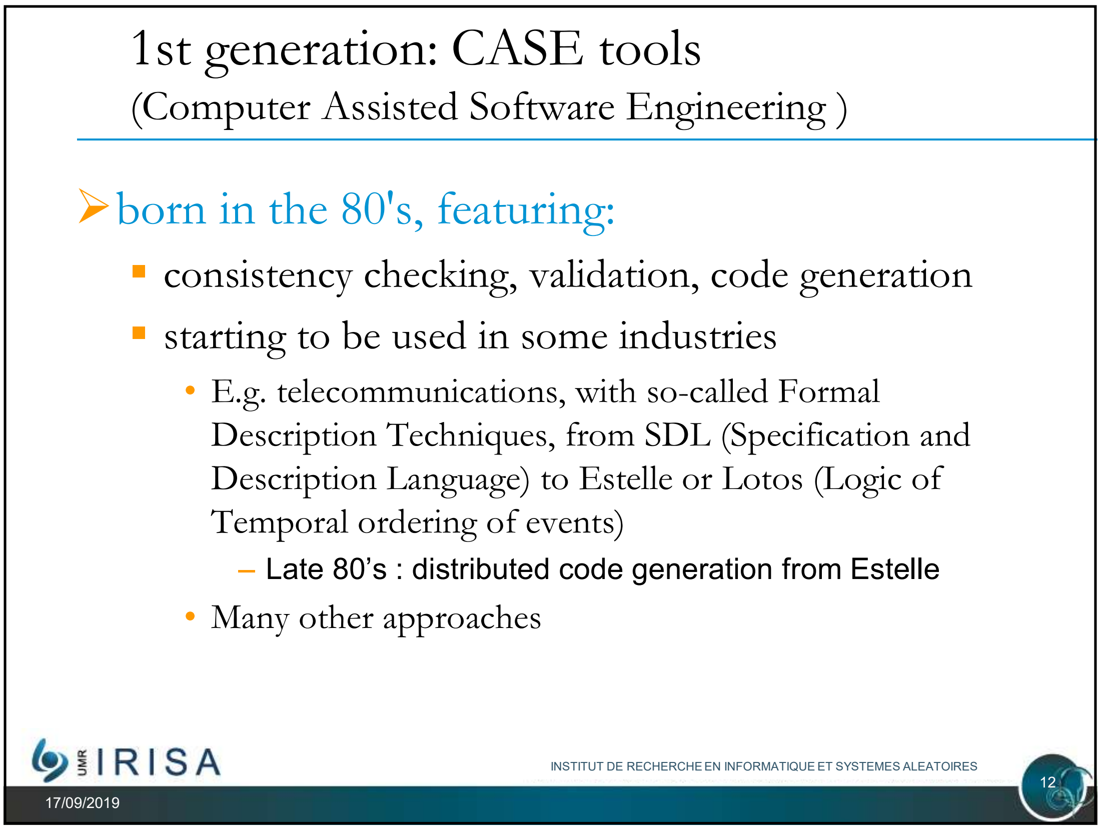
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

17/09/2019 10



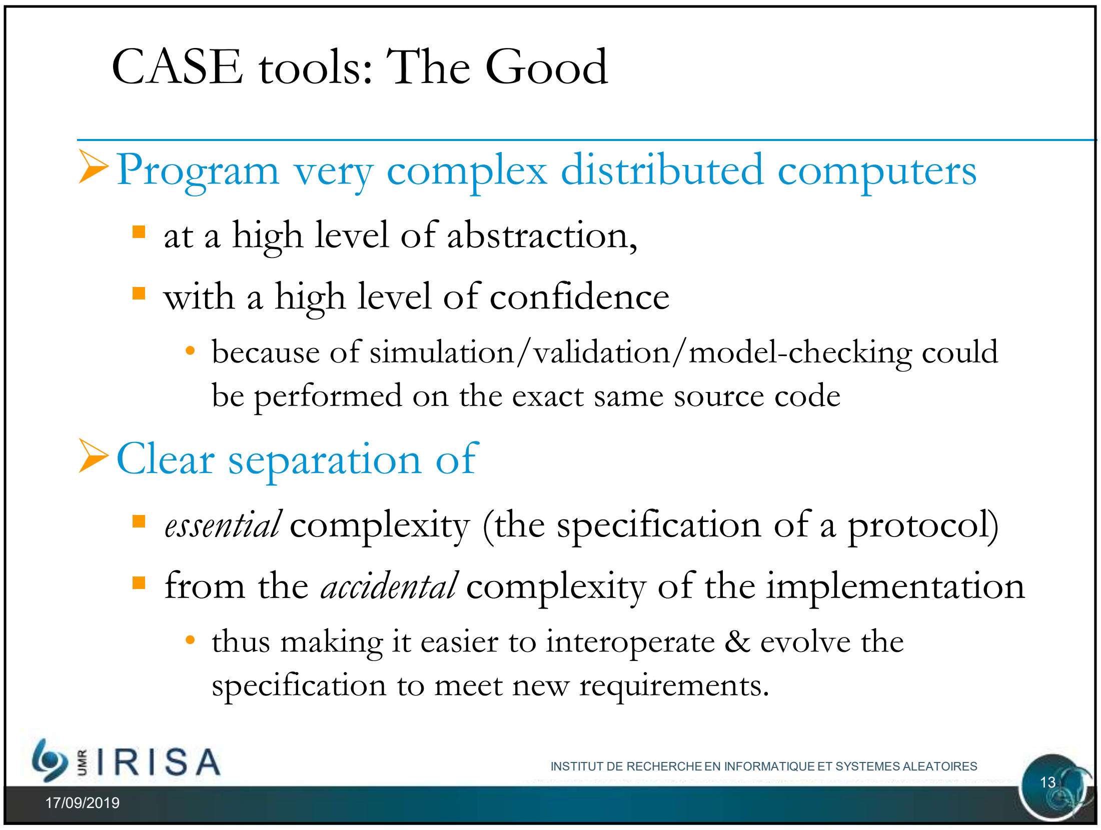
Revisiting the history of MDE: *what kind of SoC were we trying to achieve?*

=> prepare software for the unkown



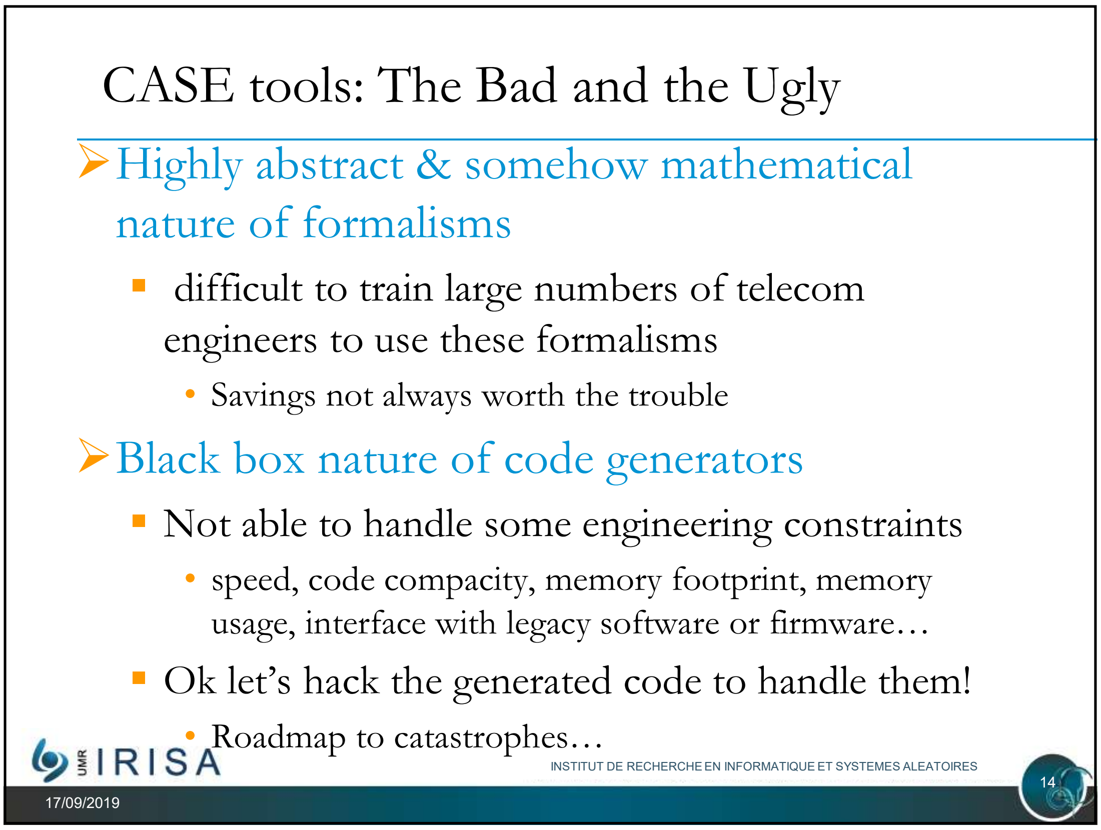
1st generation: CASE tools (Computer Assisted Software Engineering)

- born in the 80's, featuring:
  - consistency checking, validation, code generation
  - starting to be used in some industries
    - E.g. telecommunications, with so-called Formal Description Techniques, from SDL (Specification and Description Language) to Estelle or Lotos (Logic of Temporal ordering of events)
      - Late 80’s : distributed code generation from Estelle
    - Many other approaches



CASE tools: The Good

- Program very complex distributed computers
  - at a high level of abstraction,
  - with a high level of confidence
    - because of simulation/validation/model-checking could be performed on the exact same source code
- Clear separation of
  - *essential* complexity (the specification of a protocol)
  - from the *accidental* complexity of the implementation
    - thus making it easier to interoperate & evolve the specification to meet new requirements.

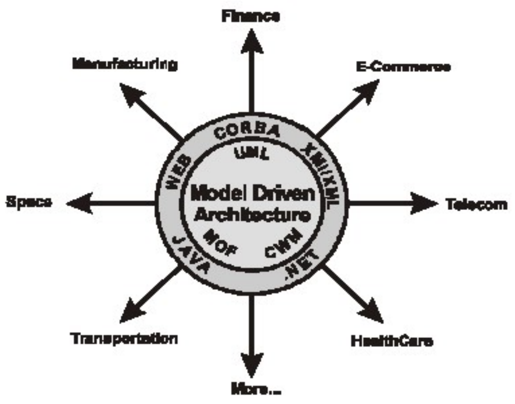


CASE tools: The Bad and the Ugly

- Highly abstract & somehow mathematical nature of formalisms
  - difficult to train large numbers of telecom engineers to use these formalisms
    - Savings not always worth the trouble
- Black box nature of code generators
  - Not able to handle some engineering constraints
    - speed, code compacity, memory footprint, memory usage, interface with legacy software or firmware…
  - Ok let's hack the generated code to handle them!
    - Roadmap to catastrophes…

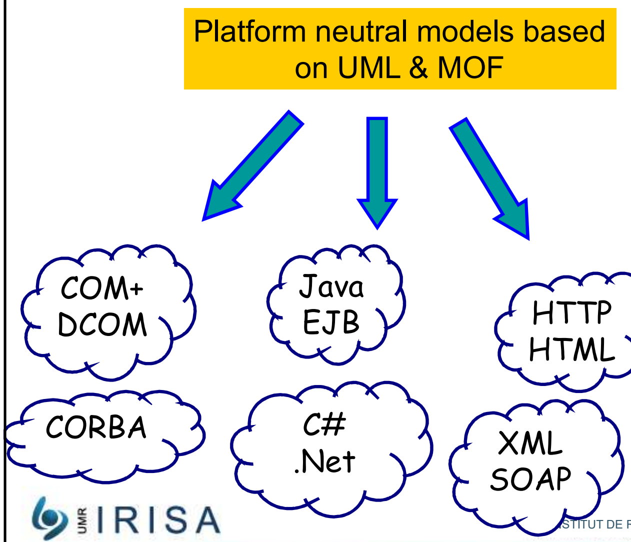
2nd generation: MDA

"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

MDA: The Good

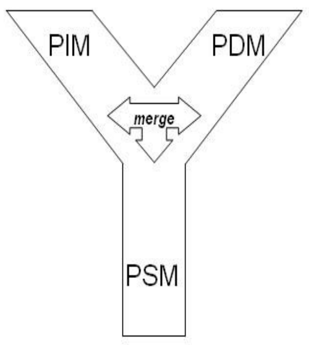


⌘Organization assets expressed as models, clear separation from platforms

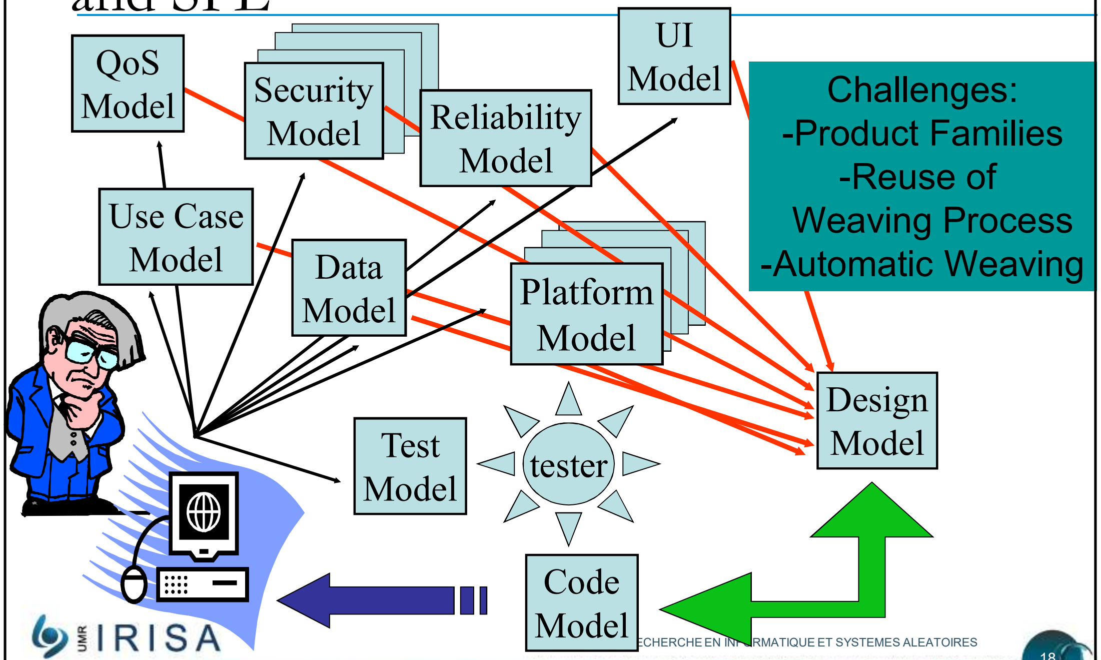
⌘Model transformations to map to technology specific platforms (QVT)

MDA: The Bad and the Ugly

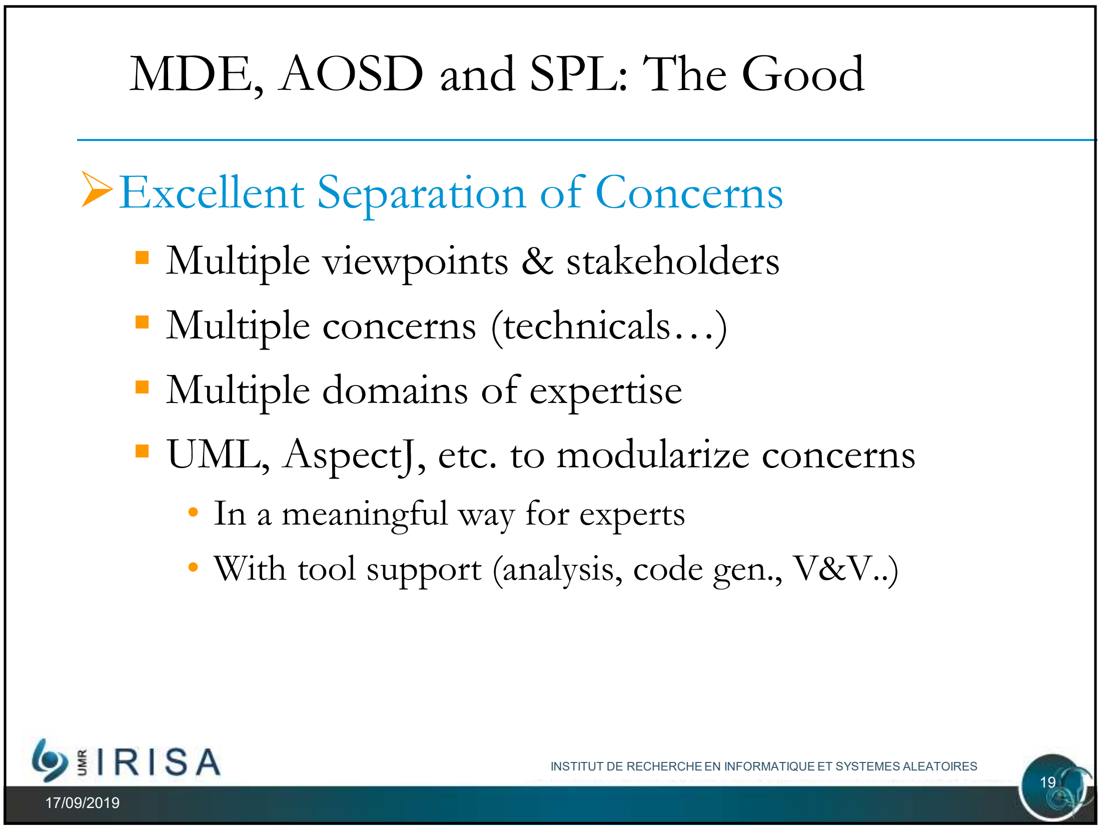
- MDA models
  - **PIM**: Platform Independent Model : an Oxymore
    - Business Model of a system abstracting away the deployment details of a system
  - **PSM**: Platform Specific Model
  - **PDM**: Platform Description Model
    - – Nobody had ever actually seen them
      - » Utterly naïve Y-shaped approach
    - – So the platform know-how is encoded into the transformation
      - » Hard to write, evolve, maintain…



3rd generation: SoC with MDE, AOSD and SPL



Challenges:
-Product Families
-Reuse of Weaving Process
-Automatic Weaving

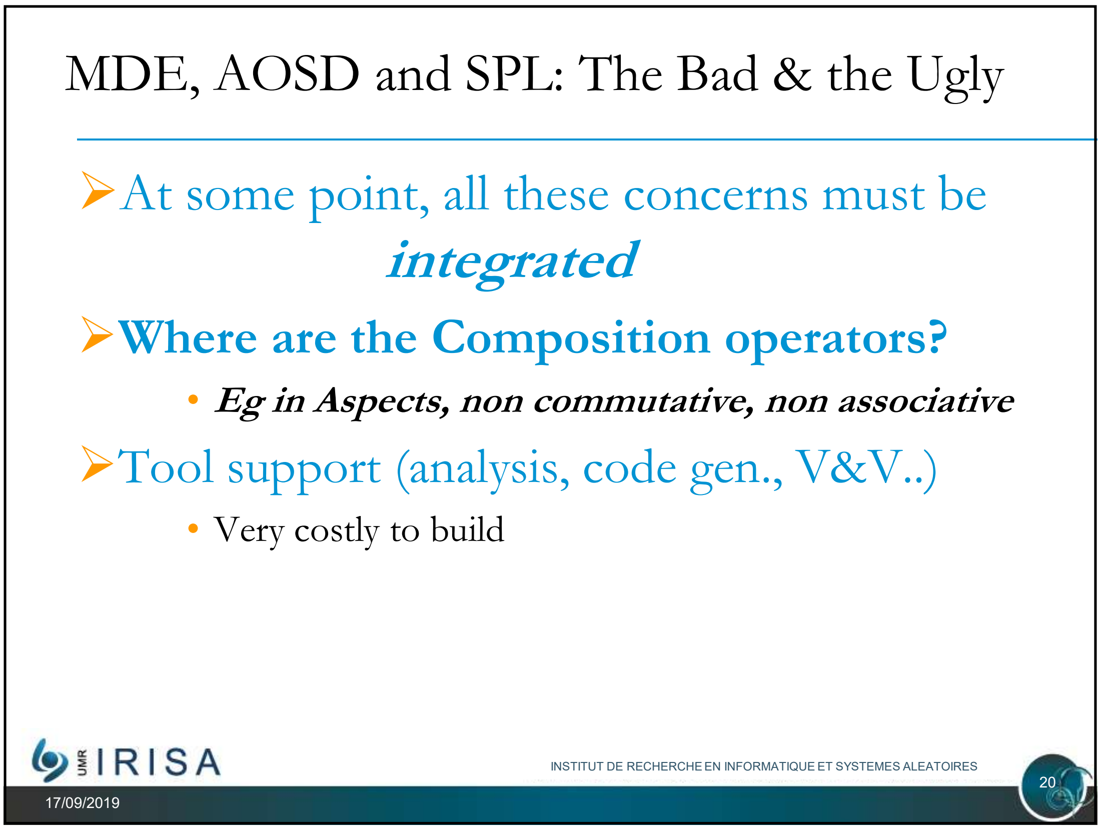


MDE, AOSD and SPL: The Good

- Excellent Separation of Concerns
  - Multiple viewpoints & stakeholders
  - Multiple concerns (technicals…)
  - Multiple domains of expertise
  - UML, AspectJ, etc. to modularize concerns
    - In a meaningful way for experts
    - With tool support (analysis, code gen., V&V..)

IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

17/09/2019 19



MDE, AOSD and SPL: The Bad & the Ugly

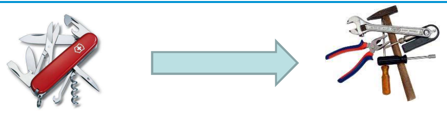
- At some point, all these concerns must be ***integrated***
- **Where are the Composition operators?**
  - ***Eg in Aspects, non commutative, non associative***
- Tool support (analysis, code gen., V&V..)
  - Very costly to build

IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

17/09/2019 20

4th generation: DSL & SLE
Domain Specific Languages, Software Language Engineering

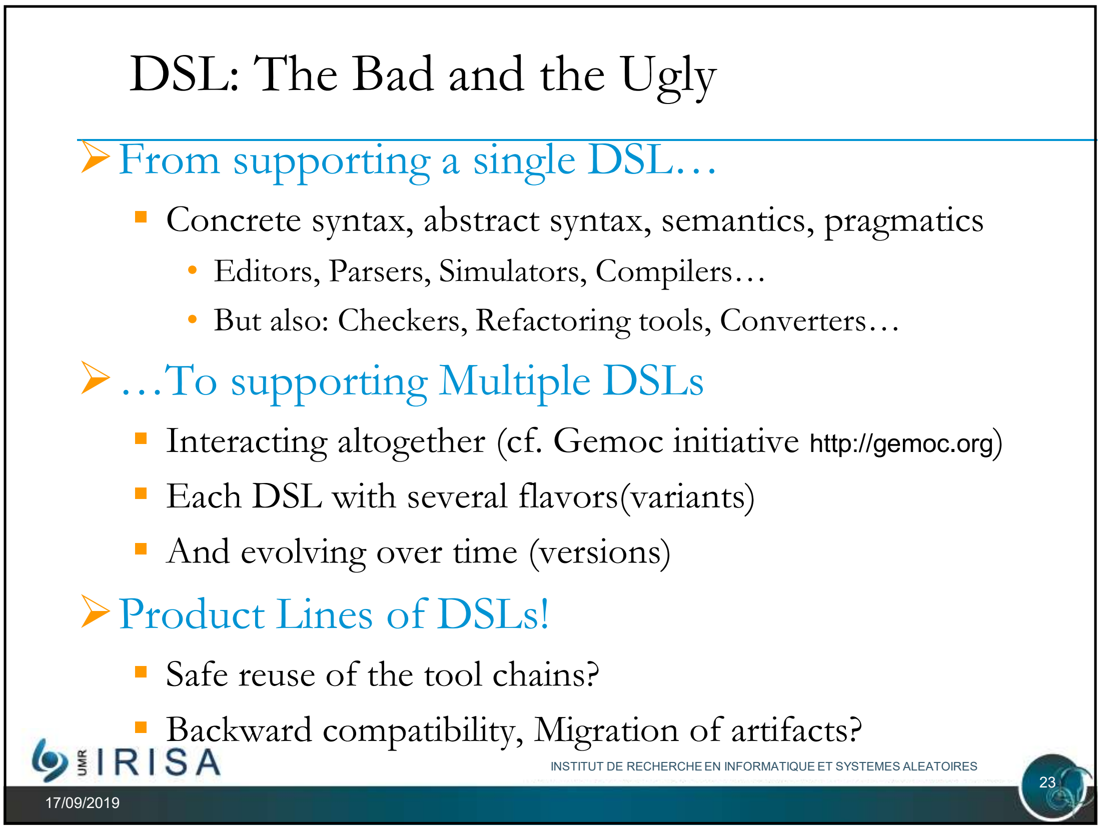
- DSL



  - Targeted to a particular kind of problem
    - Long history of DSL, with dedicated notations (textual or graphical), support (editor, checkers, etc.)
  - Promises: more « efficient » languages for resolving a set of specific problems in a domain
  - Each concern described in its own language => reduce abstraction gap
- SLE:
  - Raise composition issues at Language Level

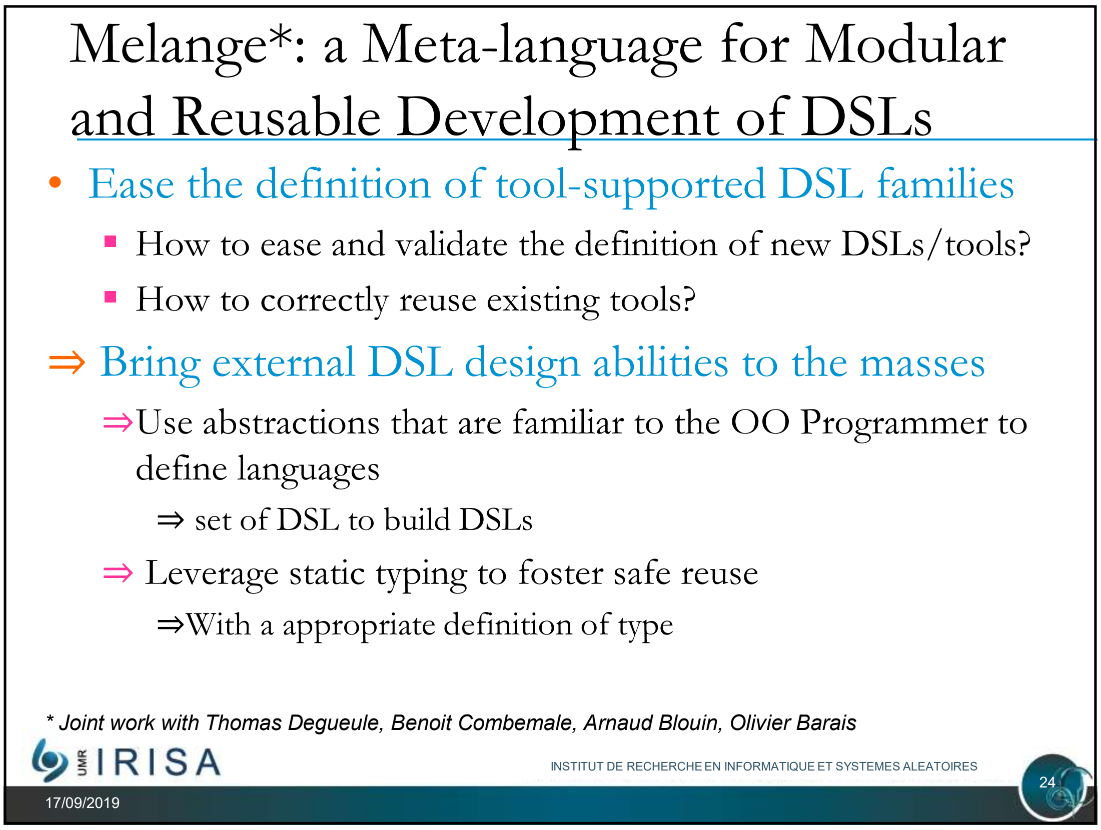
DSL: The Good

- A DSL program is a 3D abstraction
  - from the domain (cf Newton vs Relativity),
  - from the function (requirements)
  - from the platform
- Embrace uncertainty
  - Smaller abstractions than with GPL
    - We better know and control the unknown
  - Apply rigorous methods to uncertain systems so that we get known uncertainties



DSL: The Bad and the Ugly

- From supporting a single DSL…
  - Concrete syntax, abstract syntax, semantics, pragmatics
    - Editors, Parsers, Simulators, Compilers…
    - But also: Checkers, Refactoring tools, Converters…
- …To supporting Multiple DSLs
  - Interacting altogether (cf. Gemoc initiative http://gemoc.org)
  - Each DSL with several flavors(variants)
  - And evolving over time (versions)
- Product Lines of DSLs!
  - Safe reuse of the tool chains?
  - Backward compatibility, Migration of artifacts?



Melange*: a Meta-language for Modular and Reusable Development of DSLs

- Ease the definition of tool-supported DSL families
  - How to ease and validate the definition of new DSLs/tools?
  - How to correctly reuse existing tools?

⇒ Bring external DSL design abilities to the masses

⇒Use abstractions that are familiar to the OO Programmer to define languages

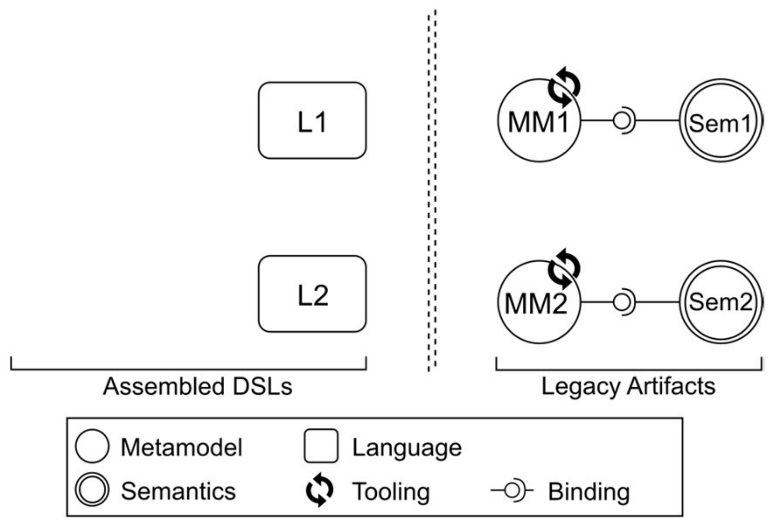
⇒ set of DSL to build DSLs

⇒ Leverage static typing to foster safe reuse

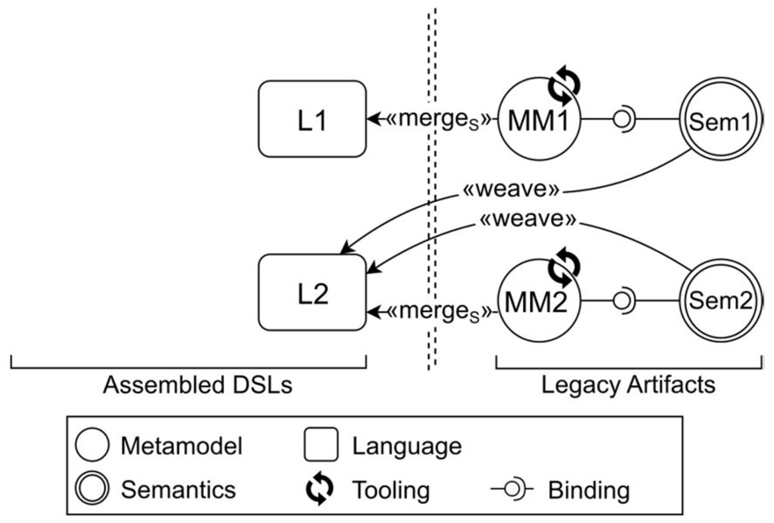
⇒With a appropriate definition of type

** Joint work with Thomas Degueule, Benoit Combemale, Arnaud Blouin, Olivier Barais*

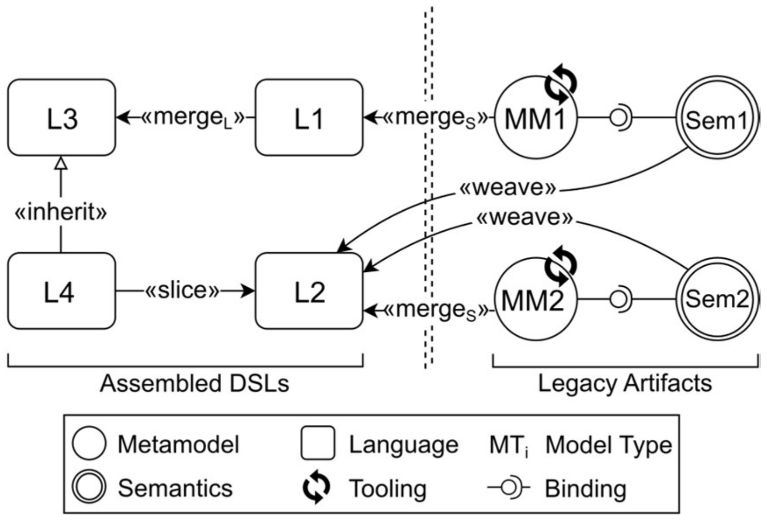
Approach Overview



Approach Overview



Approach Overview



Inspired by *eg*. Erdweg *et al.*, *Language Composition Untangled*, LDTA, 2012

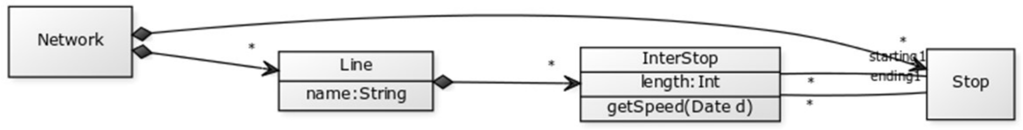
MELANGE

- An open-source (EPL) language workbench
- or… a language-based, model-oriented language for DSL engineering
- An implementation of the algebra
- Supported by a model-oriented type sytem
- Based on Xtext
- Seamlessly integrated with the EMF ecosystem
- Bundled as a set of Eclipse plug-ins

5th generation: Models & Data

➢ The model is no longer fully known a priori

- Example of bus network



  - getSpeed(Date pastDate)
    - Engineering Model: access to DB to yield result
  - getSpeed(Date now)
    - Model@runtime: access to sensors to yield result
  - getSpeed(Date futureDate)
    - How to predict the future?

UMR IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

17/09/2019 29

Predicting the future: getSpeed(Date futureDate)

➢ Model with some mathematical approximation

- Aka Scientific Model
- Calibrate it with data

➢ Learn it from data

- Inductive reasoning principle, i.e., generalization from specific cases.
  - implies some uncertainty: do specific cases sufficiently represent the rules and principles?
- Continue to learn over ongoing collected data
  - Still need models at runtime!

UMR IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

17/09/2019 30

Models & Data: Challenges

- How to cleanly separate and compose
  - Engineering Models
  - Scientific models
  - Models obtained through Machine Learning
- Continuous update
  - Online training
    - Confidence level in the prediction
  - Link with models@runtime

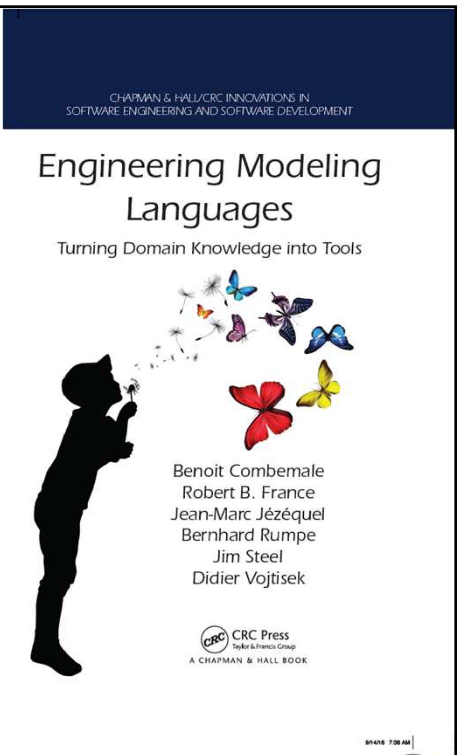
Acknowledgement

- All these ideas have been developed with my colleagues of the DiverSE team at IRISA/Inria

DiverSE Diversity-Centric Software Engineering

Formely known as Triskell

@jmjezequel


CHAPMAN & HALL/CRC INNOVATIONS IN SOFTWARE ENGINEERING AND SOFTWARE DEVELOPMENT

Engineering Modeling Languages

Turning Domain Knowledge into Tools

Benoit Combemale
Robert B. France
Jean-Marc Jézéquel
Bernhard Rumpe
Jim Steel
Didier Vojtisek

CRC Press Taylor & Francis Group
A CHAPMAN & HALL BOOK