UMR IRISA
Institut de Recherche en Informatique et Systèmes Aléatoires

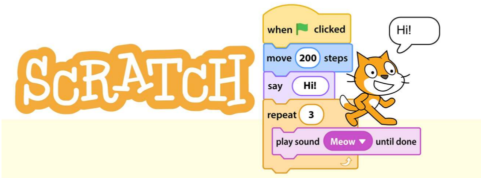
A Tale of Taming Variability with MDE

Jean-Marc Jézéquel
jezequel@irisa.fr
@jmjezequel

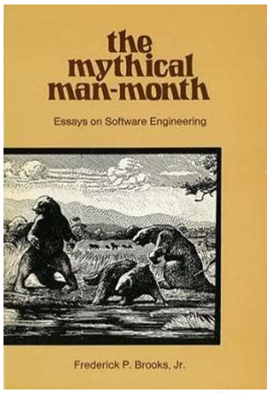
UNIVERSITÉ DE RENNES 1

DiverSE
Diversity-Centric Software Engineering

The software paradoxe



- So easy to write simple programs that a 6 years old child can do Logo or Scratch programming right after a few minutes of training.
- On the other hand, it is so difficult to write complex ones that basically nobody is able to write large, bug free programs
  - writing a 100,000 line program is much more difficult than 1000 times the effort of writing a 100 line program



Dimensions of complexity in building software

- Inherent complexity due to indecidability
- Complexity due to size of the problem
  - Essential vs. Accidental
- Complexity due to variability/uncertainty
  - Variability with requirements
    - including business or legal rules and human expected behavior
      - generally *incomplete* and do *evolve* over time
  - Uncertainty in assumptions about the world
    - quite rough, often implicit, and do not take into account all corner cases
      - both inherent (platform) and accidental (misunderstanding of API)
    - Not counting cybersecurity

Reasoning (proof, dbc, tests)

Modularity/Abstraction

OO Modeling
Model Driven Engineering

Separation of concerns, Variability management

IRISA

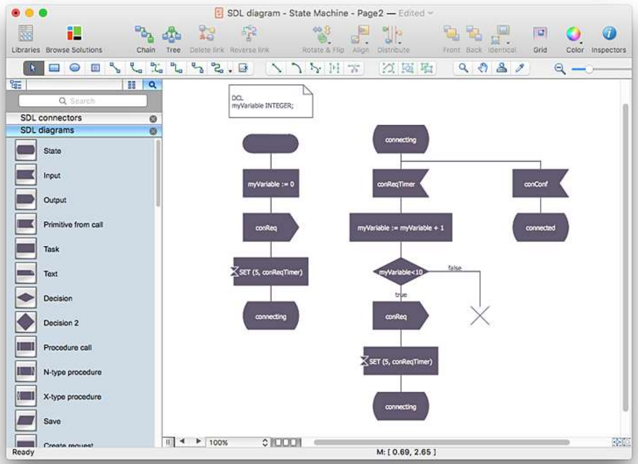
The 5 Ages of Model Driven Engineering

1. CASE tools
2. Model Driven Architecture
3. Separation of Concerns with Models, Aspects and Features
4. Domain Specific Languages & Software Language Engineering
5. Handling Data-Centric Socio-Technical Systems

IRISA

Mid 80's-Mid 90's

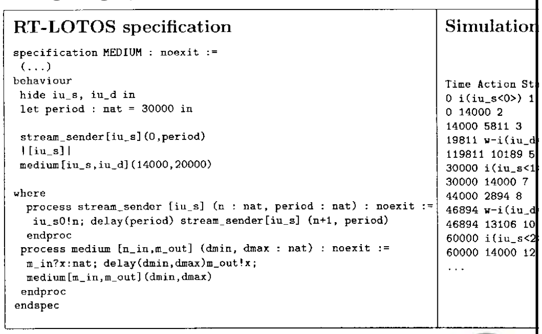
1. CASE tools



Address variability in the specification
=> A small change in the specification should not be that hard to validate/implement

Computer Assisted Software Engineering CASE Tools

- Formal Description Techniques used in e.g. the Telecom industry since the 80's
  - Estelle or SDL (Specification and Description Language)
    - based on extended state machines
  - Lotos (Logic of Temporal ordering of events)
    - based on process algebra
- Features:
  - Graphic/textual editors
  - consistency checking
  - Validation (simulation, model-checking)
  - code generation



RT-LOTOS specification

```
specification MEDIUM : noexit :=
 (...)
behaviour
 hide iu_s, iu_d in
 let period : nat = 30000 in

 stream_sender[iu_s](0,period)
 |[iu_s]|
 medium[iu_s,iu_d](14000,20000)

where
  process stream_sender [iu_s] (n : nat, period : nat) : noexit :=
   iu_s0!n; delay(period) stream_sender[iu_s] (n+1, period)
  endproc
 process medium [m_in,m_out] (dmin, dmax : nat) : noexit :=
  m_in?x:nat; delay(dmin,dmax)m_out!x;
  medium[m_in,m_out](dmin,dmax)
 endproc
endspec
```

Simulation

```
Time Action St
0 i(iu_s<0>) 1
0 14000 2
14000 5811 3
19811 w-i(iu_d
119811 10189 5
30000 i(iu_s<1
30000 14000 7
44000 2894 8
46894 w-i(iu_d
46894 13106 10
60000 i(iu_s<2
60000 14000 12
...
```

CASE Tools

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

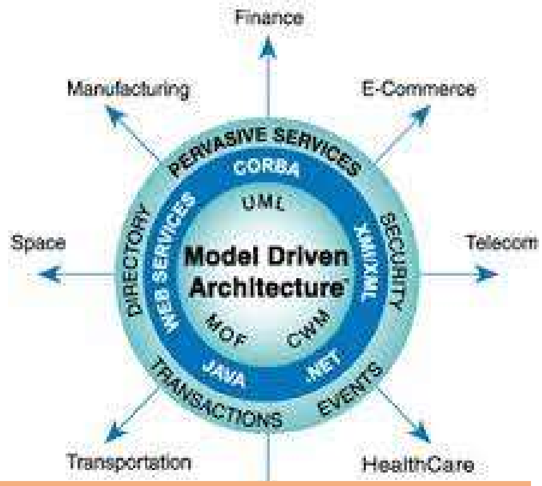
CASE Tools

- Highly abstract and somehow mathematical nature of FDT
  - difficult to train large numbers of telecom engineers
- Code generators at that time were black boxes
  - Sometimes perfectly fitted the engineering needs => published success stories
  - Most often missed at least one engineering constraint
    - speed, code compacity, memory footprint, memory usage, interface with legacy software or firmware
  - Then we are stuck!
    - Workarounds (risky w.r.t. FDT semantics)
- Bottom line: not worth the trouble

UMR IRISA
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALÉATOIRES

Mid 90's-Mid 2000's

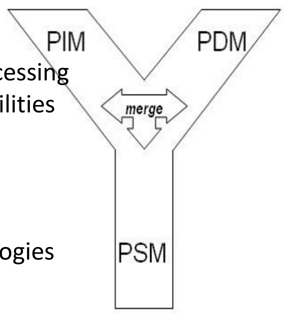
2. Model Driven Architecture



Explicitly address the variability of the platform

MDA

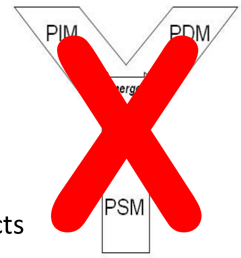
- Separate the fundamental logic behind a specification from the specifics of the particular middleware that implements it
- Main concepts
  - **CIM** a Computation Independent Model focuses on the context and requirements of the system without consideration for its structure or processing
  - **PIM** a The Platform Independent Model focuses on the operational capabilities of a system outside the context of a specific platform
    - showing only those parts that can be abstracted out of that platform.
  - **PSM** a Platform Specific Model augments a PIM with details
    - relating to the use of a specific platform.
  - **PDM** a Platform Description Model describes set of subsystems & technologies that provide a coherent set of functionalities
    - e.g.; CORBA, Java/EJB, C\#/.NET etc.
  - **Model Transformations** are automated ways of modifying and creating models.



MDA

- Portability
  - increasing application re-use and reducing the cost and complexity of application development and management, now and into the future.
- Cross-platform Interoperability
  - using rigorous methods to guarantee that standards based on multiple implementation technologies all implement identical business functions.
- Platform Independence
  - greatly reducing the time, cost and complexity associated with re-targeting applications for different platforms ---including those yet to be introduced.
- Domain Specificity
  - through Domain-specific models that enable rapid implementation of new, industry-specific applications over diverse platforms.
- Productivity
  - by allowing developers, designers and system administrators to use languages and concepts they are comfortable with, while allowing seamless communication and integration across the teams.

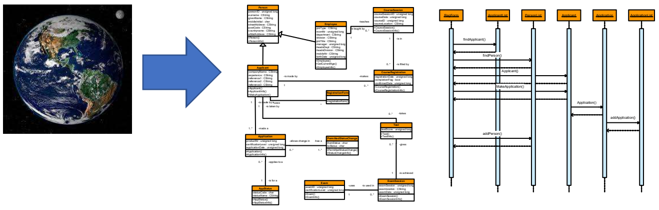
MDA

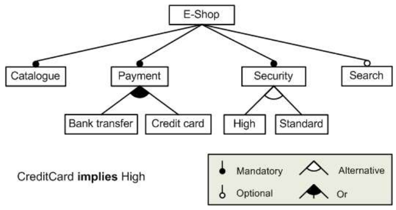


- Mostly a forward engineering approach
  - models are transformed into implementation artifacts
  - But PDM do not really exists!
- Not everything captured in the source models =>
  - some modification of the generated code has to be carried out manually
    - to take into account the missing concerns.
  - nightmare from the maintenance point of view
    - even with tricks to alleviate the burden of keeping them in synch
- But **know-how** not just in PIM but *also* in the design process
  - Transformations can get much more complex than the PIM!
    - But at the wrong level of abstraction
      - QVT/ATL meant for relatively simple MT, cannot compete with modern java Kotlin
    - Hard to use at scale: Model Transformation Languages are a dead end

Early 2000's-Mid 2010's

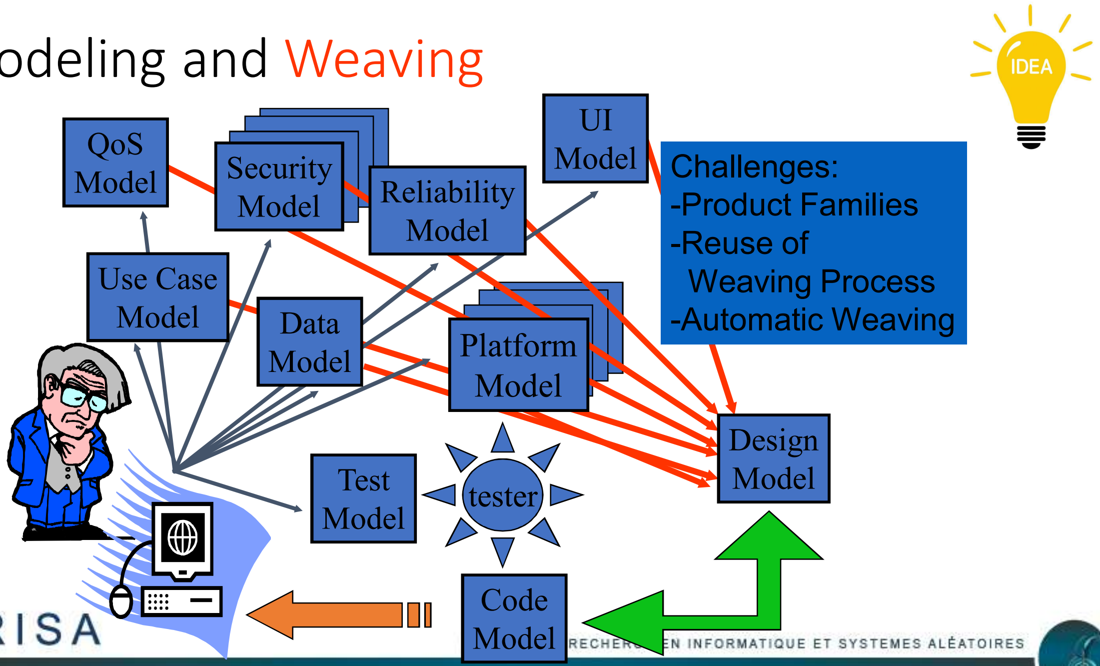
3. Separation of Concerns with Models, Aspects and Features



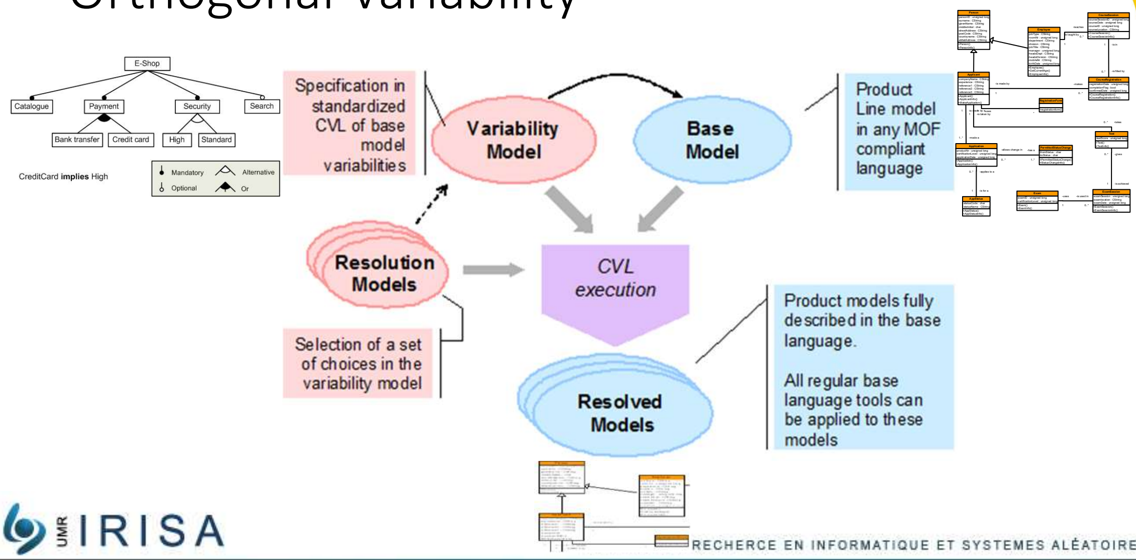


A model is the abstraction of an aspect of reality for handling a given concern
=> Multi-dimentional, explicit management of variability (not just the platform)

Modeling and Weaving



Orthogonal Variability



SoC with Models, Aspect & Features

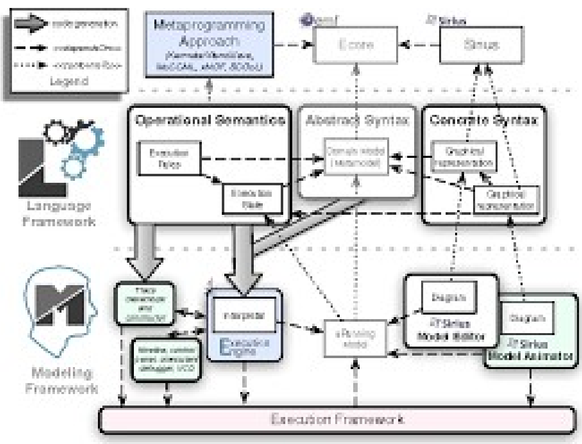
- Modeling is the activity of separating concerns into aspects
- Design is weaving of these aspects into a detailed design model
  - Using eg Design Patterns
- MDE is about *mechanizing* this design process
  - Making it possible to change one's mind on which version of which variant of any particular aspect she wants in the system. And she wants to do it cheaply, quickly and safely.
  - when a new product has to be derived from the product-line, we can automatically replay the design process, just changing a few things here and there

SoC with Models, Aspect & Features

- Cleanly separating concerns of a system is not always completely straightforward
  - But difficulty proportional to the inherent complexity of the problem at hand
- Weaving a single aspect is pretty straightforward, weaving a second one at the same join point is another story
  - Depending on how you define join point matching mechanism, detection can get undecidable and/or advice composition tricky
  - Missing associativity and commutativity for the weaving operator
- No hope for a fully general purpose, meta-model independent, model-level aspect weaver
  - Specific solutions might exist and be valuable though

Early 2010's-?

4. Domain Specific Languages & Software Language Engineering



Provide a **language** to each stakeholder to express variable problems/solutions
=> lift the composition at the language (or meta-model) level

UMR IRISA
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALÉATOIRES

« Another lesson we should have learned from the recent past is that the development of 'richer' or 'more powerful' programming languages was a mistake in the sense that these baroque monstrosities, these conglomerations of idiosyncrasies, are really unmanageable, both mechanically and mentally. »

« I see a great future for very systematic and very modest programming languages »

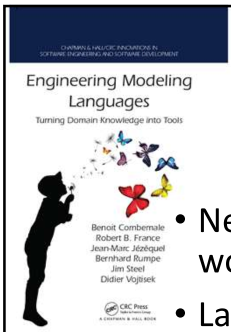
aka **Domain-Specific Languages**

1972

ACM Turing Lecture,
« The Humble Programmer »
Edsger W. Dijkstra

Domain Specific Languages

- Targeted to a **particular** kind of problem
  - 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
- Emergence of the Software Language Engineering (SLE)
  - application of systematic, disciplined, and measurable approaches to the development, use, deployment, and maintenance of software languages



DSL/Software Language Engineering

- New DSLs can nowadays easily be developed using a language workbench
- Languages as first-class entities that can be extended, composed, and manipulated as a whole. Melange (Degueule et al. 2015)
- Co-develop a set of related DSLs
  - *Globalization of modeling languages* (Combemale et al.2014)
  - Allowing eg a system engineer may need to analyze a system property that requires information scattered in models expressed in different DSLs.

IRISA

INSTITUT DE RECHERCE EN INFORMATIQUE ET SYSTEMES ALÉATOIRES

DSL/Software Language Engineering

- GEMOC Studio http://gemoc.org/studio.html
  - metaprogramming approaches and associated execution engines to design and execute the behavioral semantics of executable modeling languages,
  - efficient and domain-specific execution trace management services,
  - model animation services,
  - advanced debugging facilities such as forward and backward debugging (i.e. omniscient debugging), timeline, etc.
  - coordination facilities to support concurrent and coordinated execution of heterogeneous models
  - an extensible framework for easily adding new execution engines & runtime services

IRISA

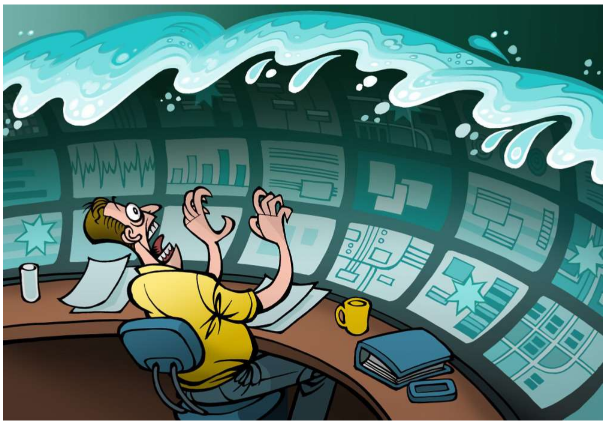
INSTITUT DE RECHERCE EN INFORMATIQUE ET SYSTEMES ALÉATOIRES

DSL/Software Language Engineering

- Software languages are software too
  - Inherit all the complexity of software development in terms of maintenance, evolution, user experience, etc.
  - Requires specialized knowledge for conducting the development of complex artifacts such as grammars, metamodels, interpreters, or type systems
    - Lot of progress since Dijktra's time, but still…
- Globalization of DSLs
  - Relationships among the languages will need to be explicitly defined in a form that corresponding tools can use to realize the desired interactions.

Mid 2010's-?

5. Data-Centric Socio-Technical Systems



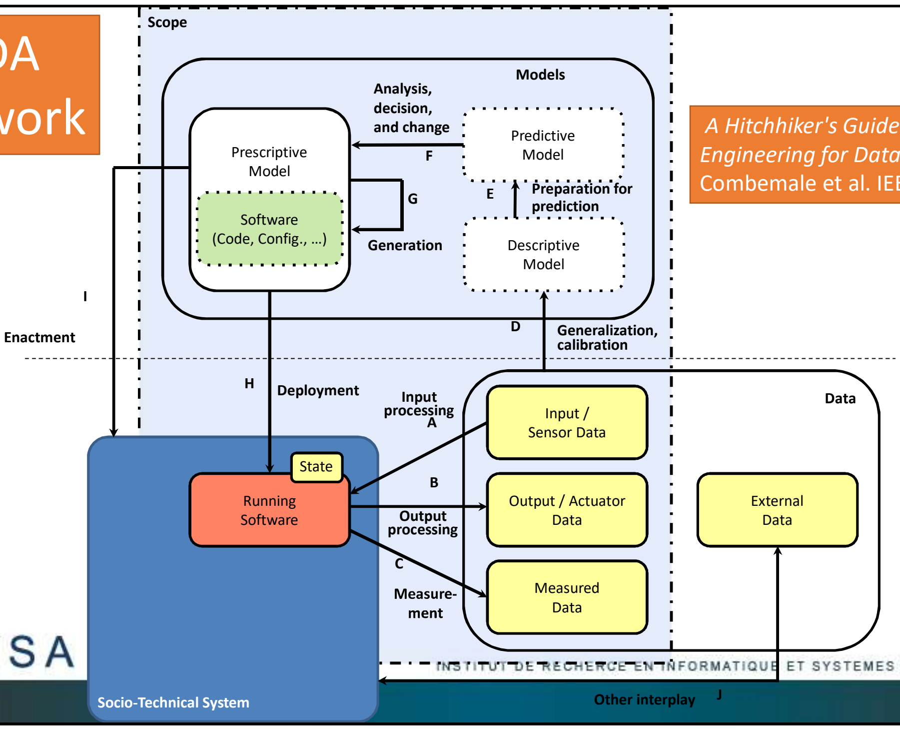
Variability: you don't know the model beforehand
=> you have to learn it from the data!

UMR IRISA
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALÉATOIRES

No longer just *a priori* Engineering Models

- Engineering Models
  - *prescriptive* during the design process of a system
  - become *descriptive* once the system is built
- Scientific Models
  - representations of some aspects of a phenomenon of the real world
  - *Descriptive* but once validated can become *Predictive*
    - If computer based simulation is needed (n-body problem) also *Engineering models*
- Inductive Models (built from Data and Machine Learning)
  - *Descriptive* of a current or past relationship
  - *Predictive* when given some hypothetical input data
  - *Prescriptive* if they are used in a larger system to make decisions
- How to organize them?

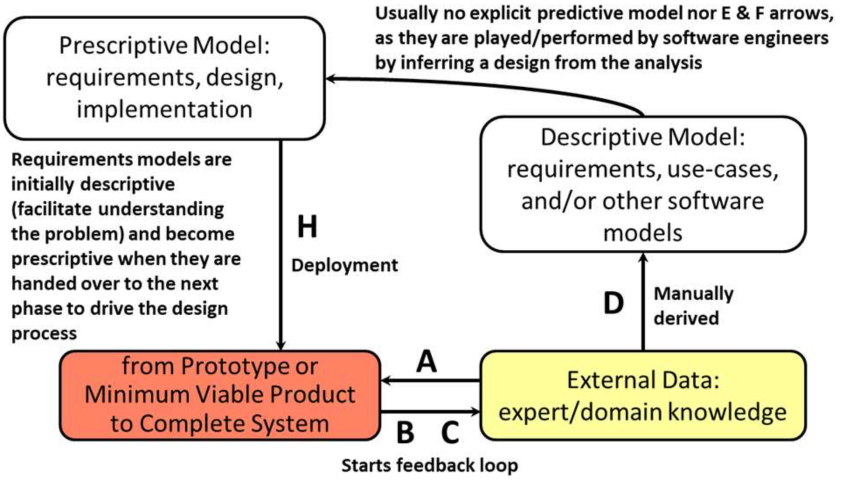
MODA framework



A Hitchhiker's Guide to Model-Driven Engineering for Data-Centric Systems
Combemale et al. IEEE Software, 2020

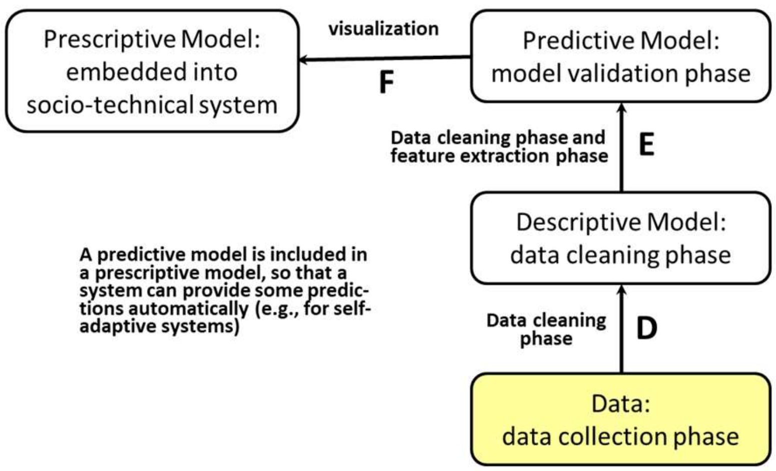
EXAMPLE

b) Iterative/Agile Methods for Software Development



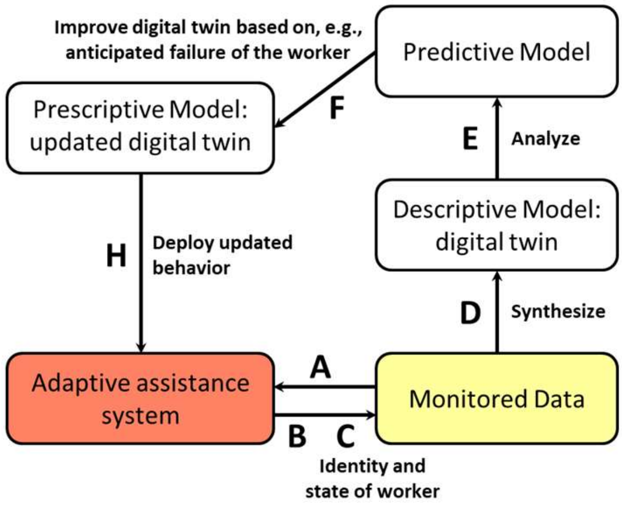
EXAMPLE

g) Commonly-used Machine Learning Pipelines



EXAMPLE

k) Development of Digital Twin Application

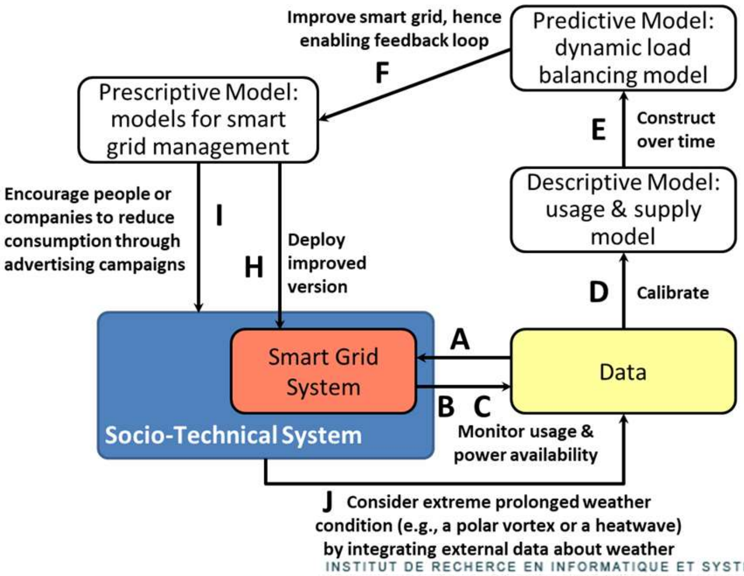


UMR IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALÉATOIRES

EXAMPLE

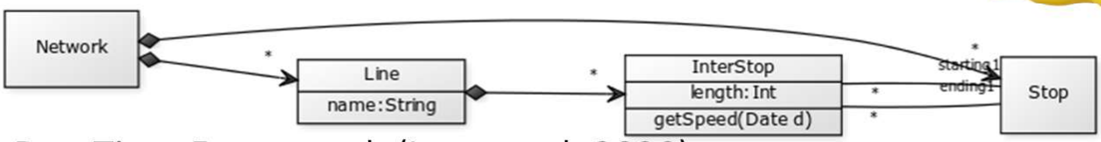
l) Development of Smart Power Grid Application



UMR IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALÉATOIRES

Integrating past/present/future



- DataTime Framework (Lyan et al. 2020)
  - optimized structure of the time series
    - capturing the past states of the system, possibly evolving over time
  - ability to get the last available value provided by the system's sensors
  - a continuous micro-learning over graph edges of a predictive model
    - query future states, either locally or globally, thanks to a composition law
  - support for what-if scenarios

UMR IRISA

Some Challenges and Opportunities

- What methods for designing the data processing pipeline?
  - from observations to decisions
- How can we control data quality through the entire pipeline?
- How can ML techniques be used to support design decisions?
- How can ML techniques be used w.r.t. online data processing?
  - measurement overhead needs to be kept low
- How can we systematically deal with data uncertainty?
- …

UMR IRISA

Acknowledgement

DiverSE
Diversity-Centric Software Engineering

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

- *In particular M. Acher, B. Combemale, O. Barais*

@jmjezequel

www.irisa.fr @irisa_lab

Institut de Recherche en Informatique et Systèmes Aléatoires