



Multi-Level Language Engineering, Modeling and Software Development with the FMML^x and the XModeler^{ML}

LE₄MM

Tony Clark
Ulrich Frank
Daniel Töpel
Michael Bartels
Luca Mattei

Before We Start ...



What is the difference between a model and a modeling language?



Why would you generate code from models?



Are objects on M0 parts of models?



What is the difference between programs and models?

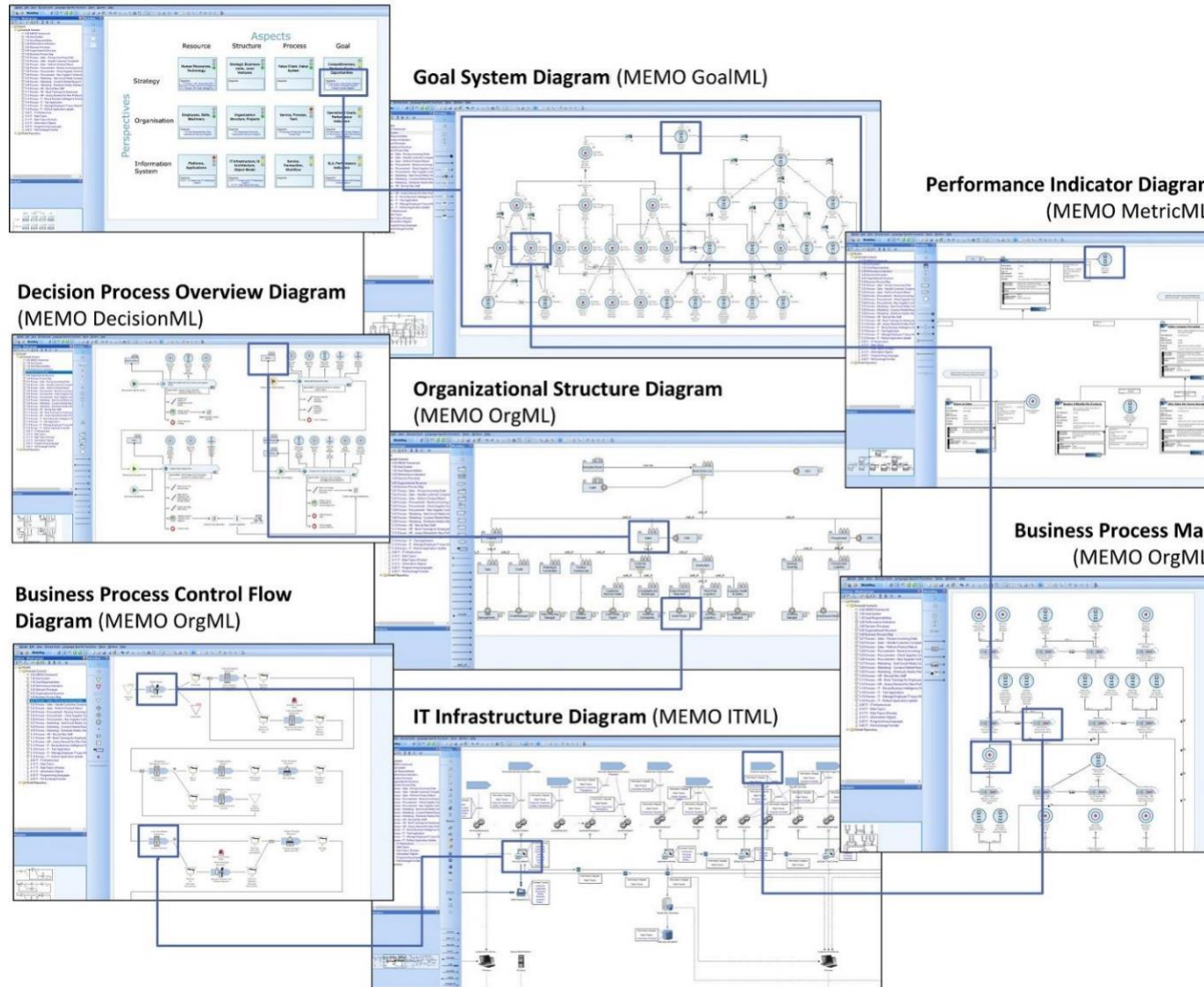


What is your vision of future enterprise software?

- Motivation
- FMML^x
- XModeler^{ML}
 - Components
 - Integrated Development and Use of Languages, Models and Apps
- Development and Use of Textual DSLs

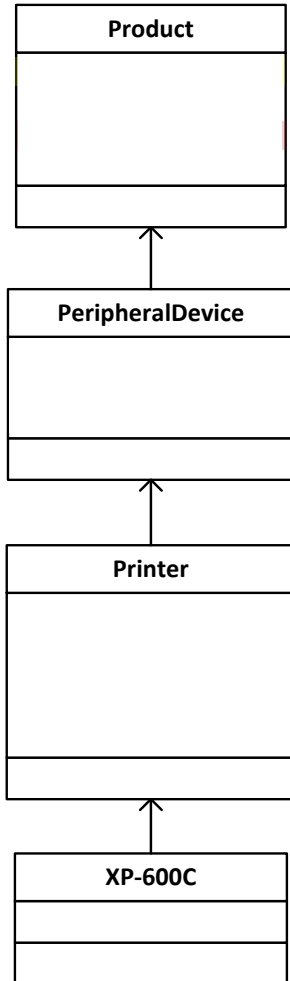
Background: DSMLs for Enterprise Modeling

MEMO Overview Framework

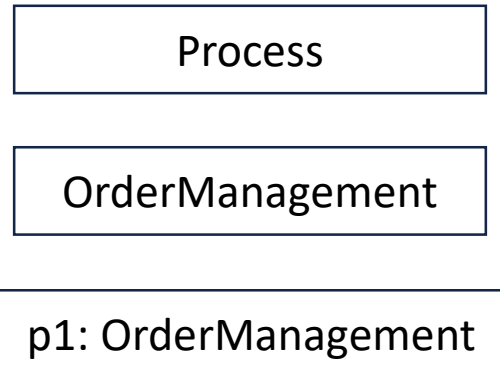


- lack of expressiveness
- unnatural dichotomy of language and model
- fundamental design conflicts
- pain of model/code synchronization
-

Lack of Expressiveness

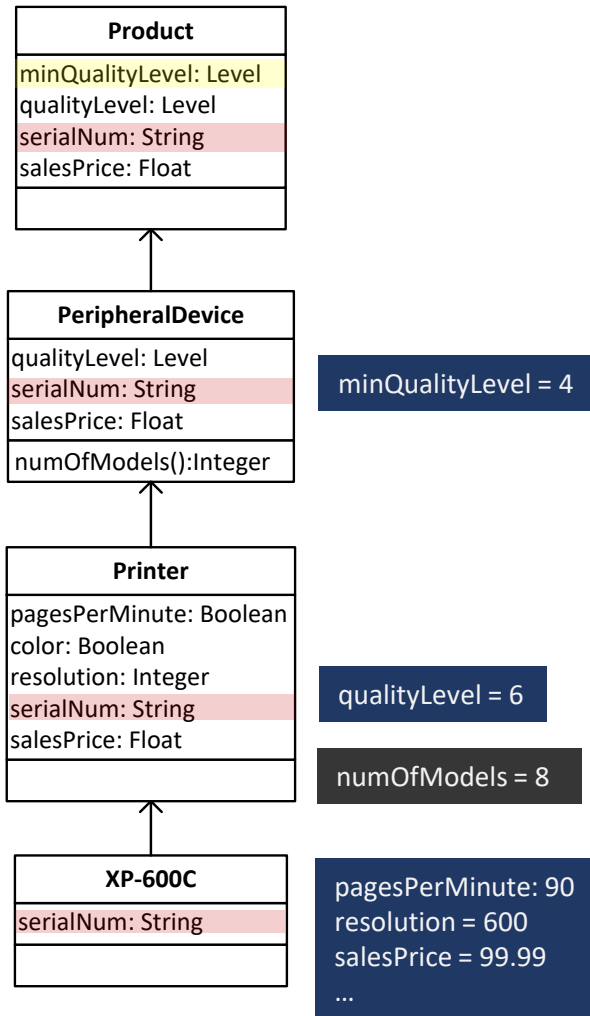


Generalization or Classification?



Where to specify that every process execution has a start time?

Lack of Expressiveness



Sometime no clear distinction between instantiation or specialisation – both make sense.

Every (meta) class is an object (has state, can execute operations).

There are multiple levels of classification.

Instantiation may be deferred to lower levels.

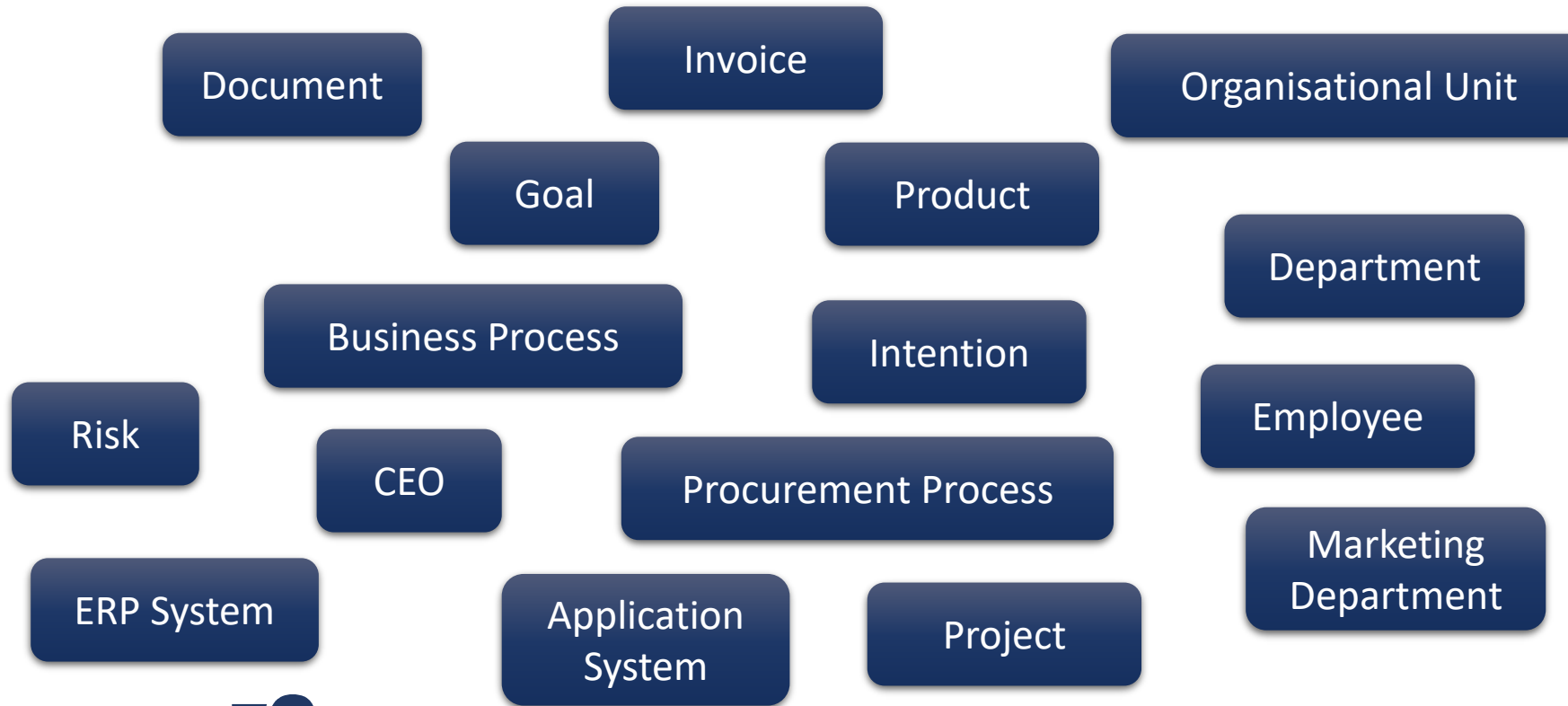


Iron Law: “Express knowledge at the highest level.”



How could that be implemented?

Distinction of Language and Model

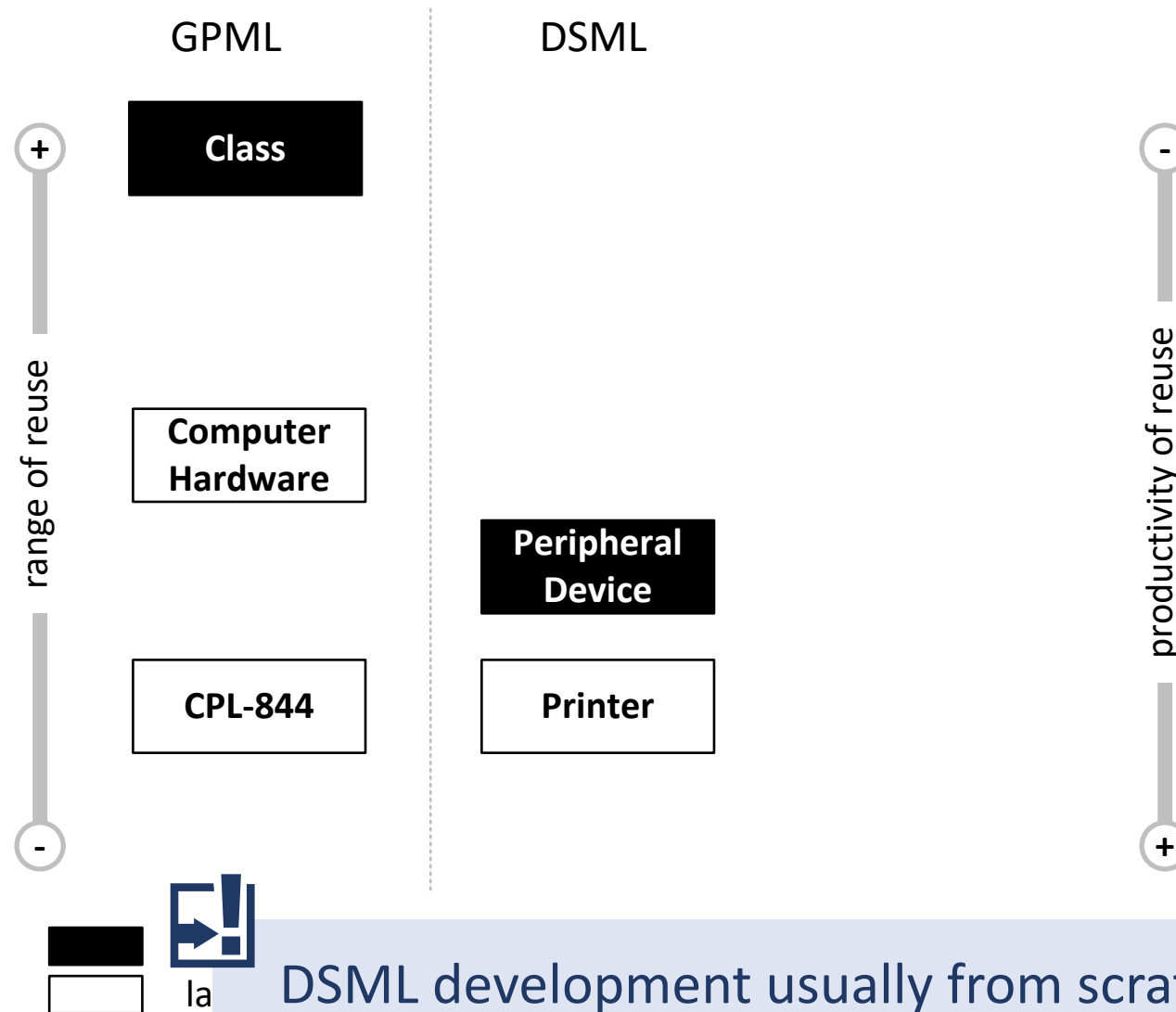


Language concept or language application?



Traditional paradigm requires distinction, but how?

Power-Generality Conflict



DSML development usually from scratch!

- Semantics promotes reuse. productivity of reuse
- Semantics compromises reuse. range of reuse (economies of scale)

- Semantics promotes integration. efficiency of communication
- Semantics compromises integration. openness of communication

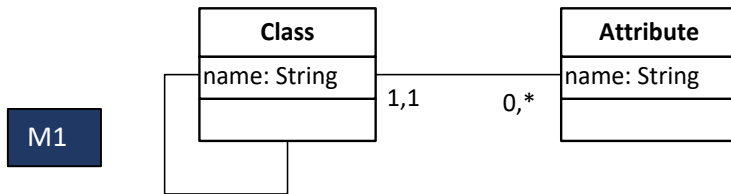
- Semantics promotes flexibility. through abstraction
- Semantics compromises flexibility. „loose coupling“

Problem: Synchronization of Model and Code

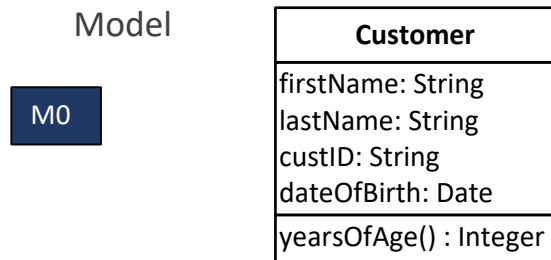
Modelling Environment

Programming Environment

Meta-Model



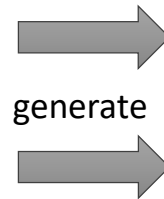
Model



Code

```
class Customer
{
    String firstName;
    String lastName;
    Date dateOfBirth;

    public int yearsOfAge()
    ....
}
```



Program instance

M0



Why is there need to generate code anyway?

- new language paradigm
- allows for an arbitrary number of classification levels
- ... and deferred instantiation
- Every class at any level is an object.
- first introduced in 2001 by Atkinson and Kühne
- with roots going back to the early 90s
- focus mainly on modelling, not on programming languages



13 Tony Clark, Ulrich Frank, Daniel Töpel, Michael Bartels, Luca Mattei | Multi-Level Language Engineering

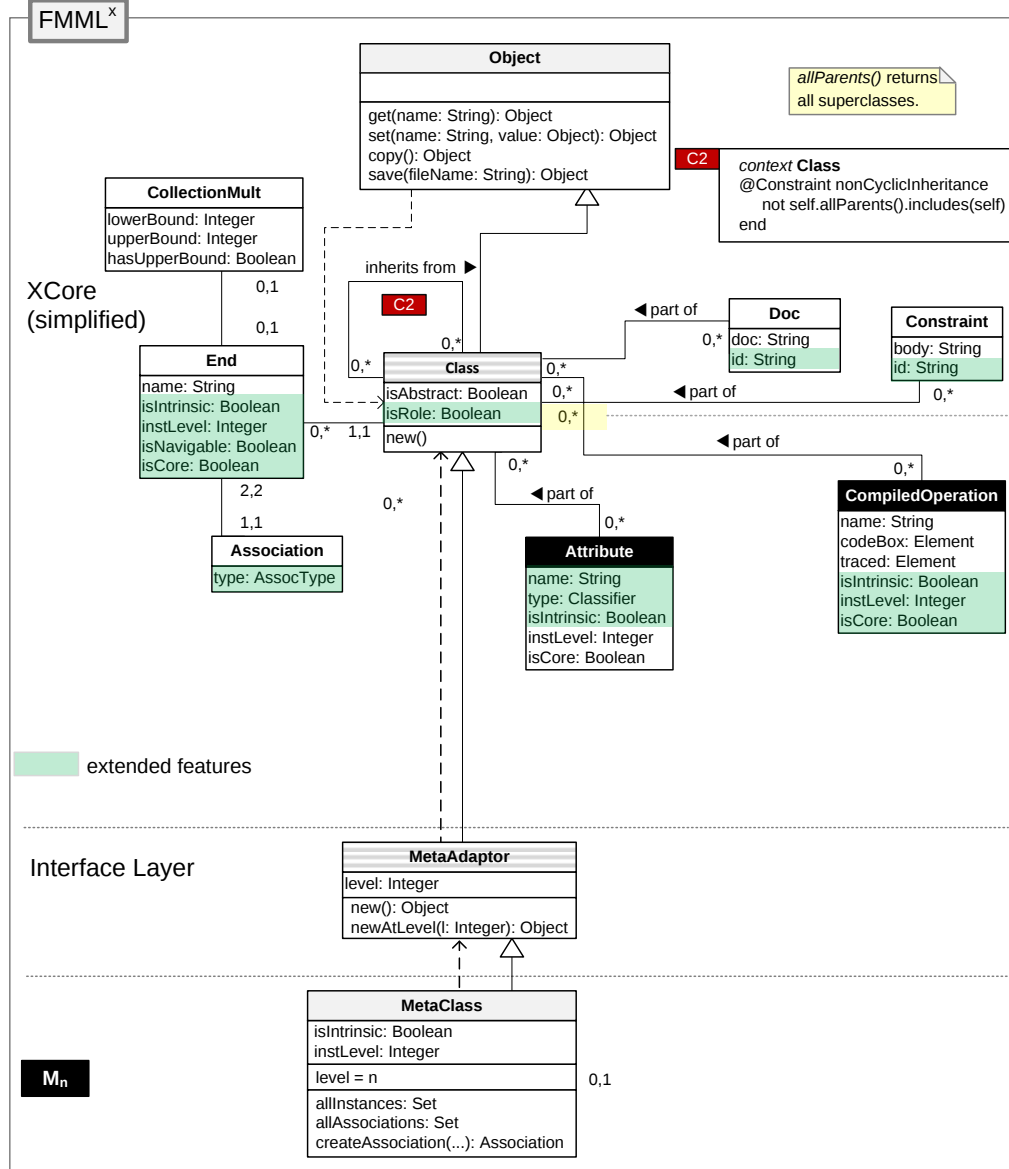
- XModeler
 - language engineering workbench
 - core language (Xcore) both reflexive and extensible
 - allows for an arbitrary number of (implicit) classification levels, but does not provide explicit levels and deferred instantiation
- both an instance of itself and a basis for defining a wide range of co-existing language variants – like the FMML^x
- common representation of programs and models
- models at any level executable

■ **FMML^x** Multi-level modeling and execution language

■ **XModeler^{ML}** Multi-level engineering, modeling, programming and execution environment, enables creation and use of graphical and textual DSL

is

FMML^x: Metamodel & Generic Notation



4	^MetaClass^
3	inRangeSince: Date[1]
3	maxWarranty: Integer[1]
1	salesPrice: Float[1]
0	serialNo: String[1]
2	numOfModels(): Integer
2	totalInStock(): Integer
1	inStock(): Integer
0	price(): Float
1	minPrice
0	uniqueSerial

3	^Product^
1	salesPrice: Float[1] (from Product)
0	serialNo: String[1] (from Product)
2	↑ numOfModels(): Integer (from Product)
2	↑ totalInStock(): Integer (from Product)
1	↑ inStock(): Integer (from Product)
0	↑ price(): Float (from Product)
inRangeSince = 09 Sep 2014	
maxWarranty = 0	

2	^PeripheralDevice^
1	resolution: Integer[1]
1	salesPrice: Float[1] (from Product)
0	serialNo: String[1] (from Product)
1	↑ inStock(): Integer (from Product)
0	↑ price(): Float (from Product)
numOfModels()-> 2	
totalInStock()-> 2	

1	^Printer^
0	serialNo: String[1] (from Product)
0	↑ price(): Float (from Product)
resolution = 300	
salesPrice = 99.99	
inStock()-> 1	

0	^ZT800^
zT8001	
serialNo = Z803	
price()-> 99.99	



Every class is an object!

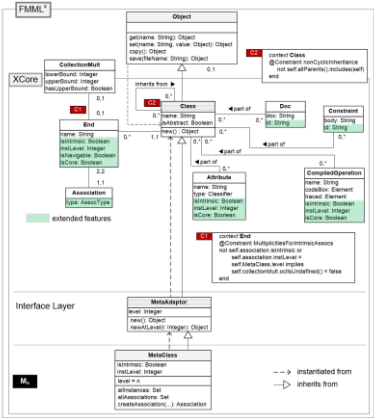
Meta Model (Meta-Modeling/Programming Language)

Representations

Text
GUIs ..

Objects at any Level
(Languages, executable models, programs)

Diagram
Code



XModeler^{ML}: Components of the Environment

The screenshot displays the XModeler ML environment with several key components:

- Diagram Editor:** A large window showing a complex diagram of relationships and elements. It includes a sidebar with a tree view of the model structure and a main area for editing the diagram.
- Instance Browser:** A window titled "Object Browser for ACD_400" showing instances of the ACD_400 class. It includes a table of instances and a "Slots" section for editing specific attributes like "serialNo".
- Model Browser:** A window titled "Model Browser" showing a tree view of the model structure. It includes a "Project" section with a list of models and a "Status" section with buttons for "Update", "View Updated", and "Do not push".
- Console:** A window at the bottom right showing the execution log. It includes a "Loading" section with a list of loaded files and a "GC" section showing memory usage statistics.

Labels with dashed lines point to the following components:

- diagram editor
- instance browser
- model browser
- console

- FMML^x allows for
 - conjoint design of models and modeling languages
 - at an arbitrary number of classification levels
- thus
 - enabling the specification of DSLs with more generic DSLs
 - hence, contributing to productivity, flexibility and integrity
 - relaxing fundamental design conflicts
- XModeler^{ML} enables
 - common representation of models and code
 - execution of models at any level
 - navigation from GUI to models (diagram or textual) at runtime
 - self-referential architectures of applications
 - contributes to user empowerment

- Further development of multi-level modeling method
- Maintenance/management of multi-level models
- New version of editor for defining graphical notations
- Multi-level modeling of processes (dynamic abstractions)
- Reconstruction of existing DSMLs for enterprise modelling
- Prototypical implementation of self-referential ERP system

LE4MM

Imagine...

you had a tool ...

<https://www.wi-inf.uni-duisburg-essen.de/LE4MM/>