Domain-Specific Modeling in Practice

29 March 2008

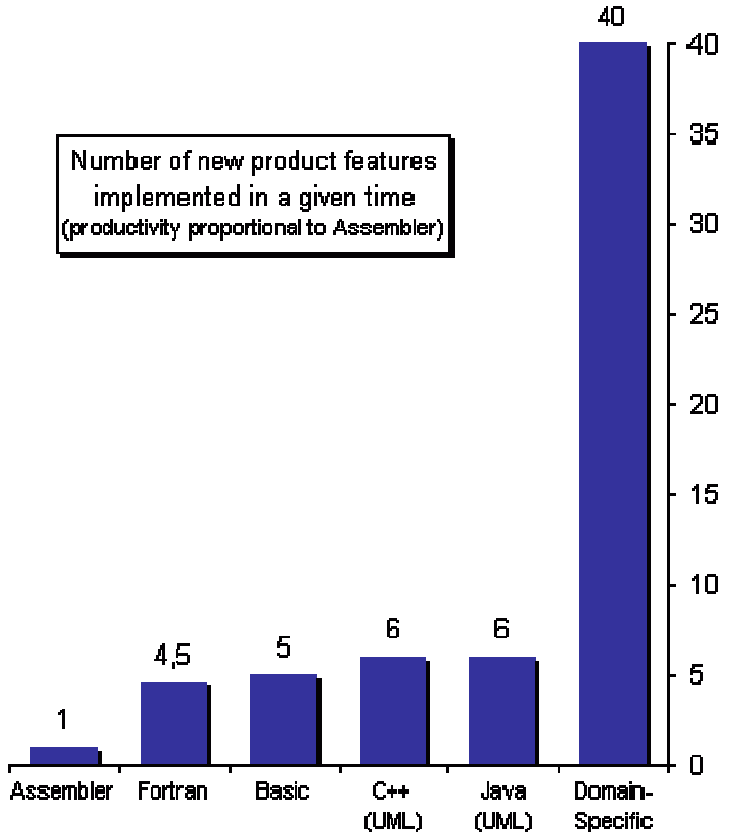
Juha-Pekka Tolvanen

Outline

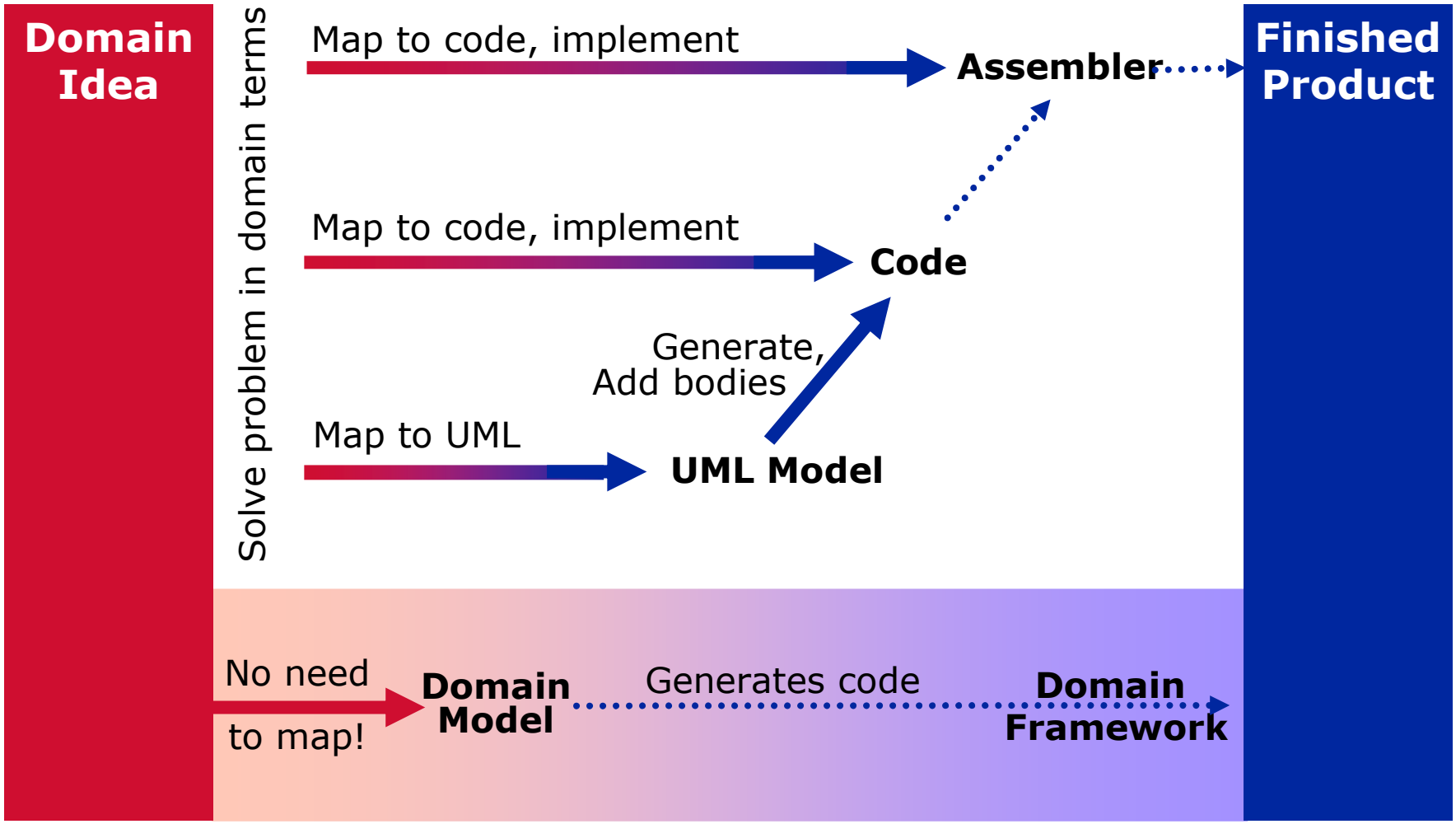
- Why Domain-Specific Modeling?
- What is Domain-Specific Modeling?
- Examples and experiences from the industry
- Living in the four levels
- Topics for research

How productivity has improved?

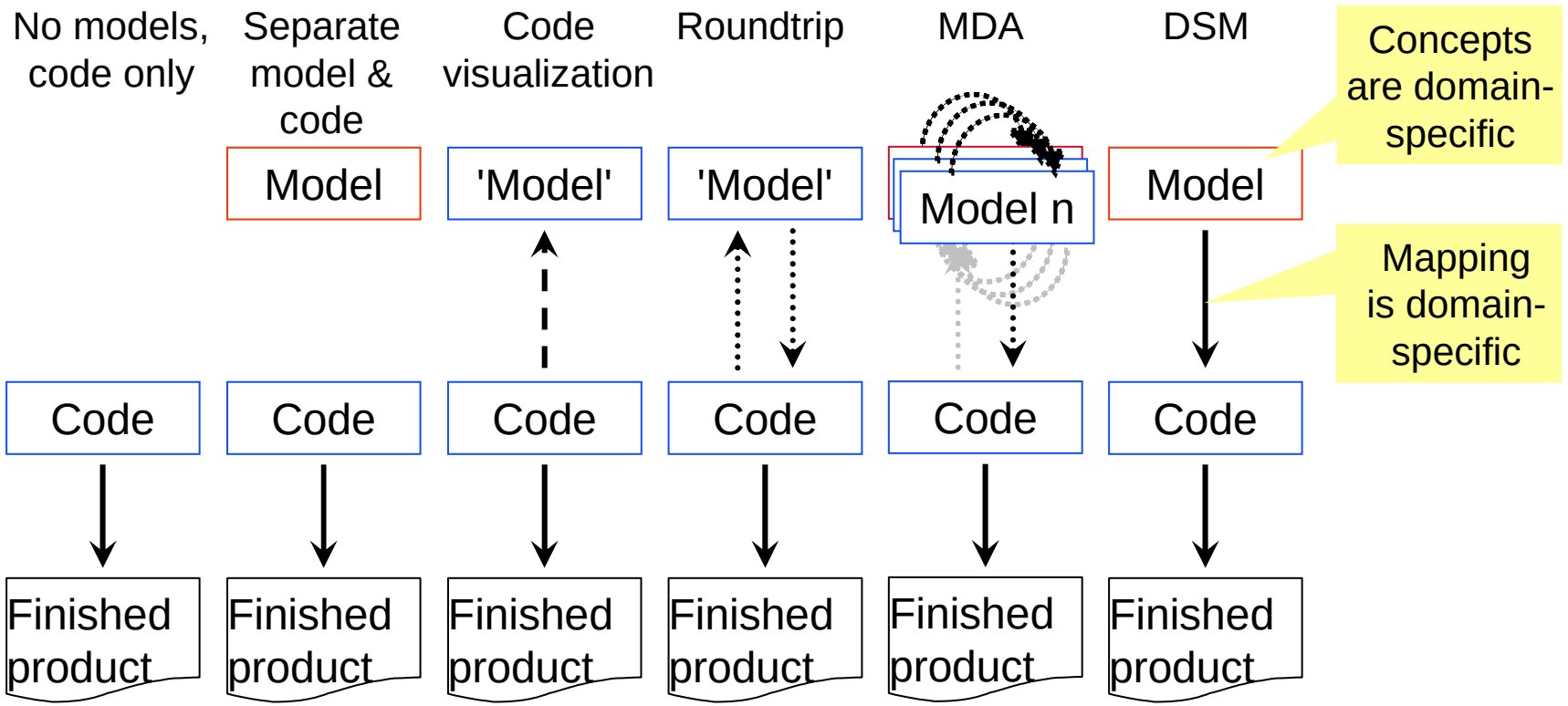
- "The entire history of software engineering is that of the rise in levels of abstraction"
- Newer programming languages have not increased productivity
- UML and visualization of code have not increased productivity
- Abstraction of development can be still raised by moving from solution domain to problem domain
  - Inside one company, product family, business area etc.



Modeling domain vs. modeling code



How do we use models?

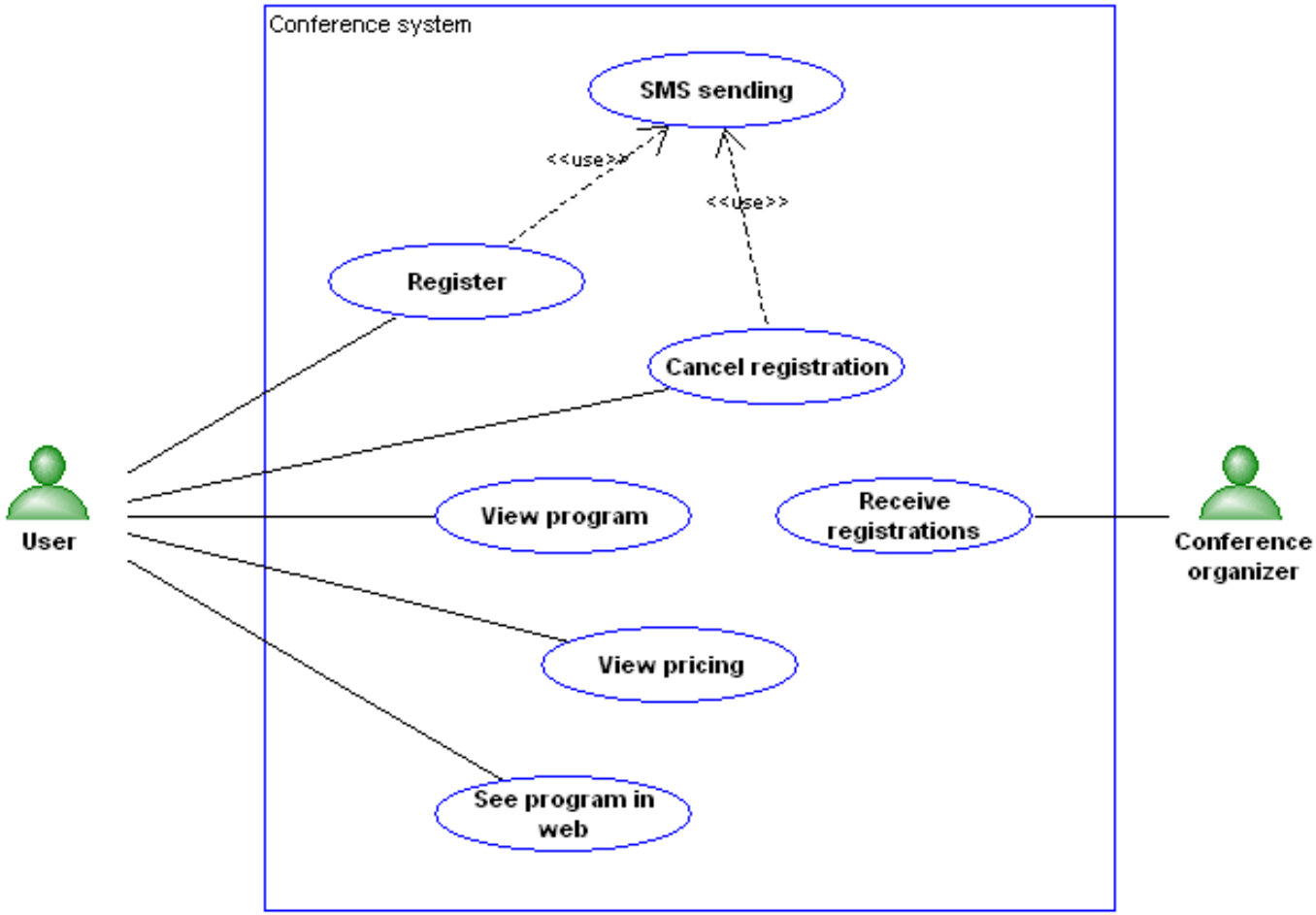


- Model alone should be sufficient in most cases
  - No need to look at code

Let’s inspect an example

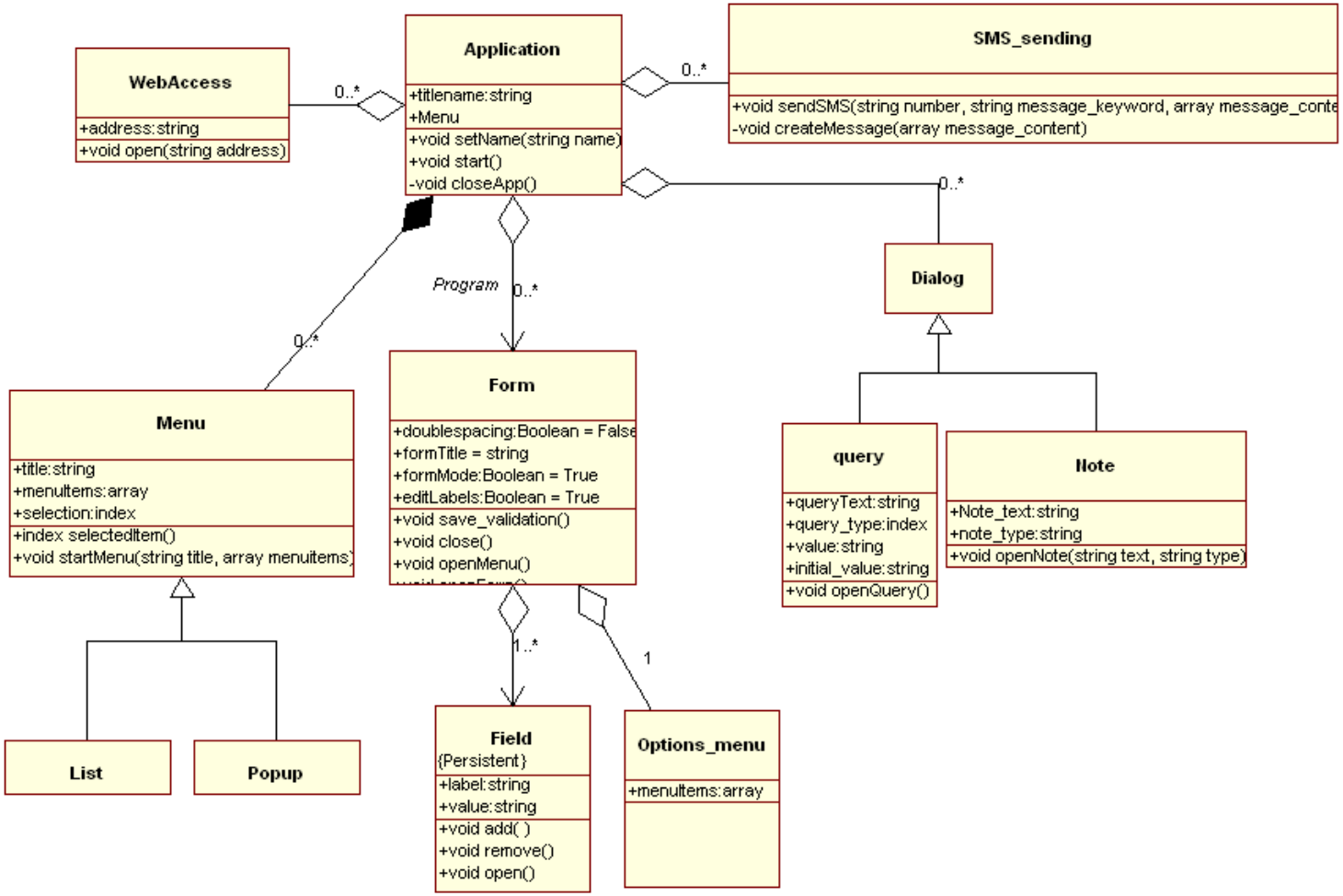
Traditional way: some modeling and then coding

- Step1: User view



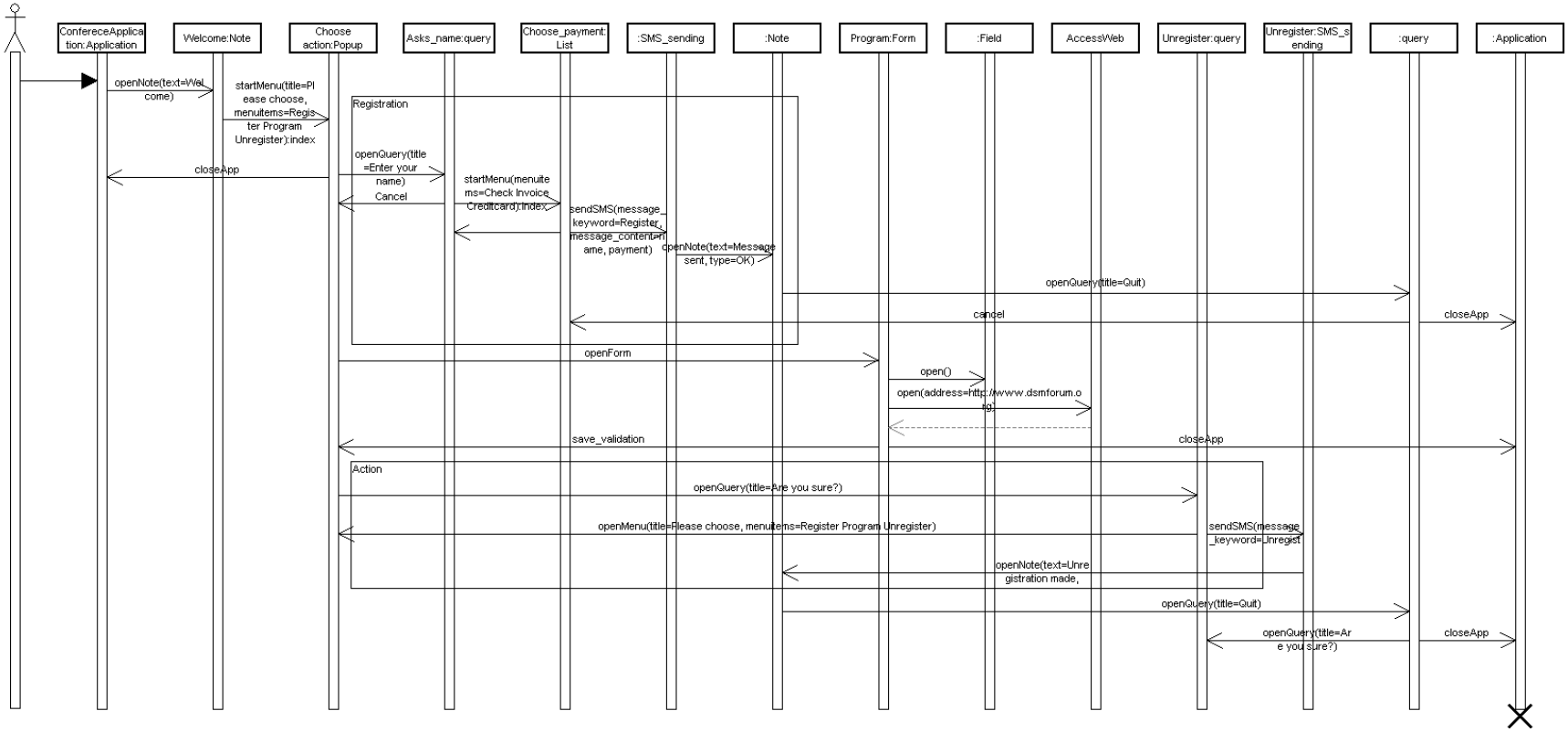
Development with UML…

- Step2: Describe static structure



Development with UML…

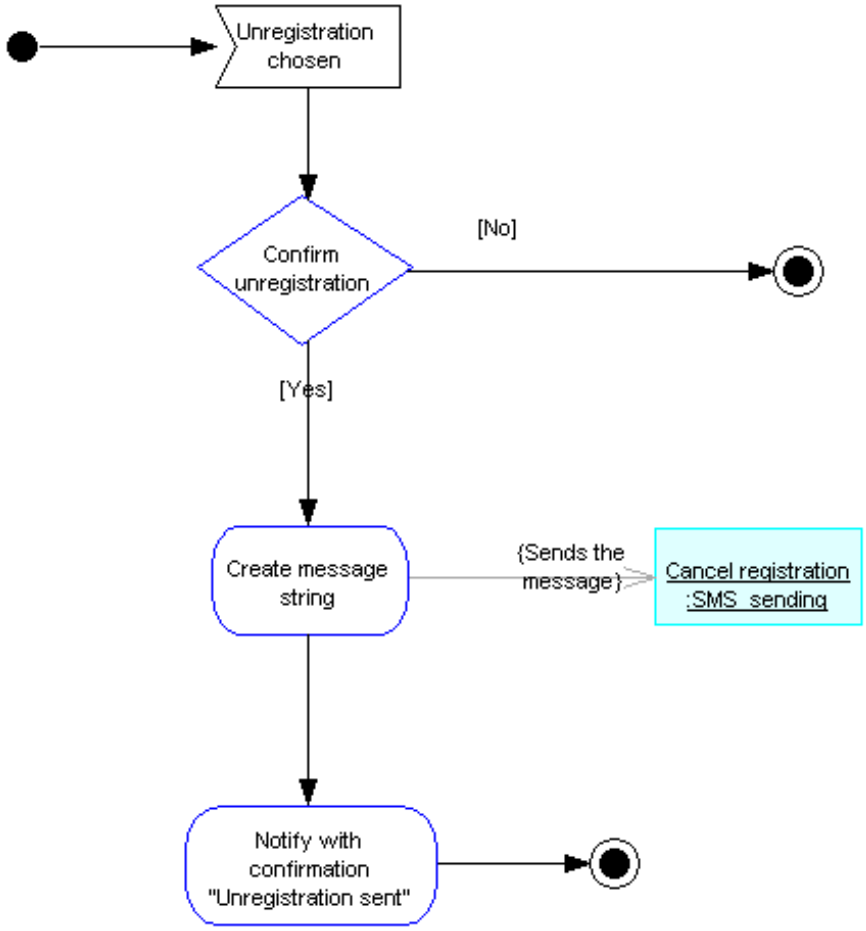
- Step3: Specify interaction



Development with UML...

- Step 4: Logic
+ user navigation
+ behavior
+ exceptions
+ etc.

+ In steps 5...N



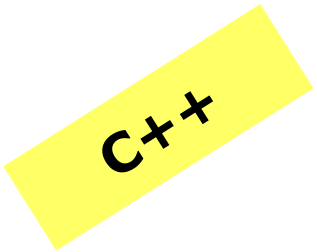
Development with UML+code

- And finally we start coding!
  - Implement the functions, access to APIs, remember the exceptions, architectural rules, UI guidelines etc.
  - ... and throw models away as they are not anymore in sync

```
...

void CGDSMSAppUi::CmdSendL() // Show notification
{         iEikonEnv->InfoWinL(_L("Confirmation"),_
                    L("SMS is in draft folder"));
          SendMessageL();
}


TBool CGDSMSAppUi::SendMessageL() // Sending SMS Message
{         TMsvEntry msvEntry = iMtm->Entry().Entry();
          CRichText& mtmBody = iMtm->Body();
          mtmBody.Reset();
          mtmBody.InsertL(0, smsNum(16400));
          SetScheduledSendingStateL(msvEntry);
}
...
```



Development with UML+code

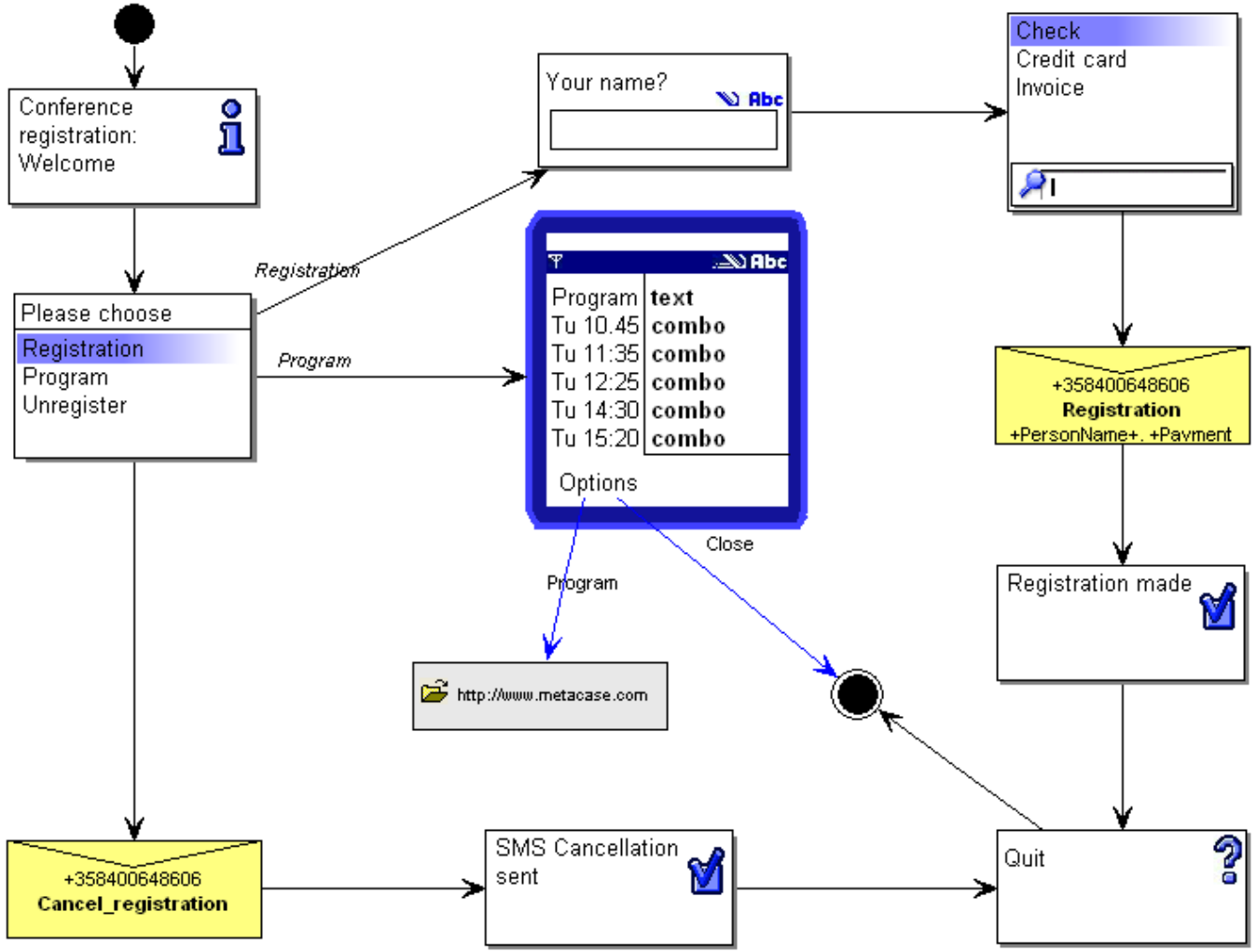
- And finally we start coding!
  - Implement the functions, access to APIs, remember the exceptions, architectural rules, UI guidelines etc.
  - ... and throw models away as they are not anymore in sync

```
...
def Query25_931():
# Query: Your name?
          global PersonNamed
          PersonNamed = appuifw.query(u"Your name?", 'text')
          if PersonNamed:
                    return (List25_275, True)
          else: # Cancel selected
                    return ((call_stack.pop()), False)

def SendSMS25_692():
# Sending SMS Cancel_registration
# Use of global variables
          string = u"Cancel_registration "
          appuifw.note(string, 'info')
          messaging.sms_send("+358400648606", string)
          return (Note25_649, False)
...
```

Python

Domain-Specific Modeling solution



after running the generator...

What is Domain-Specific Modeling

- **Captures domain knowledge (as opposed to code)**
  - Raise abstraction from implementation world
  - Uses domain abstractions
  - Applies domain concepts and rules as modeling constructs
  - Narrow down the design space
  - Focus on **single range of products**

- **Lets developers design products using domain terms**
  - ➔ Apply familiar terminology
  - ➔ Solve the RIGHT problems
  - ➔ Solve problems only ONCE!
    - directly in models, not again by writing code, round-trip etc.

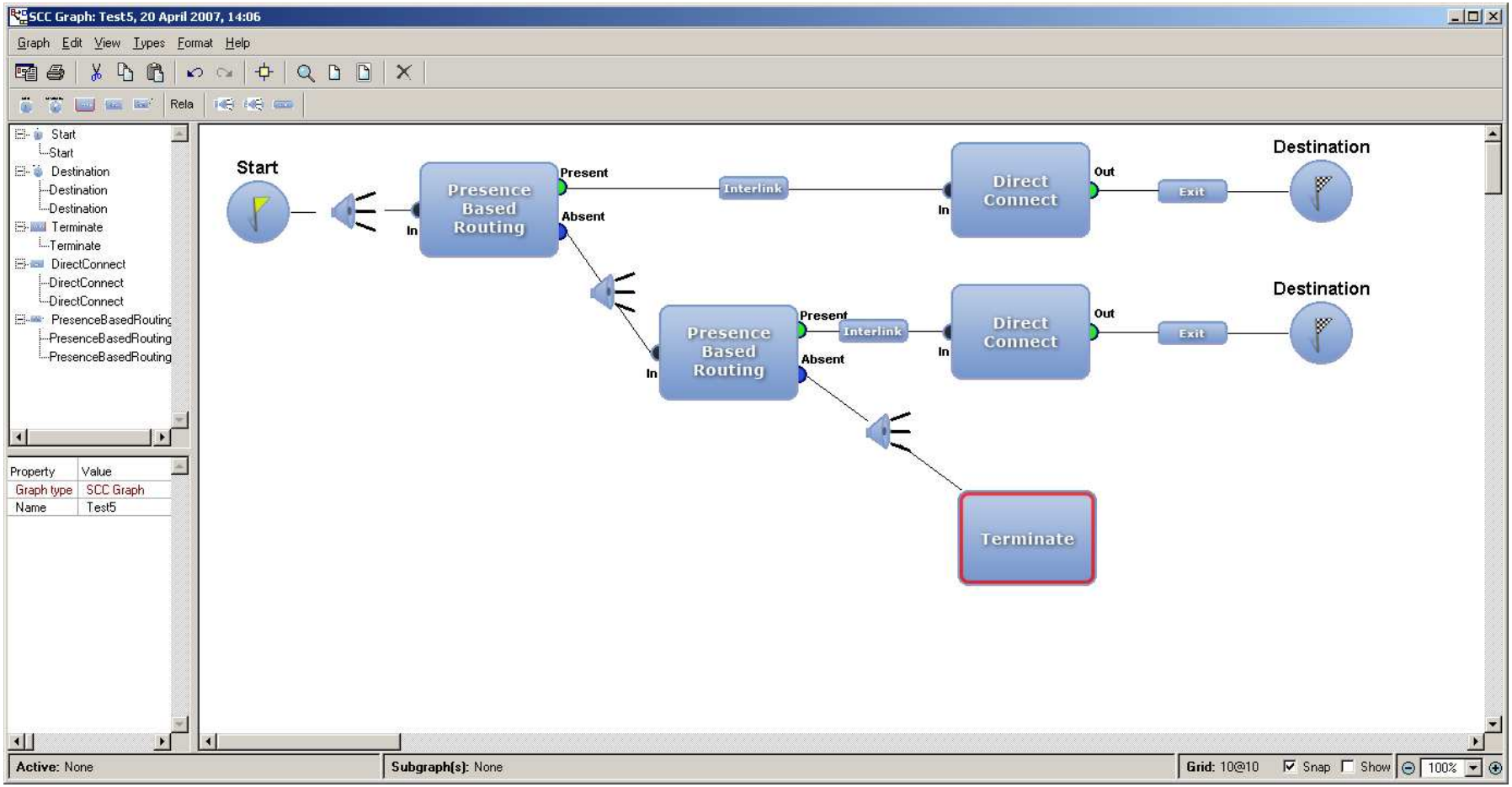
Works in any domain
(not on phone only)

Problem domain	Solution domain/ generation target
Telecom services	Configuration scripts
Insurance products	J2EE
Business processes	Rule engine language
Industrial automation	3 GL
Platform installation	XML
Medical device configuration	XML
Machine control	3 GL
Call processing	CPL
Geographic Information System	3 GL, propriety rule language, data structures
SIM card profiles	Configuration scripts and parameters
Phone switch services	CPL, Voice XML, 3 GL
eCommerce marketplaces	J2EE, XML
SIM card applications	3 GL
Applications in microcontroller	8-bit assembler
Household appliance features	3 GL
Automotive infotainment	Java, C
ERP configuration	3 GL
Handheld device applications	3 GL
Phone UI applications	C++ (C, Python, Java)
SIP services	XML

Case: IMS Service Creation*

- Rapid creation, deployment and provisioning of IP-based services
- Modeling language centralizes service concepts
- Generate all required artifacts from a single design
  - Code, configuration, documentation
- Uses a service enabling framework
  - runs on top of off-the-shelf application servers
  - industry standard SIP-servlet (JSR 116)
- ➢ Services can be created easier and faster because of the higher abstraction level
  - without the usual cross-cutting concerns seen in SIP and HTTP servlet development.

* Implemented by ICT Automation

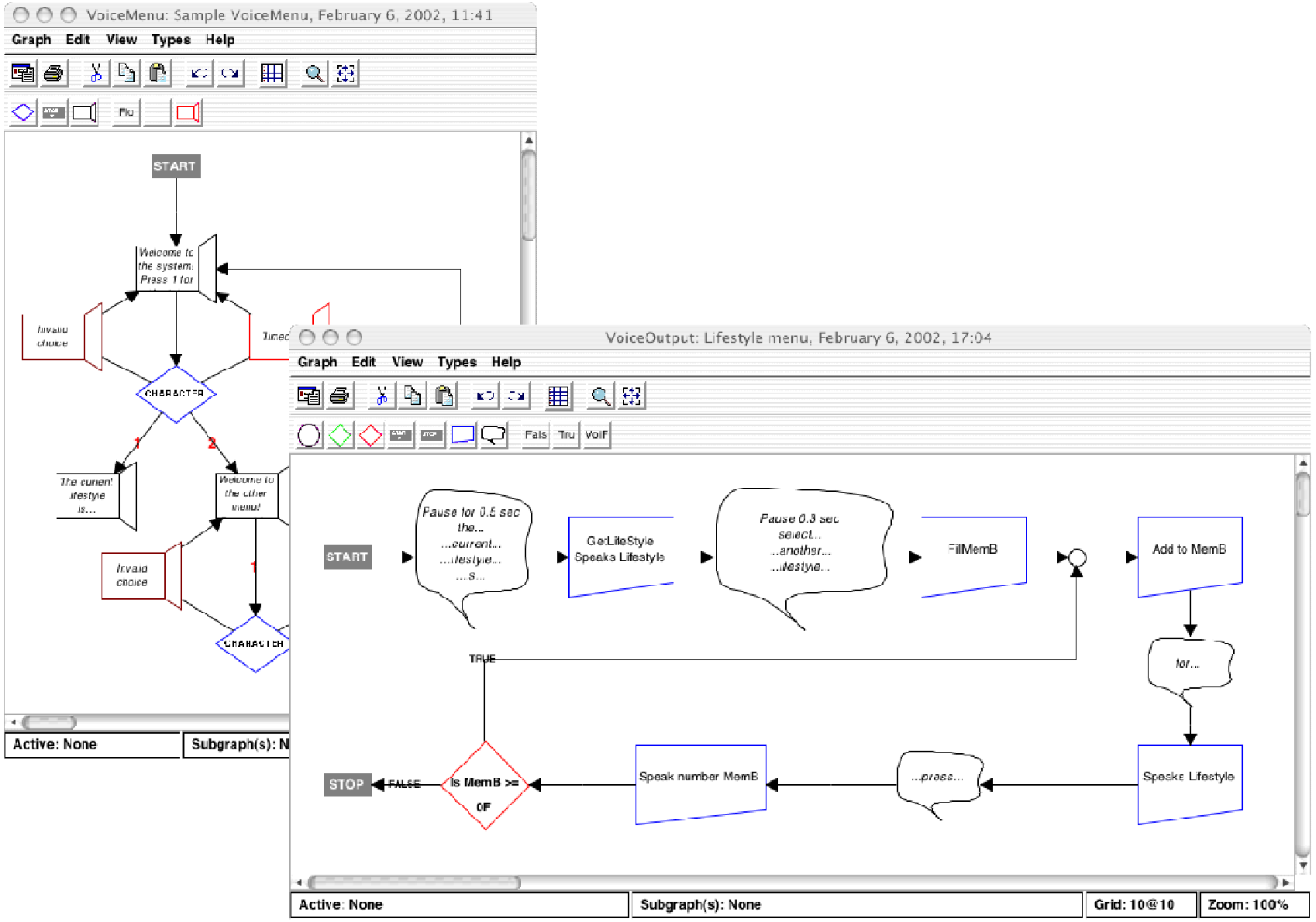


A basic sample. More complex uses include region and time based routing for e.g. Helpdesk applications.

Case: VoiceMenu for microcontroller

- Home automation system to remote control lights, heating, alarms, etc.
- VoiceMenus are programmed straight to the device with assembler-like language (8bit)
- Modeling language to define overall menu structure and individual voice prompts
- Code generator produces 100% of menu implementation

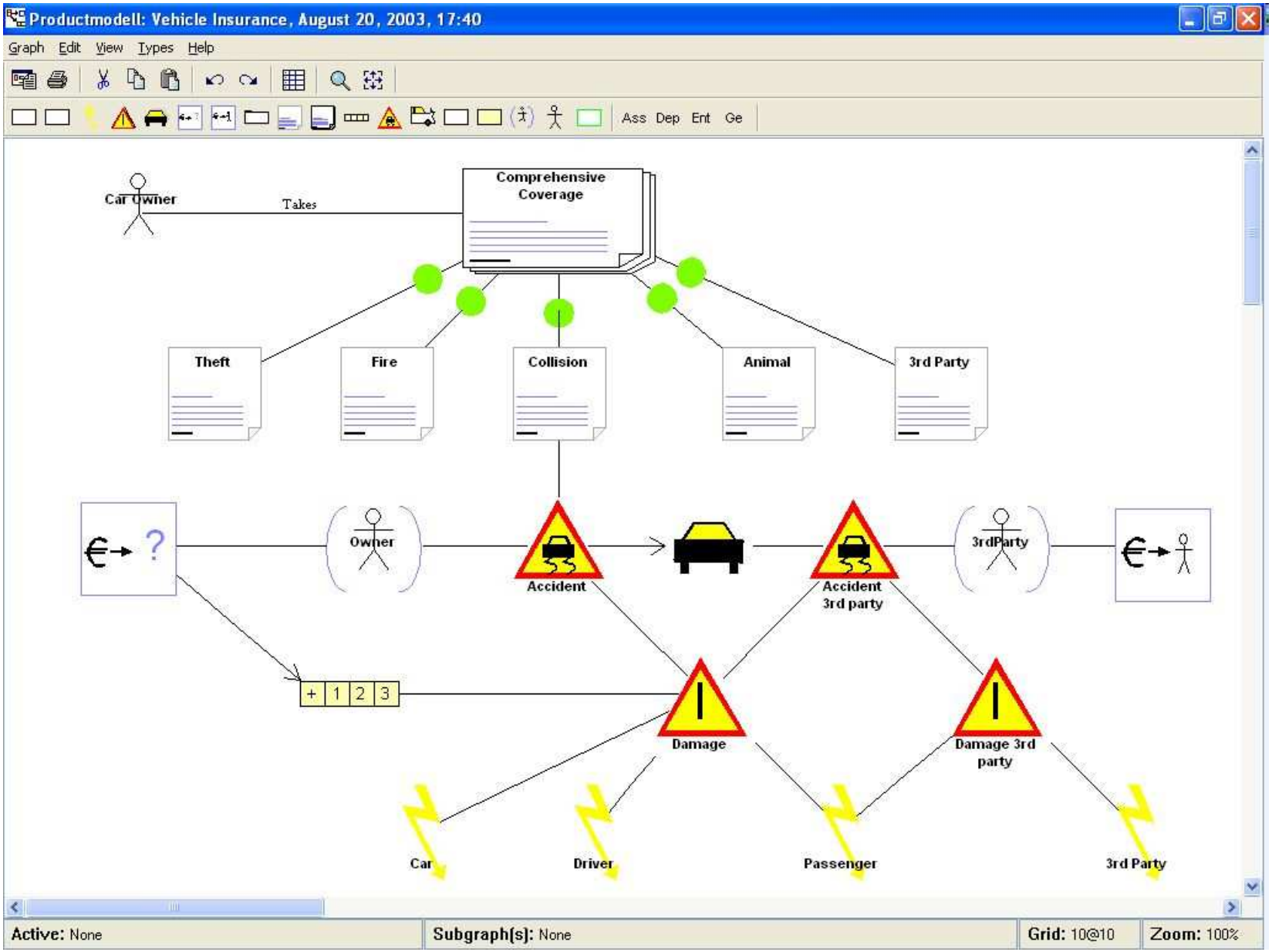
➢ Development time for a feature from a week to a day!

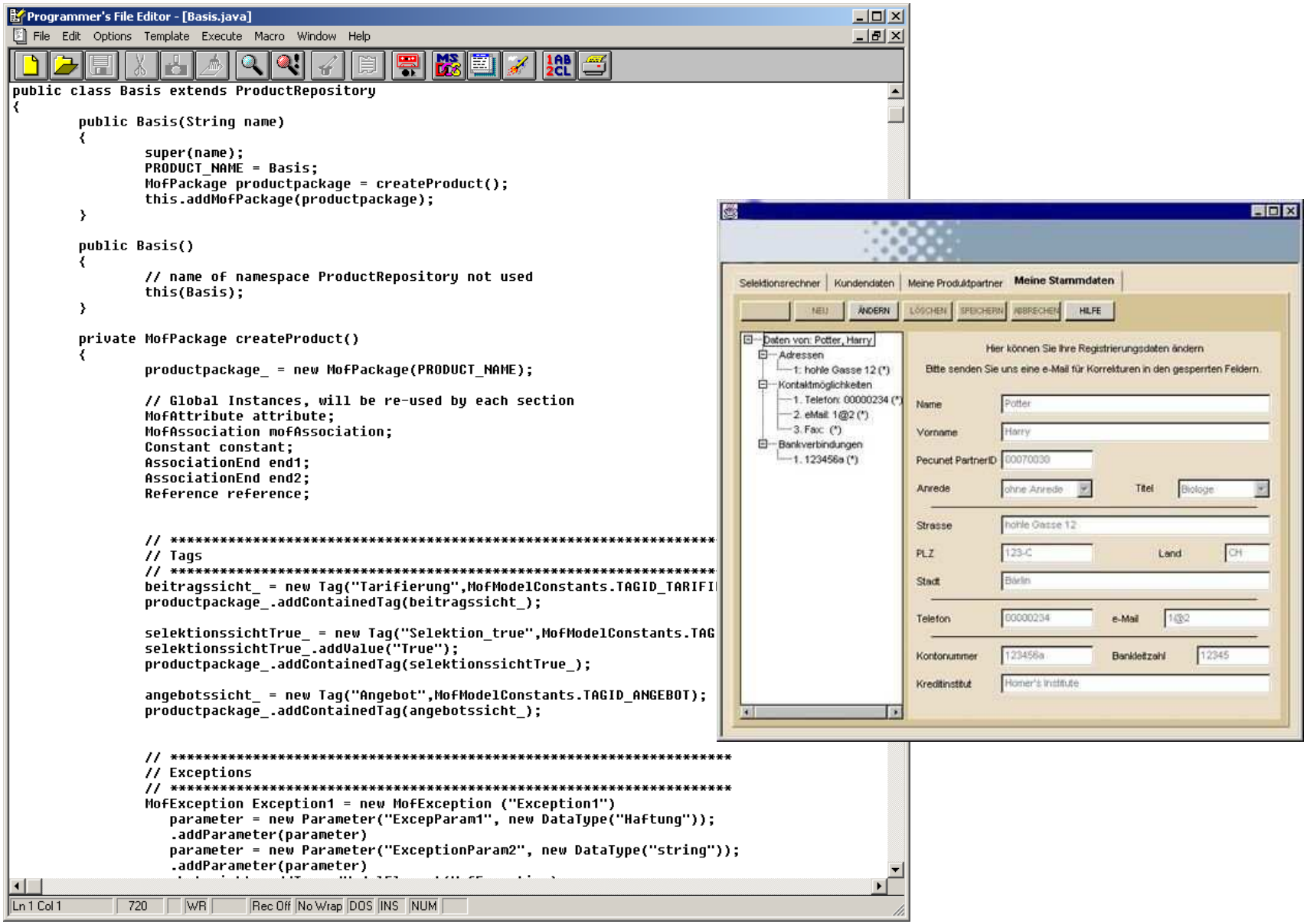


Case: Insurance products & eCommerce

- Developing portal for insurances and financial products
- Need to specify several hundred financial products
- Insurance experts visually specify insurance products and generate code to the portal

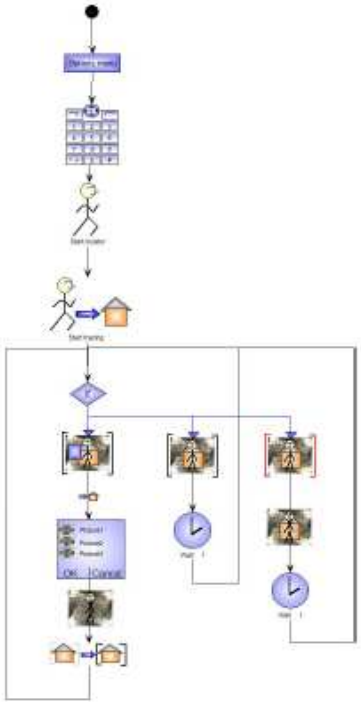
➢ Comparison to hand-writing Java after first 30 products = DSM at least 3 times faster, fewer errors



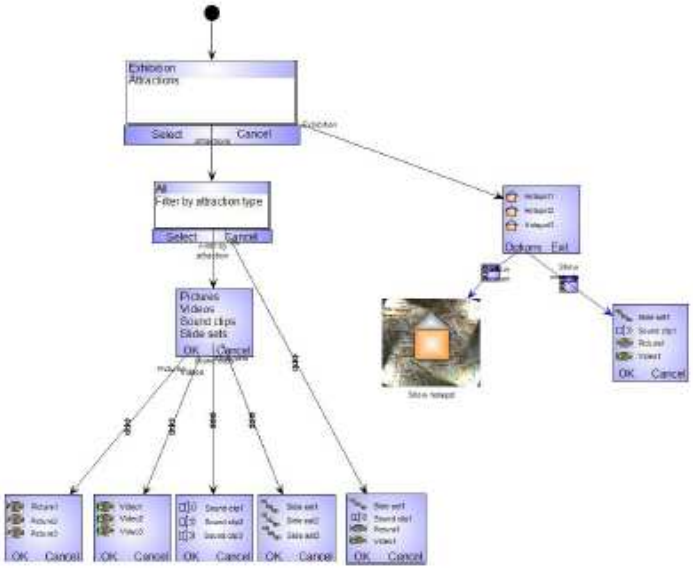


Navigation applications

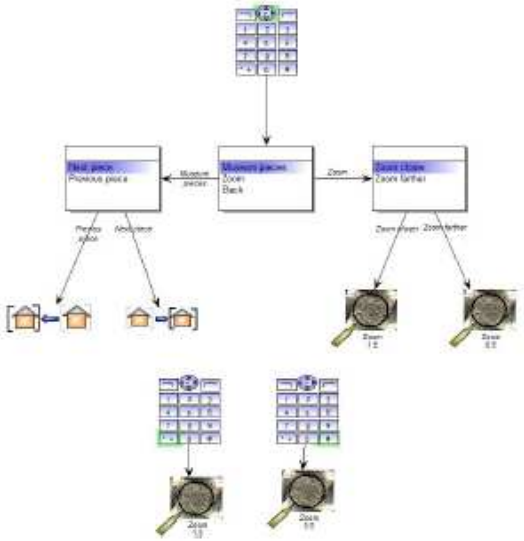
Run-time behaviour



Options menu

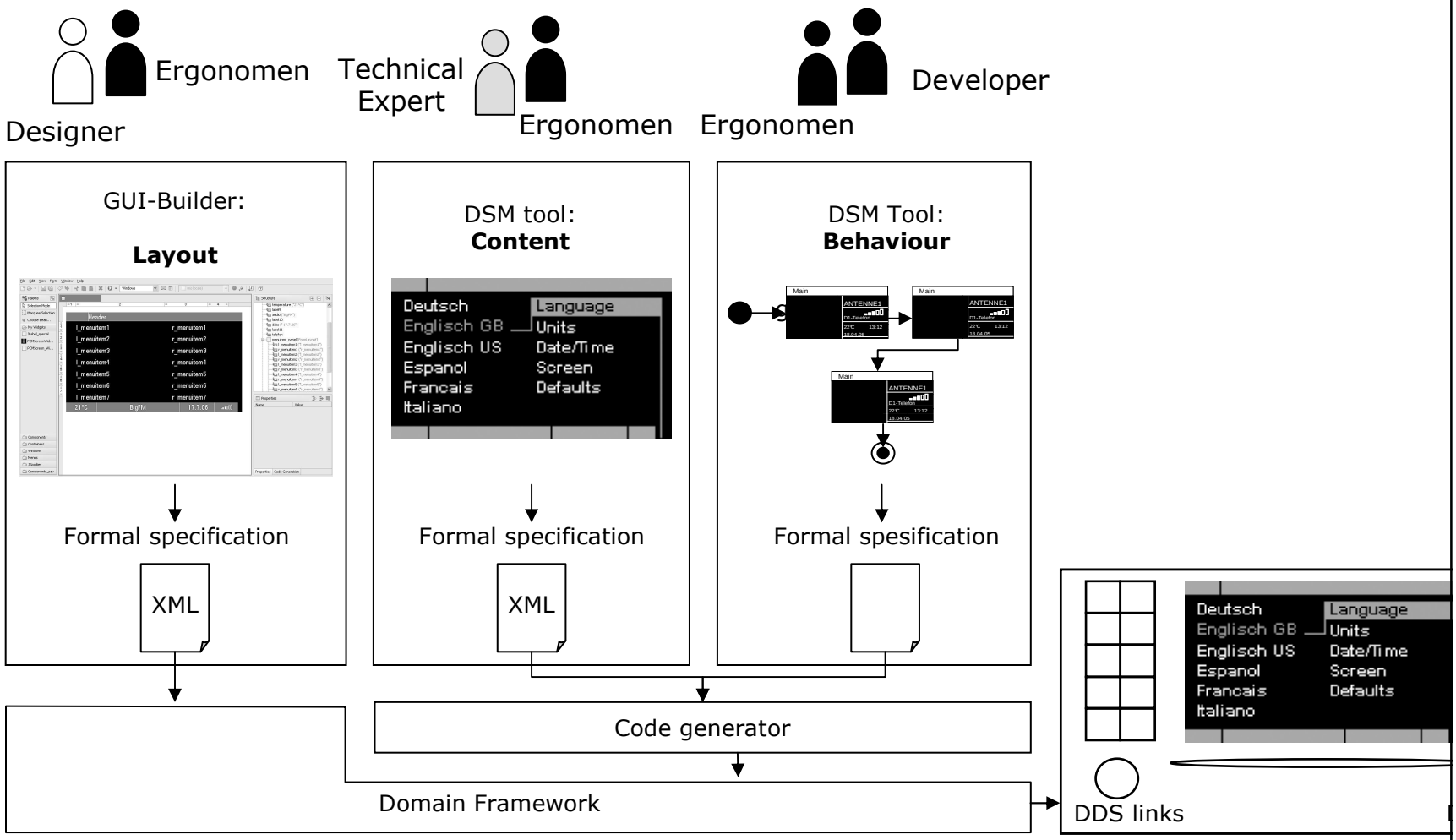


Keyboard actions



Implemented by VTT

Automotive infotainment



Bock., C., Presentation in MDD and Product Lines, Leipzig, 2006

Experiences on DSM

"**5-fold** productivity increase when compared to standard development methods" (Panasonic)

"A module that was expected to take 2 weeks now **took 1 day** from the start of the design to the finished product" (Nokia Mobile Phones)

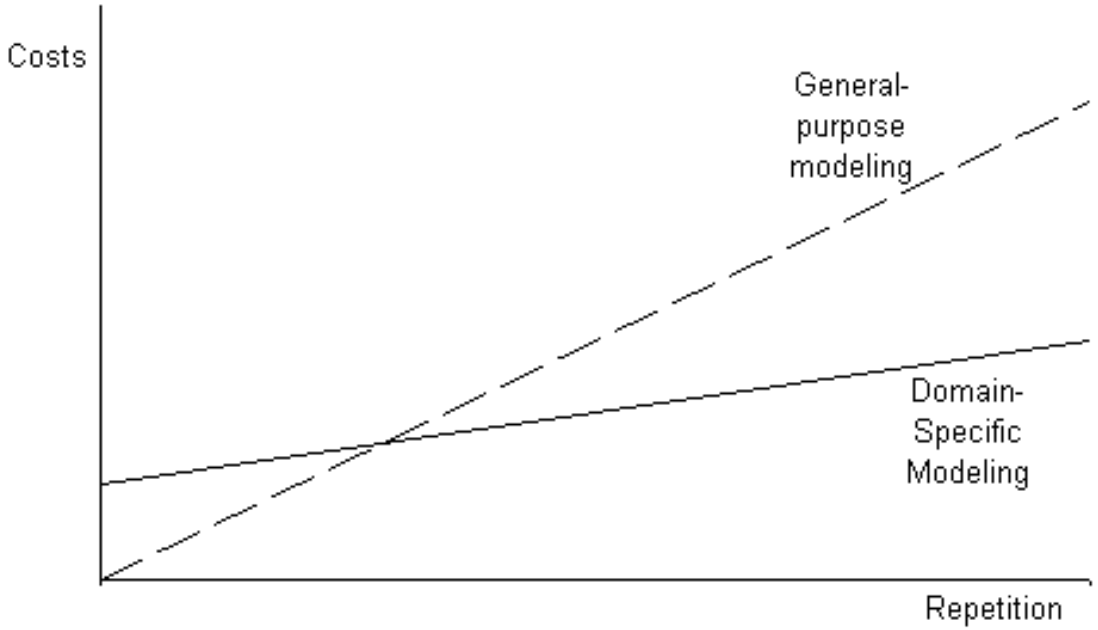
"The quality of the generated code is clearly better, simply because the modeling language **rules out errors**, eliminating them already in the design stage" (EADS)

3 times improved productivity and **50% fewer errors** when compared to earlier manual practices. (**statistically significant at confidence levels exceeding 99 percent**) (USAF, Kieburtz et al.)

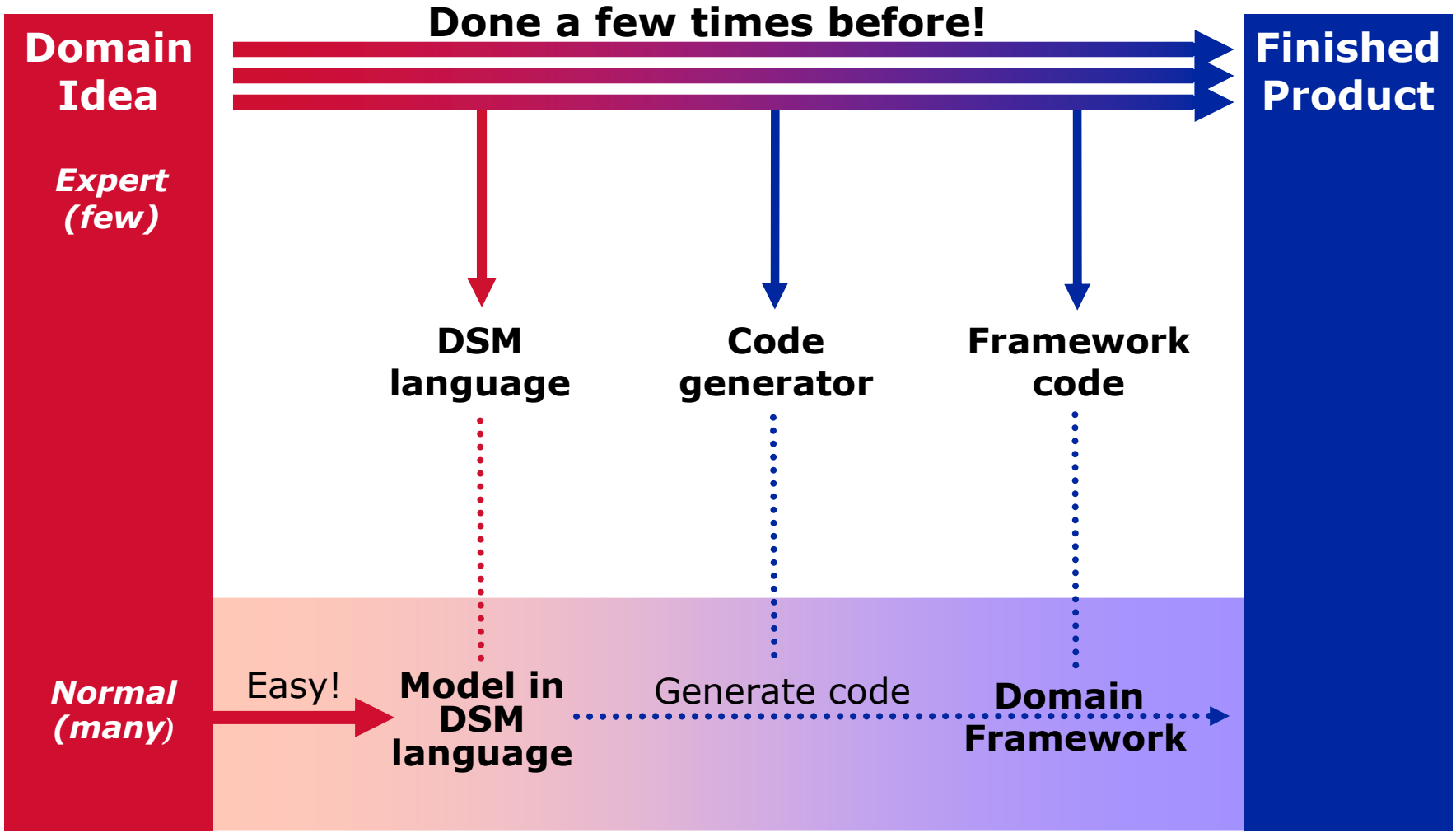
See references (slide 51)

Economics of DSM

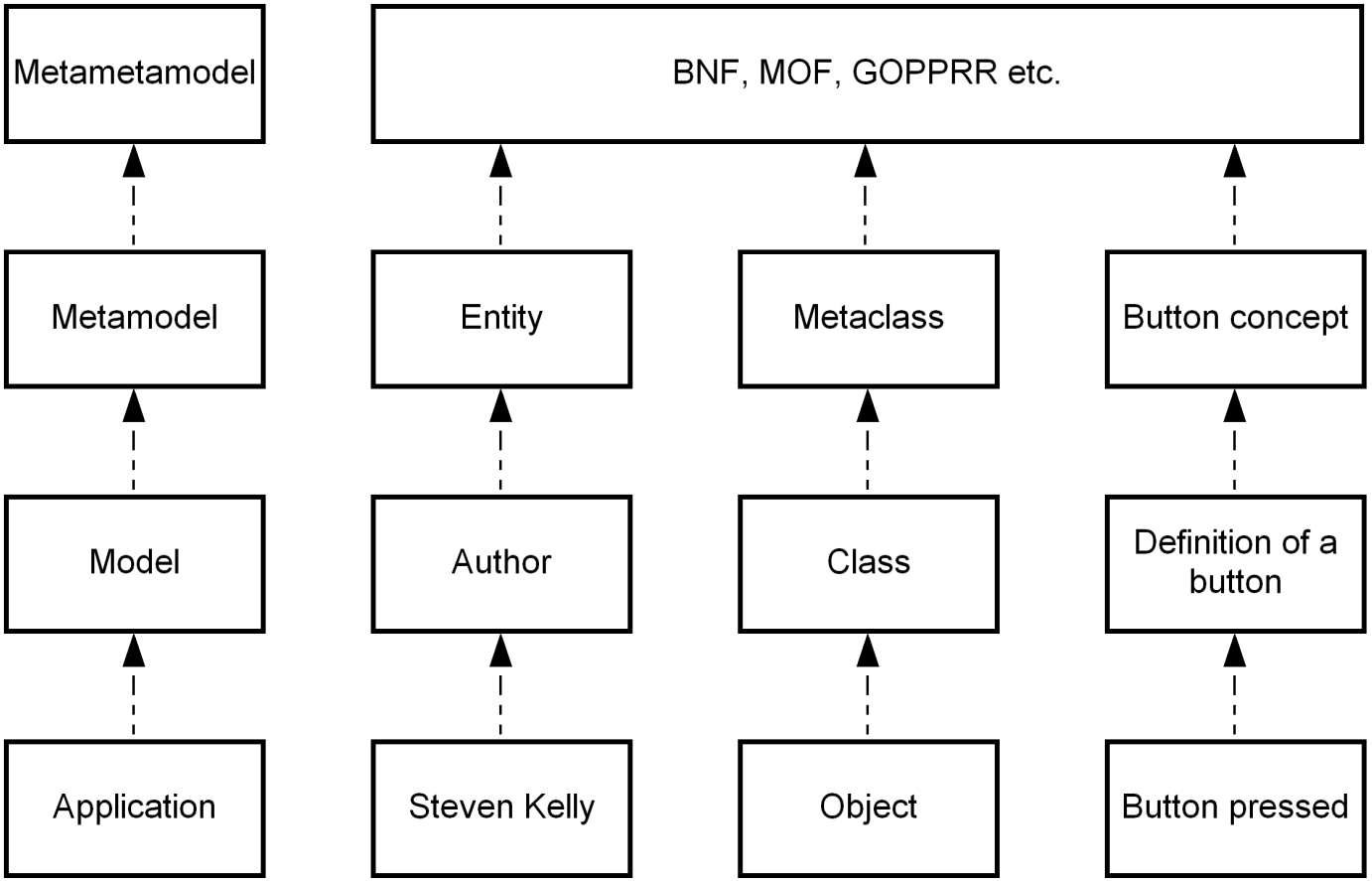
- Repetition:
  - # of product variants
  - # of similar features
  - # of developers
  - ”outsourcing” to domain experts



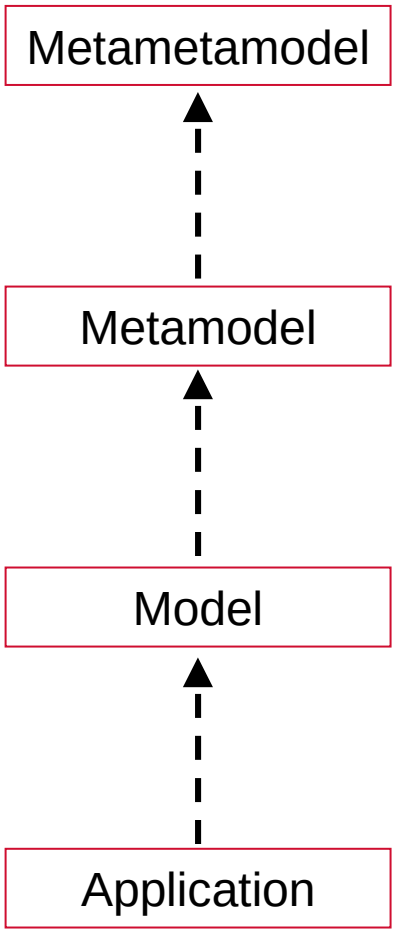
How to implement DSM



The four levels

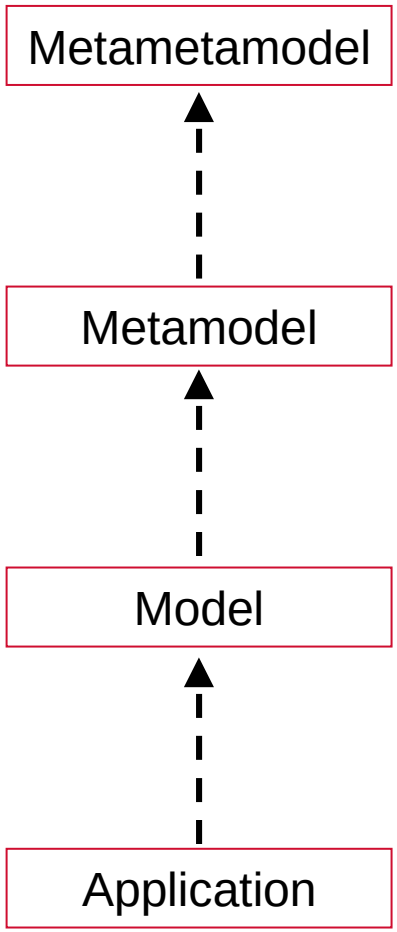


Living in the four levels: MMM?



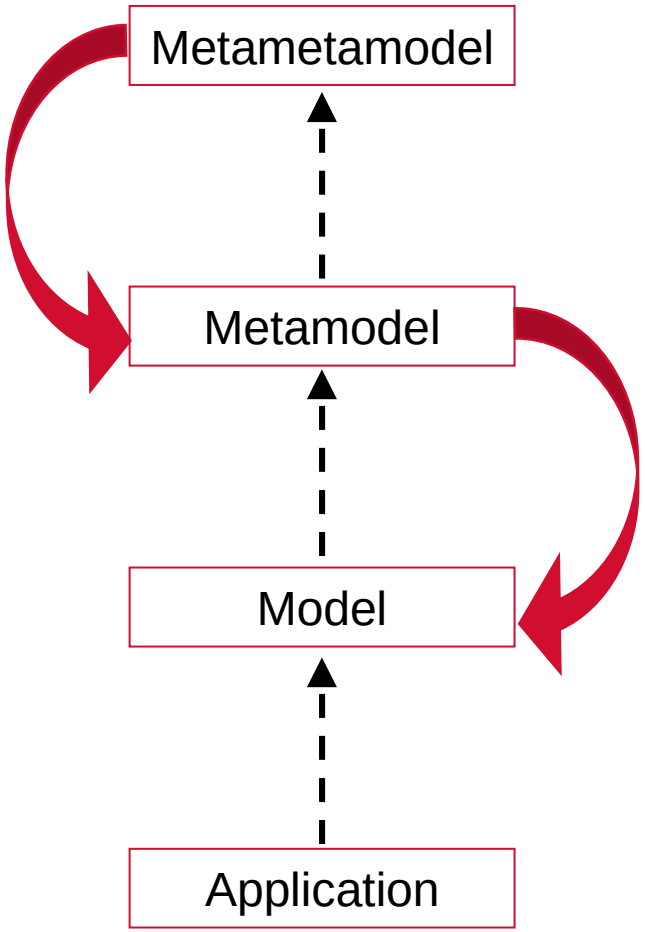
- Which kind of metamodeling language?
  - Expression power, easy to learn,…
  - Representation style?
    - text, graphical diagram, table, matrix,...
  - ER, OPRR, CoCoA, NIAM, GOPRR, MOF, EMOF, CMOF, Class diagram, GOPPRR, Domain model...
- Reuse existing metamodels and libraries vs. start from the scratch
- Definition of multiple languages and integration between them vs. multiple disconnected languages (and models)

Living in the four levels: DSMs



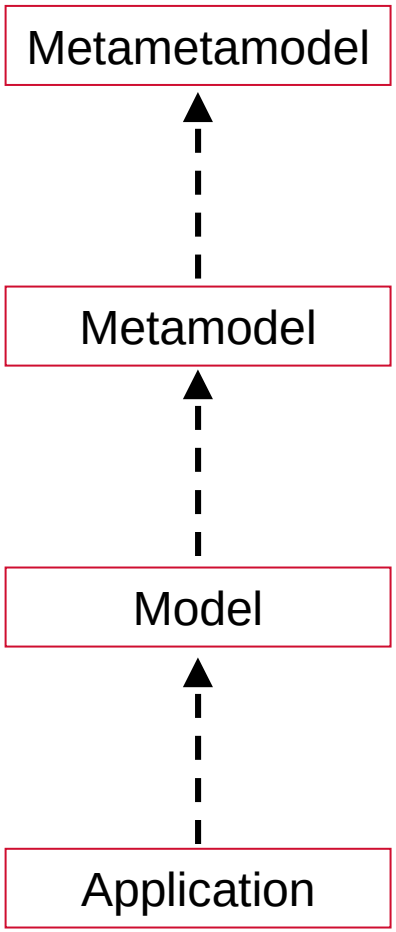
- How to aid and support language developer?
  - Defining and allocating domain concepts to language concepts
    - A model, a collection of models, an object, a property, relationship, role, port, link between these,...?
  - Testing the language
  - Integrating multiple languages
  - Reuse and links between models made
    - Multiple languages, users of models, definition steps
  - Sharing languages to developers
  - Updating languages (and models made)
- External DSLs vs embedded DSLs
- Graphical, matrix, table, vs textual

Transformations in DSM



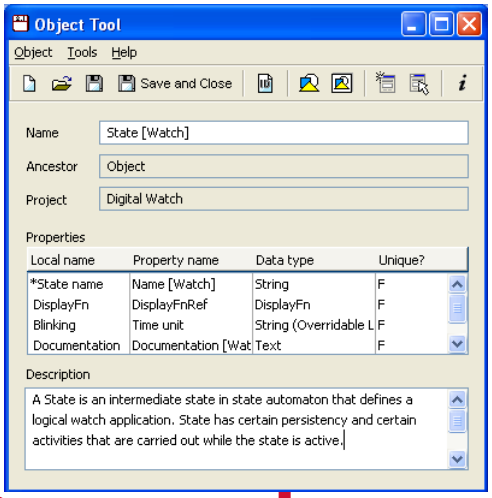
- Tools automate the key transformations on language definition and language use
- Tools should minimize resource use
- Tools should allow metamodel to change
  - And reflect changes to modelling tools and models already made

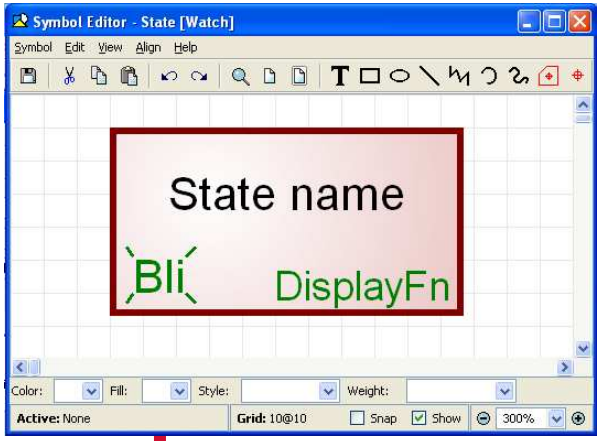
Living in the four levels: Tools



- 6 ways to get the tools for your need
  1. Write your own modeling tool from scratch
  2. Write your own modeling tool based on frameworks
  3. Metamodel, generate a modeling tool skeleton over a framework, add code
  4. Metamodel, generate the full modeling tool over a framework
  5. Metamodel, output configuration data for a generic modelling tool
  6. Integrated metamodeling and modeling tool

Defining DSM solution: Steps



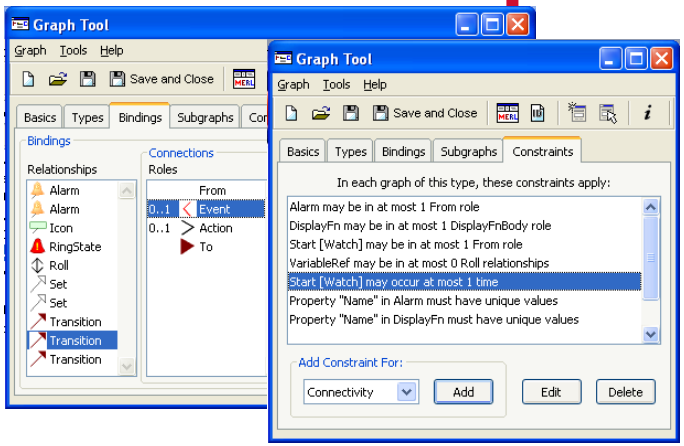


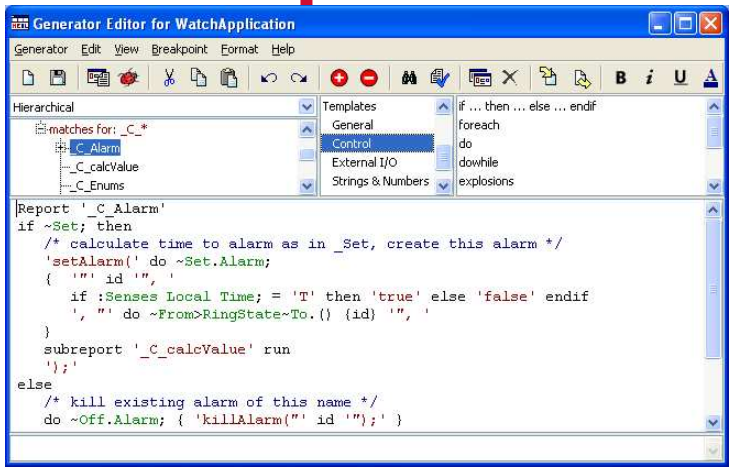
1 Concepts

2 Rules

3 Symbols

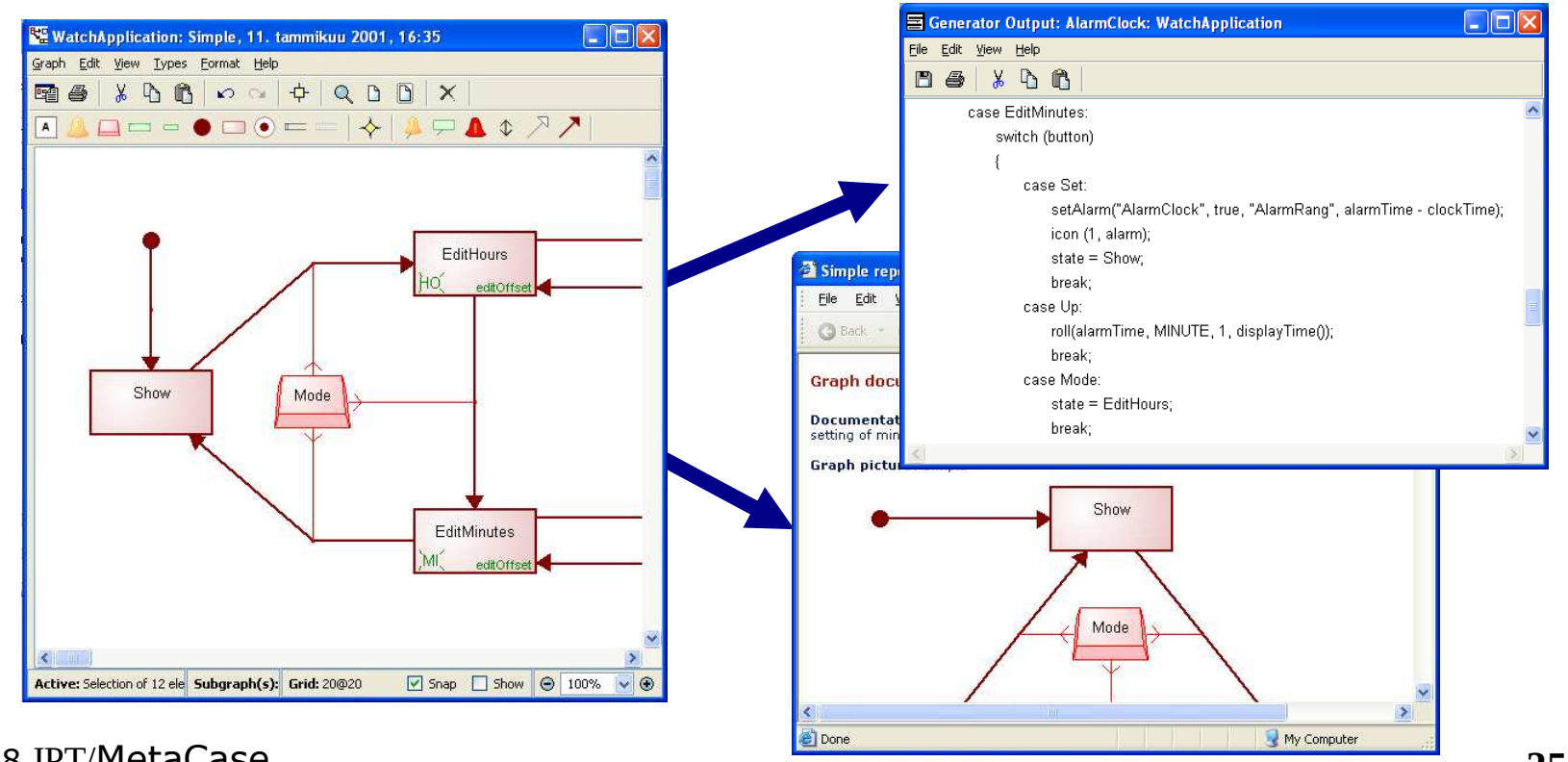
4 Generators





DSM solution in use

- Editors (diagram, matrix, table), browsers, generators, multi-user, multi-project, multi-platform environment
- Language and generator maintenance and sharing
  - language versions, updates models already made



Identifying language constructs [1/2]

- Use domain concepts directly as modeling constructs
  - already known and used
  - established semantics exist
  - natural to operate with
  - easy to understand and remember
  - requirements already expressed using them
  - architecture often operates on domain concepts
- Focus on expressing design space with the language
  - use parameters of variation space
  - keep the language simple
  - try to minimize the need for modeling
  - do not visualize product code!
    - better to ”forget” your current code
- Implement incrementally (test with models)!

Identifying language constructs [2/2]

- Enrich chosen computational models with domain-specific concepts and rules
  - look at the type of design languages already used
- Investigate various alternatives for describing domain with the chosen models, e.g.
  - model element(s)
  - element properties
  - certain collection of elements
  - relationships between elements
  - model organization structures
- Specify as a metamodel in some format
  - draft samples with pen & paper
  - document early as a metamodel
  - implement in some metamodel-based tool
  - test it with real models

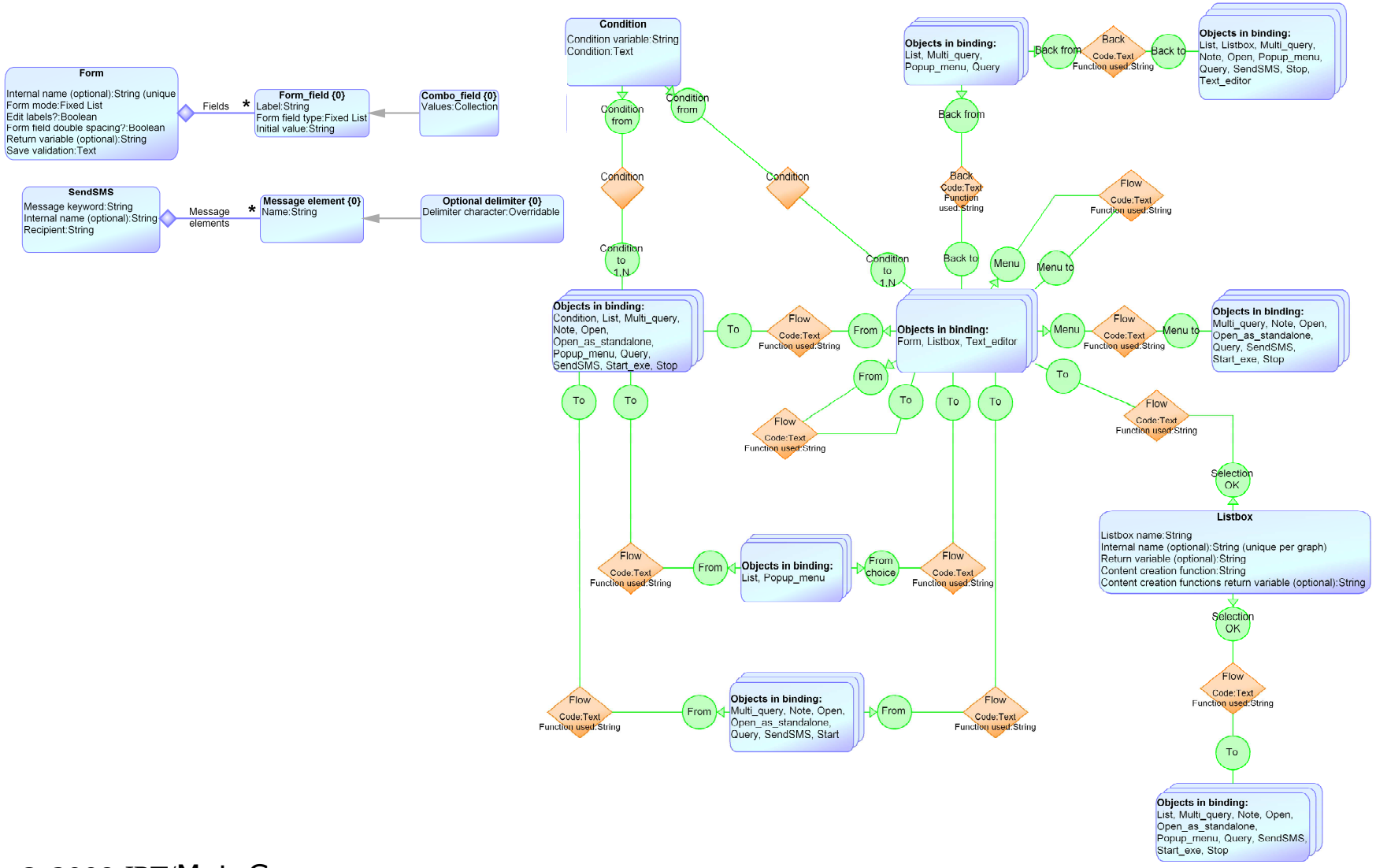
Rules in the languages

- The domain concepts of a modeling language are bound together with rules
- Putting the rules into the language allows
  - preventing creation of illegal models
  - informing about missing data
  - keeping models consistent
  - make code generation possible
- Prefer having rules as part of metamodel to having separate checker
  - Support early error prevention and provide guidance
  - But going overboard can hinder flow of modeler

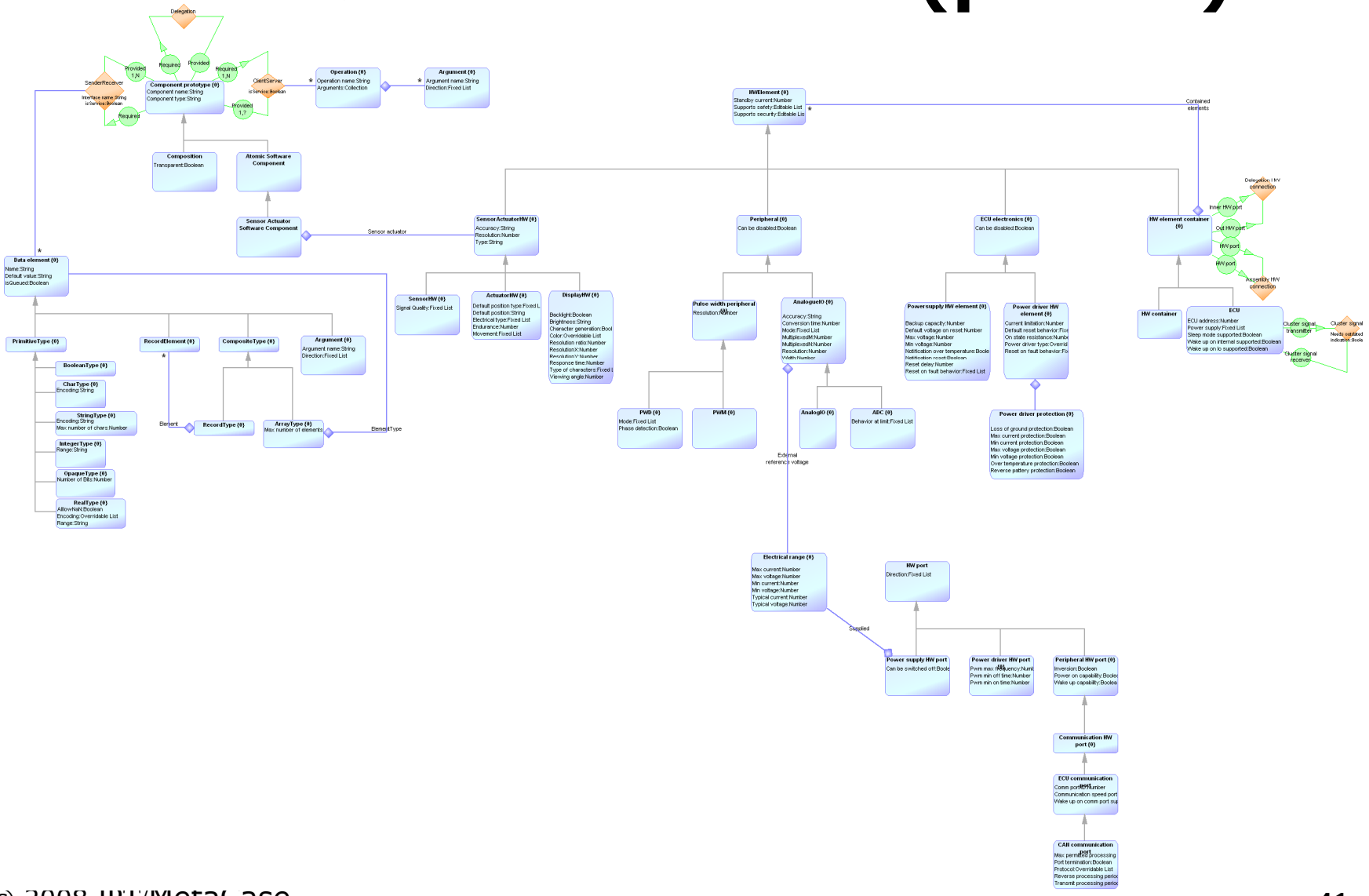
Defining notation

- Vital for acceptance and usability
- Symbols can vary from boxes to photorealism
  - Best to resemble closely the actual domain representation
  - Worst is having everything a box and special text to show the difference (cf. stereotypes)
  - Design information needs space: compromise
- Don’t create notation from scratch
  - Use known/existing elements (and, or, start, stop etc)
- Hint: ask users to define the notation
  - It is much easier to introduce their own language than something you created
  - Remember also model readers
    - managers, test engineers, customers, deployment, configuration, packaging and even sales

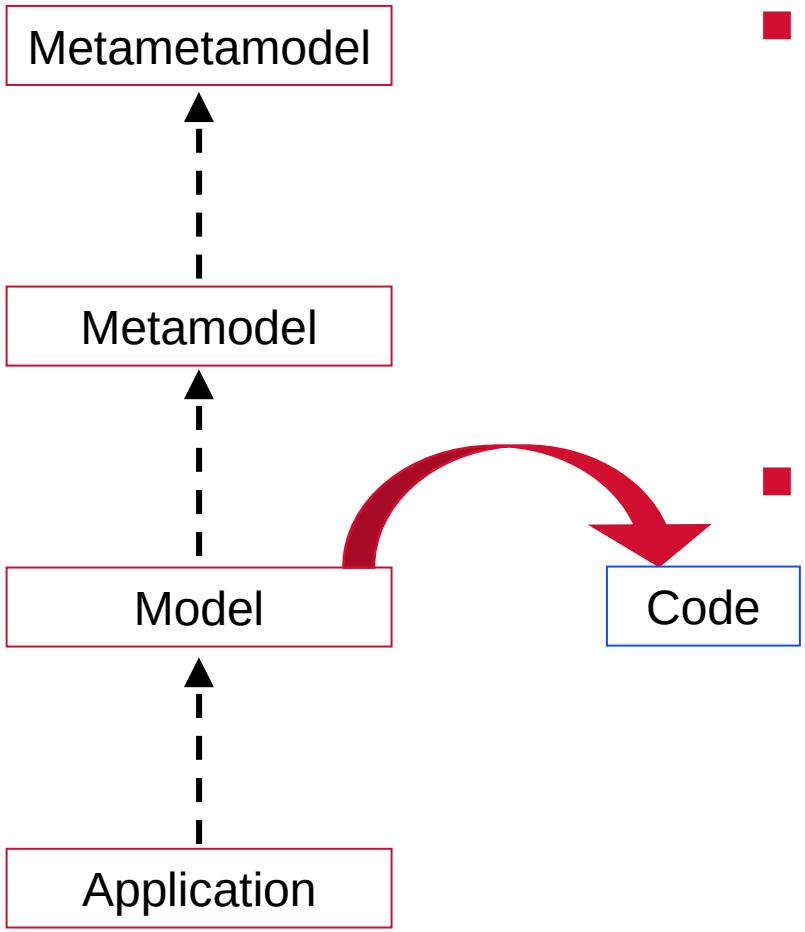
Metamodel: Mobile phone app



Metamodel: AUTOSAR (partial)

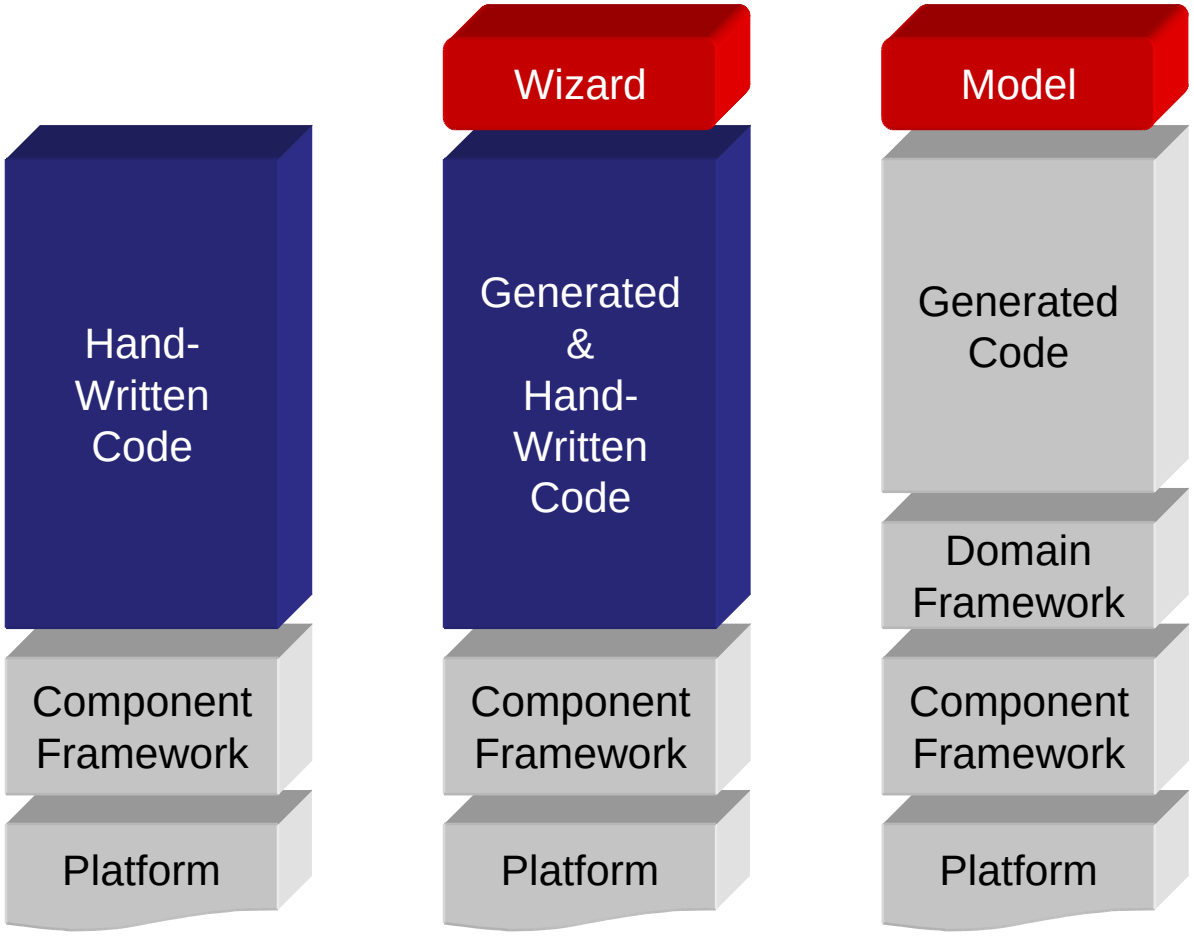


Transformations in DSM

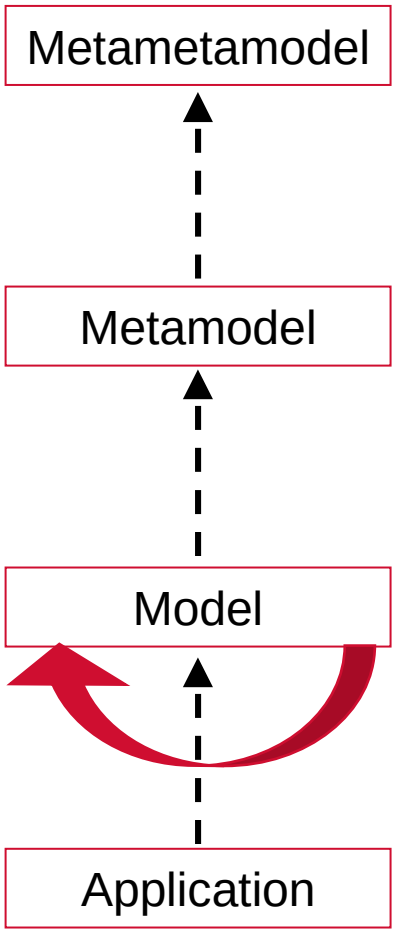


- Code generation principles
  - Generate always full code from modelers perspective
  - Never touch the generated code
  - Keep generated and manually written code separete
    - Avoid generation of sections of files (protected blogs)
- Single source, multiple outputs/streams
  - Configuration
  - Testing and analysis
  - Automated build → automating compile and execution
  - Metrics
  - Simulation & prototypes
  - Help text and user guides
  - Documentation and review

Raise abstraction with domain frameworks

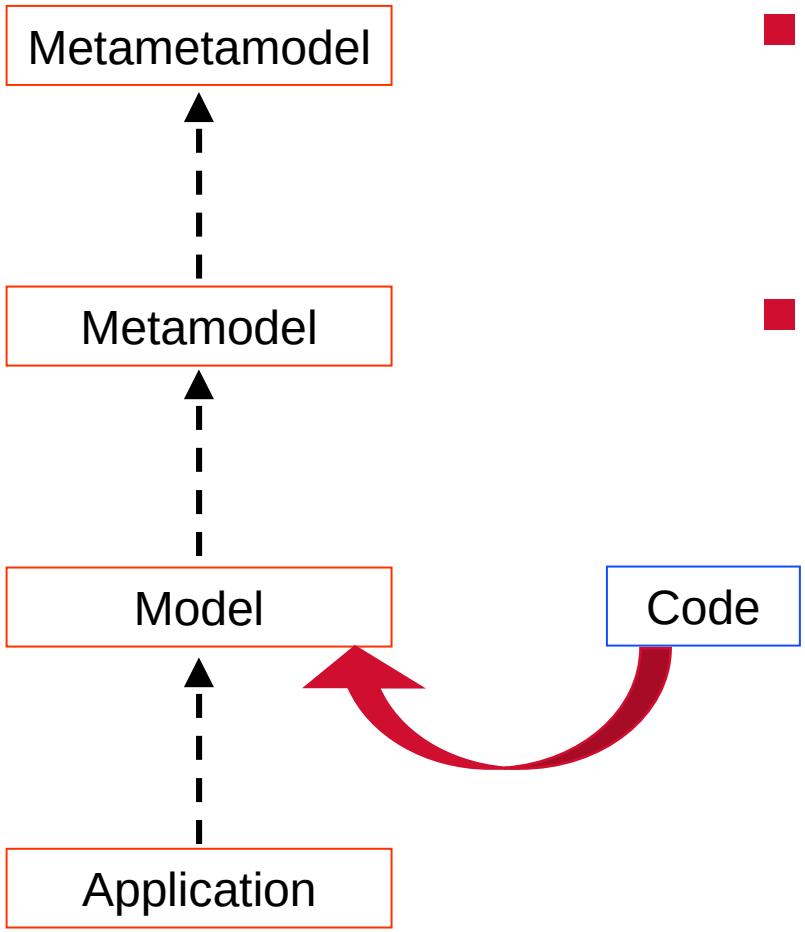


Transformations in DSM



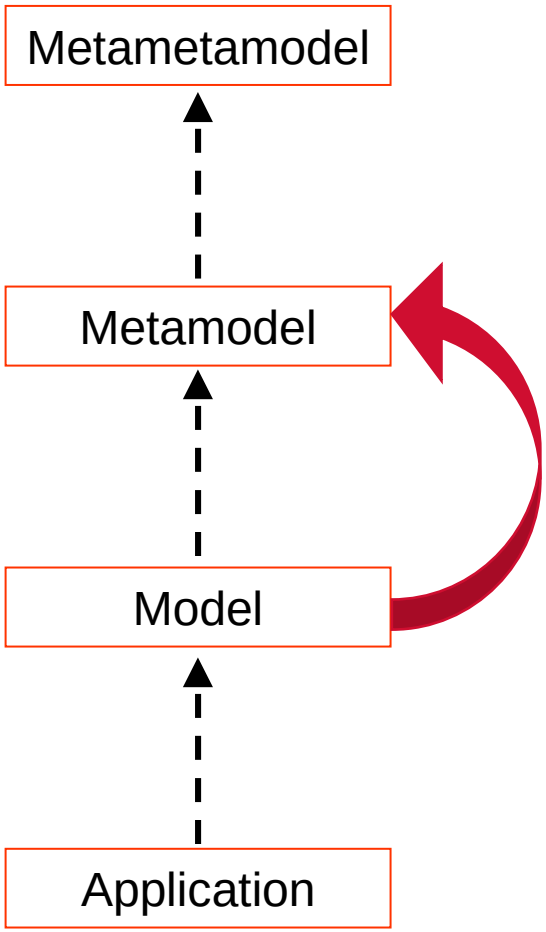
- Model-to-model transformation is usually ”a bad thing”:
  - Creates copies of the same data
  - Running transformation again after model has been manually edited difficult
- Better to extend the modeling language
- Acceptable is some cases
  - Models based on a subset of another language
  - Generator exists for an other metamodel

Transformations in DSM



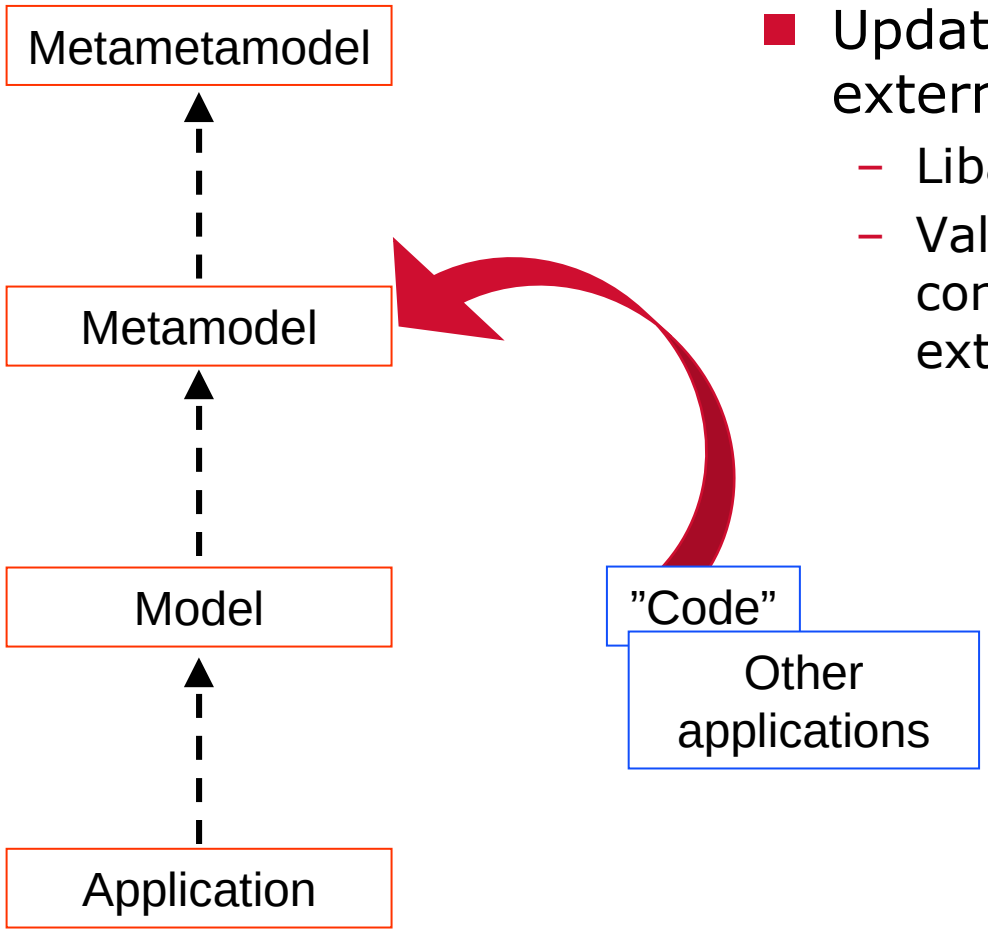
- Import to models (reverse)
  - Data definitions
  - Interfaces
  - Message types etc.
- Imported data is tied to specific types in the modeling language
  - E.g. Specific kind of function

What kind of transformations



- Create metamodels based on models
  - Graphical metamodeling

What kind of transformations



- Update language based on external data
  - Libaries
  - Values to be selected contextually based on external data

Research topics

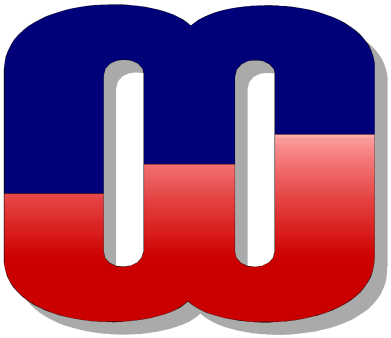
- Emprical studies needed more!
  - Development time and resource use
  - Benefits
  - ROI
  - Where and when to use DSM
    - Application areas, Types of project, Existing styles and practices, Before, during or after the Big New Architecture?
- How do you design a domain-specific language?
  - Identifying variable aspects of the domain?
  - Gradual evolution or big upfront design?
  - Testing in terms of the DSML and its abstractions
- Choosing a type of language
  - Graphical, text, matrix, table, form, interactive
- Evolution of languages in accordance with a domain
  - Maintaining compatibility as the language evolves
- Tooling related
  - (Meta)model versioning principles & tools
  - Graphical DIFF

Summary

- We can still continue to raise the level of abstraction
- Domain-Specific approach seems a viable approach
- We need to study (and teach) how to:
  - specify languages
  - implement generators
  - create frameworks for automation
  - seek right abstractions for automation
- There is a growing interest in DSM
  - research
  - industry

Thank you!

Question and comments?



Contact: jpt@metacase.com

MetaCase
Ylistönmäentie 31
FI-40500 Jyväskylä, Finland
Phone +358 14 4451 400
Fax +358 14 4451 405

References

- EADS case study, www.metacase.com/papers/MetaEdit_in_EADS.pdf
- Kelly, S., Tolvanen, J-P., Domain-Specific Modeling, Wiley 2008
- Kieburtz, R. et al., "A Software Engineering Experiment in Software Component Generation," Proceedings of 18th International Conference on Software Engineering, Berlin, IEEE Computer Society Press, March, 1996.
- Nokia case study, www.metacase.com/papers/MetaEdit_in_Nokia.pdf
- Safa, L., The Making Of User-Interface Designer, A Proprietary DSM Tool, Procs of 7th OOPSLA workshop on Domain-Specific Modeling, 2007
- DSMForum, www.dsmforum.org