Defining Domain-Specific Modelling Languages

22 April 2009

Juha-Pekka Tolvanen, Ph.D.
MetaCase

Outline

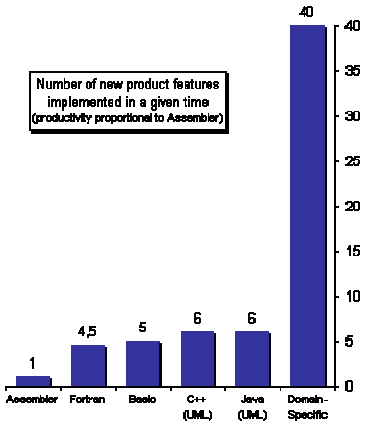
- Introduction
- The vision of Model Driven Development
- Examples and case studies
- Architecture for defining and using MDD
- Implementing MDD
- Summary

Outline

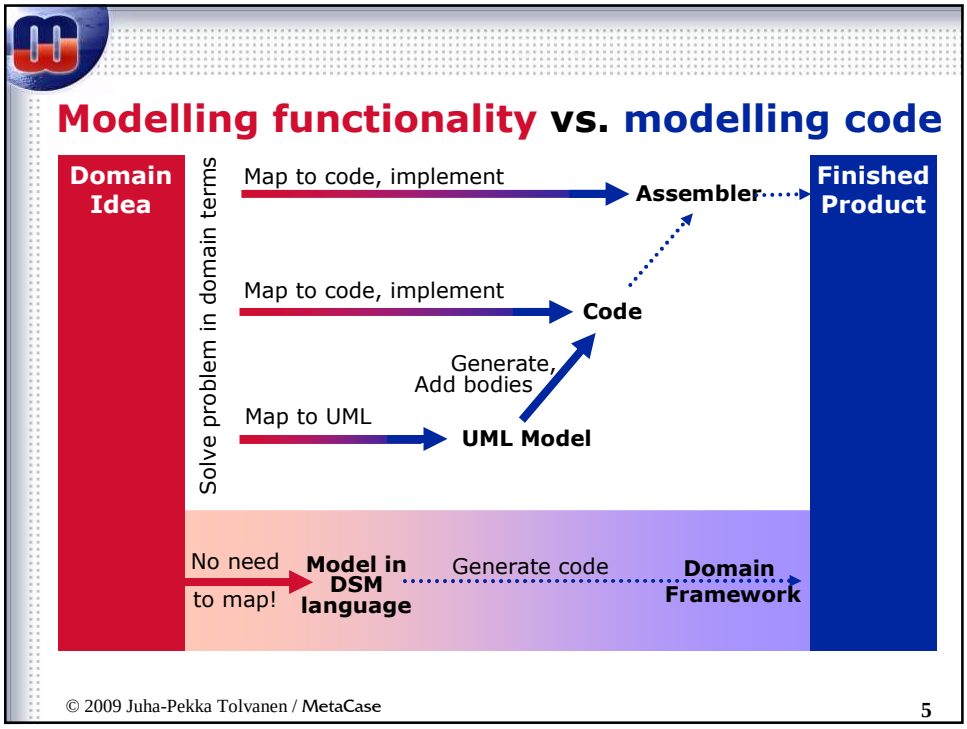
- Introduction
- The vision of Model Driven Development
  - DSM: Domain-Specific Modelling
  - SF: Software Factories
  - MDA: Model Driven Architecture

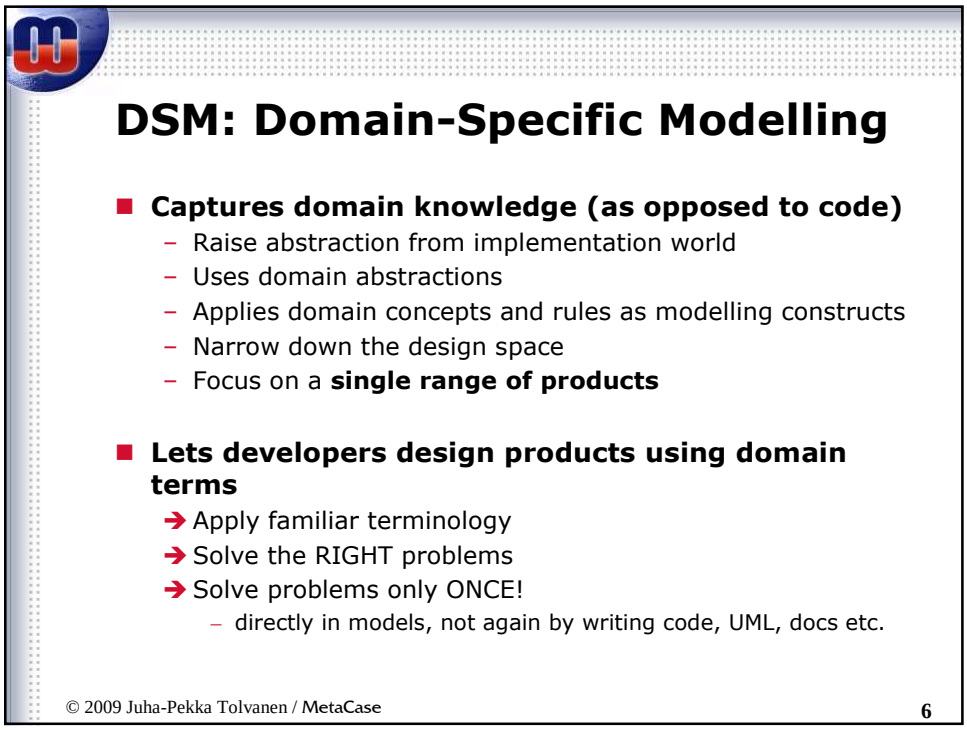
How has productivity improved?

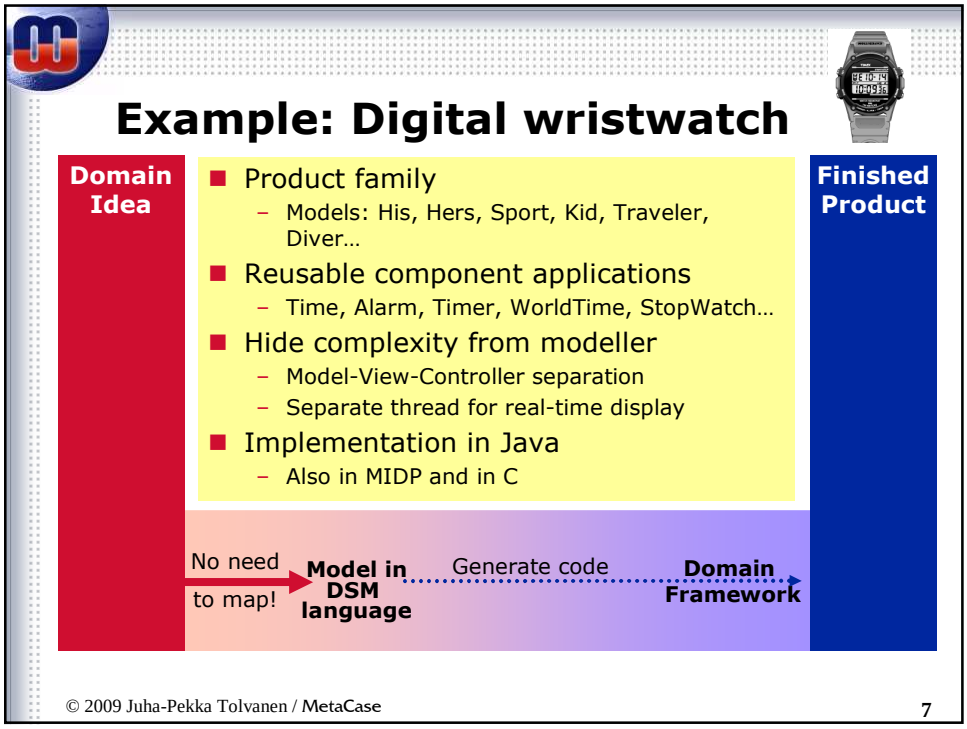
- "The entire history of software engineering is that of the rise in levels of abstraction"
- New programming languages have not increased productivity
- UML and visualization of code have not increased productivity
- Abstraction of development can be raised above current level...
- ... and still generate full production code (and ignore it!)

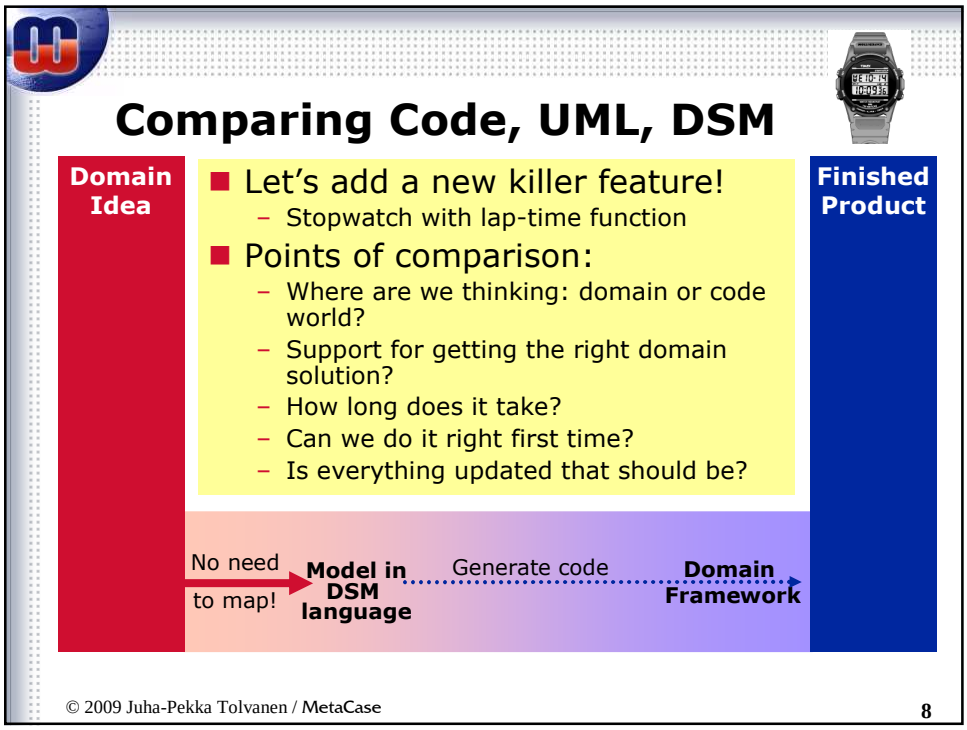


*Software Productivity Research & Capers Jones, 2002

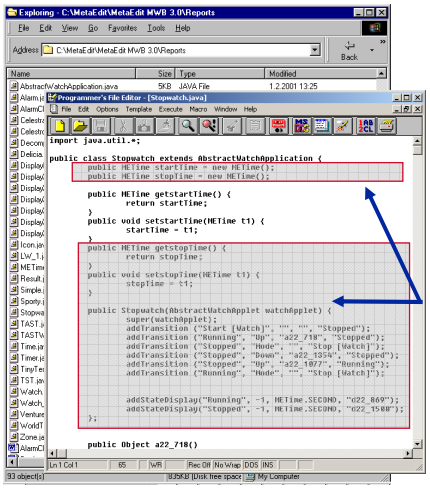

Modelling functionality vs. modelling code
Domain Idea
Solve problem in domain terms
Map to code, implement
Assembler
Finished Product
Map to code, implement
Code
Generate, Add bodies
Map to UML
UML Model
No need to map!
Model in DSM language
Generate code
Domain Framework
© 2009 Juha-Pekka Tolvanen / MetaCase
5

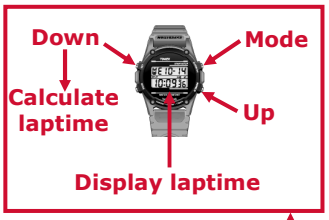

DSM: Domain-Specific Modelling
Captures domain knowledge (as opposed to code)
– Raise abstraction from implementation world
– Uses domain abstractions
– Applies domain concepts and rules as modelling constructs
– Narrow down the design space
– Focus on a single range of products
Lets developers design products using domain terms
➔ Apply familiar terminology
➔ Solve the RIGHT problems
➔ Solve problems only ONCE!
– directly in models, not again by writing code, UML, docs etc.
© 2009 Juha-Pekka Tolvanen / MetaCase
6


Example: Digital wristwatch
Domain Idea
Product family
Models: His, Hers, Sport, Kid, Traveler, Diver…
Reusable component applications
Time, Alarm, Timer, WorldTime, StopWatch…
Hide complexity from modeller
Model-View-Controller separation
Separate thread for real-time display
Implementation in Java
Also in MIDP and in C
Finished Product
No need to map!
Model in DSM language
Generate code
Domain Framework
© 2009 Juha-Pekka Tolvanen / MetaCase
7


Comparing Code, UML, DSM
Domain Idea
Let's add a new killer feature!
Stopwatch with lap-time function
Points of comparison:
Where are we thinking: domain or code world?
Support for getting the right domain solution?
How long does it take?
Can we do it right first time?
Is everything updated that should be?
Finished Product
No need to map!
Model in DSM language
Generate code
Domain Framework
© 2009 Juha-Pekka Tolvanen / MetaCase
8

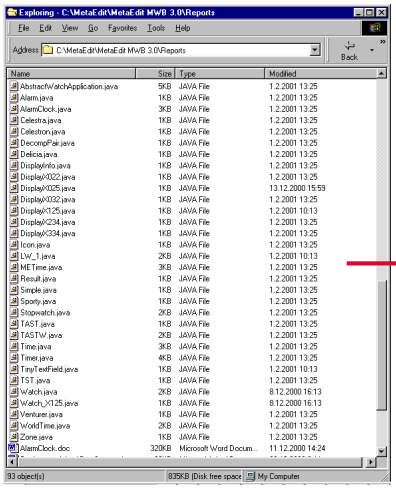
Code-based approach

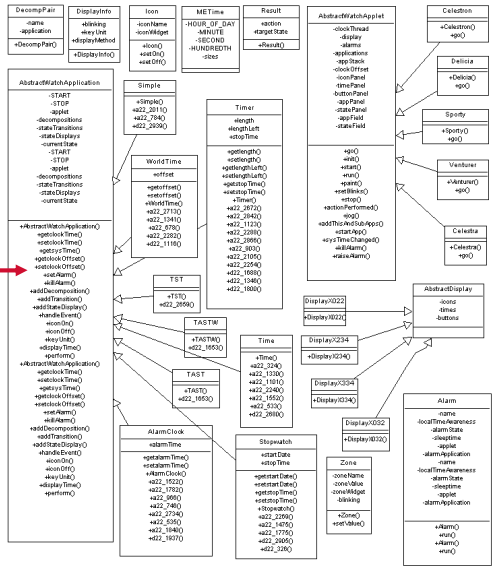




1. Read the documents
2. Find the solution
3. Find the relevant code
4. Change the right code
5. Document the code change
6. Test the changes
7. Document the solution

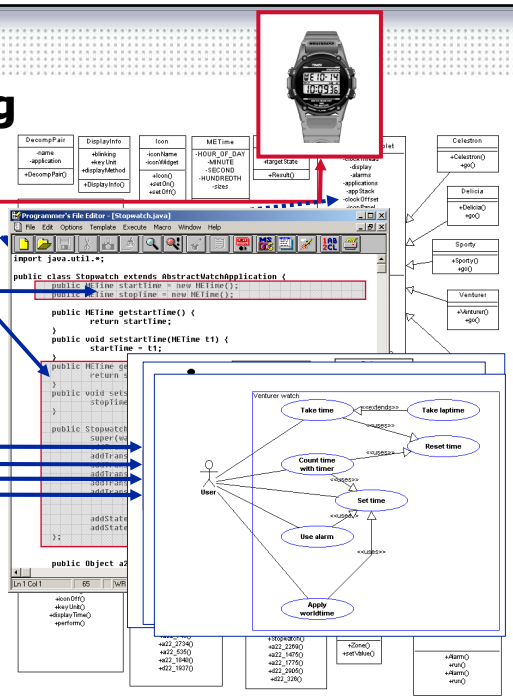
Code visualization approach





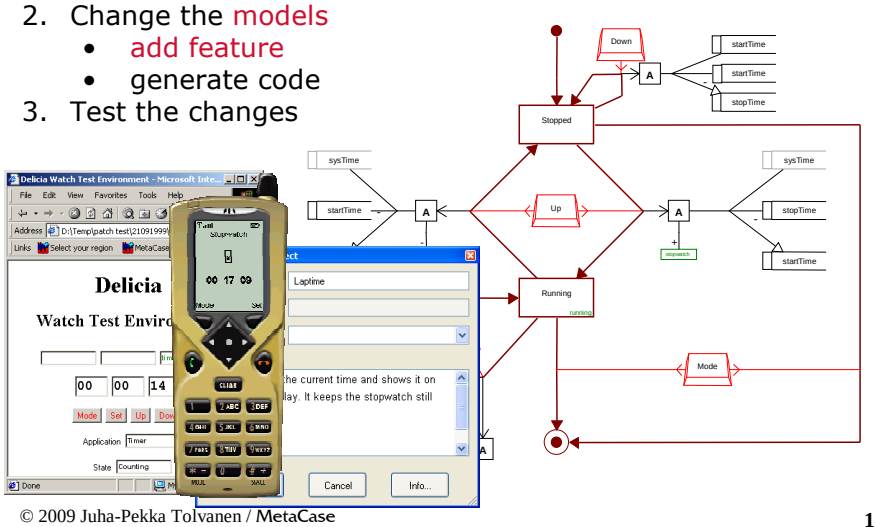
UML Modelling

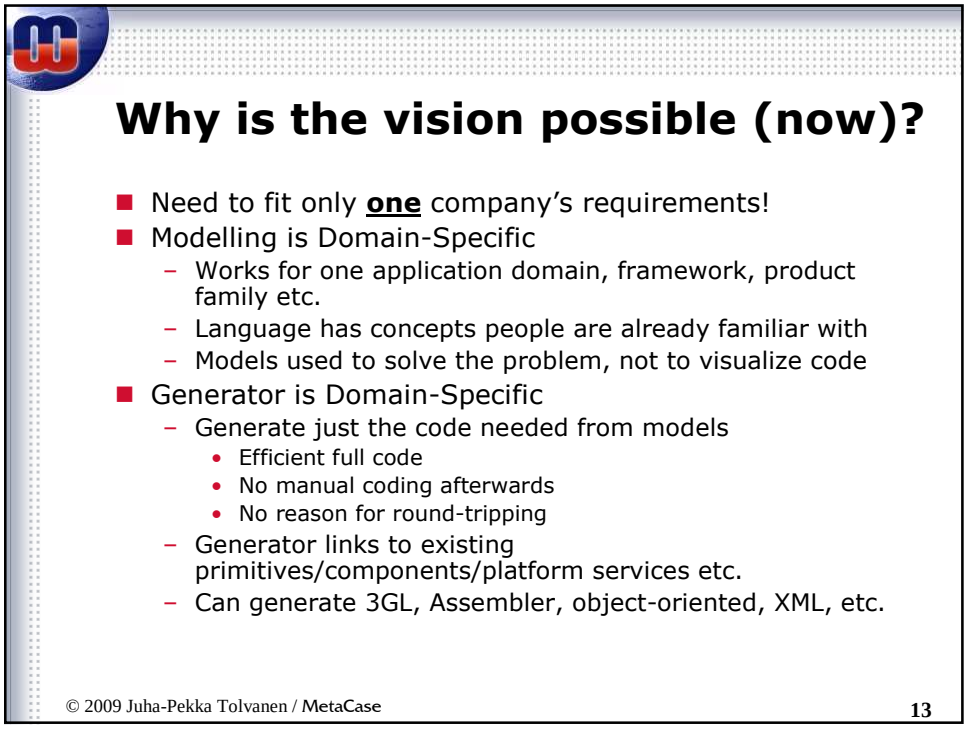
1. Read the documents
2. Find the solution
3. Find the relevant models
4. Change the right code and models
5. Document the code and model changes
6. Update models (Use cases, Class models, Message sequences models, State models etc.)
7. Test the changes
8. Document the solution



Domain-Specific Modelling

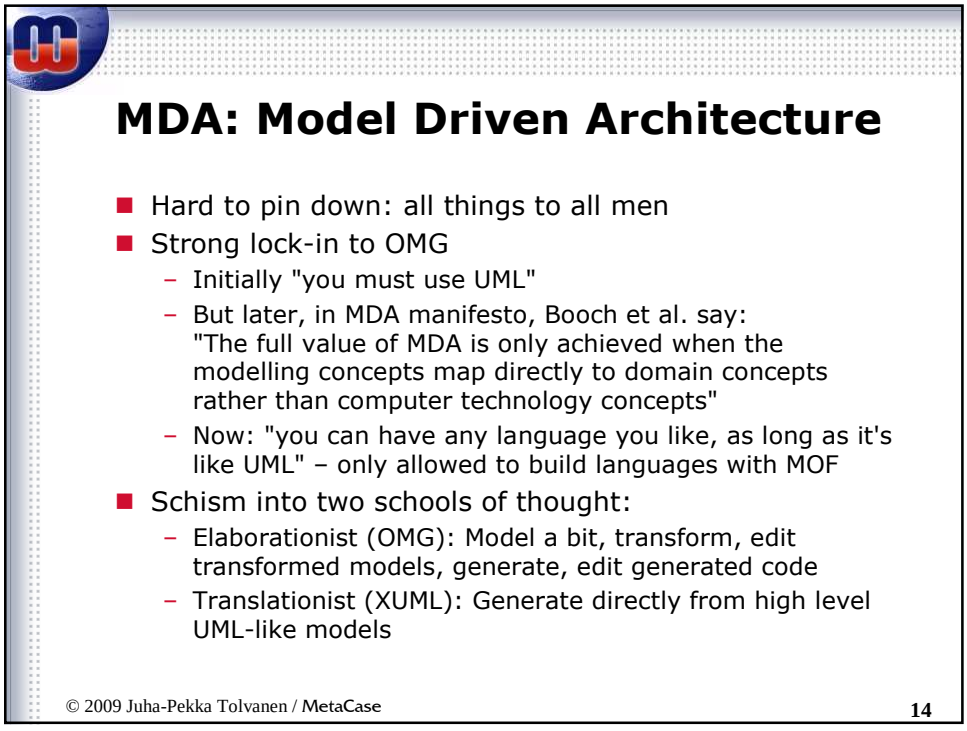
1. Find the relevant models
2. Change the models
   - add feature
   - generate code
3. Test the changes





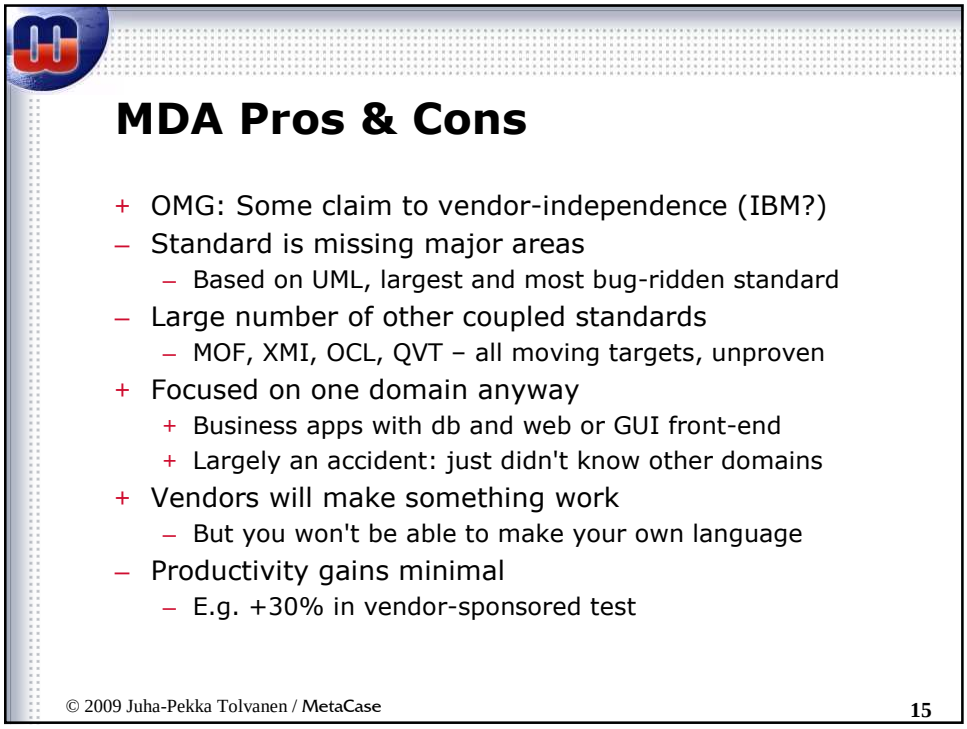
Why is the vision possible (now)?

- Need to fit only **one** company's requirements!
- Modelling is Domain-Specific
  - Works for one application domain, framework, product family etc.
  - Language has concepts people are already familiar with
  - Models used to solve the problem, not to visualize code
- Generator is Domain-Specific
  - Generate just the code needed from models
    - Efficient full code
    - No manual coding afterwards
    - No reason for round-tripping
  - Generator links to existing primitives/components/platform services etc.
  - Can generate 3GL, Assembler, object-oriented, XML, etc.



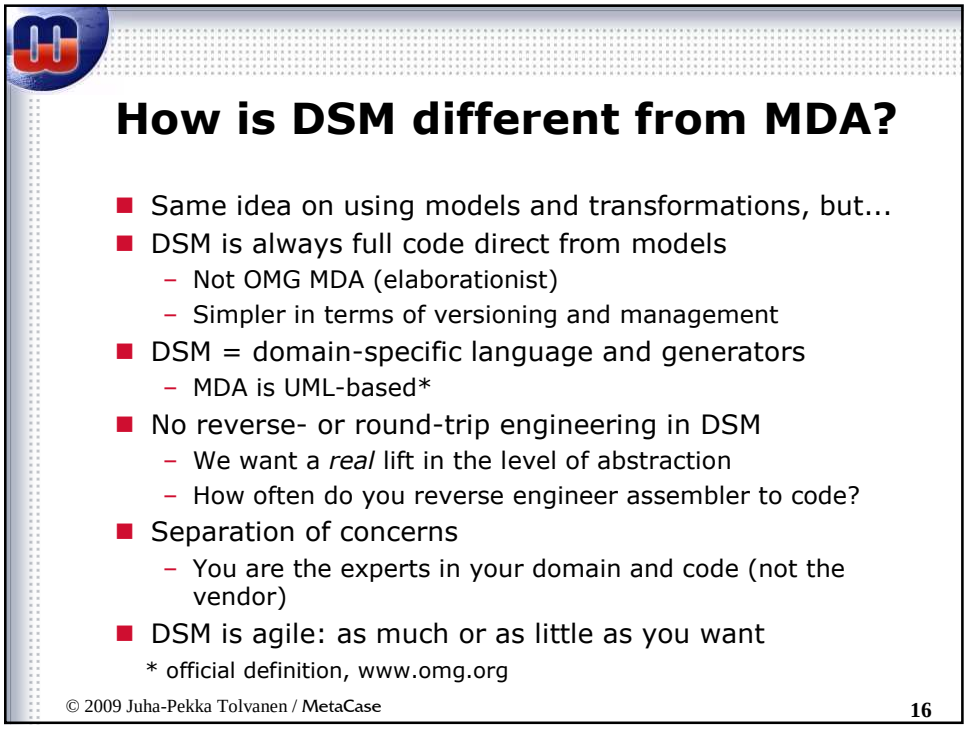
MDA: Model Driven Architecture

- Hard to pin down: all things to all men
- Strong lock-in to OMG
  - Initially "you must use UML"
  - But later, in MDA manifesto, Booch et al. say: "The full value of MDA is only achieved when the modelling concepts map directly to domain concepts rather than computer technology concepts"
  - Now: "you can have any language you like, as long as it's like UML" – only allowed to build languages with MOF
- Schism into two schools of thought:
  - Elaborationist (OMG): Model a bit, transform, edit transformed models, generate, edit generated code
  - Translationist (XUML): Generate directly from high level UML-like models



MDA Pros & Cons

+ OMG: Some claim to vendor-independence (IBM?)
– Standard is missing major areas
  – Based on UML, largest and most bug-ridden standard
– Large number of other coupled standards
  – MOF, XMI, OCL, QVT – all moving targets, unproven
+ Focused on one domain anyway
  + Business apps with db and web or GUI front-end
  + Largely an accident: just didn't know other domains
+ Vendors will make something work
  – But you won't be able to make your own language
– Productivity gains minimal
  – E.g. +30% in vendor-sponsored test



How is DSM different from MDA?

- Same idea on using models and transformations, but...
- DSM is always full code direct from models
  - Not OMG MDA (elaborationist)
  - Simpler in terms of versioning and management
- DSM = domain-specific language and generators
  - MDA is UML-based*
- No reverse- or round-trip engineering in DSM
  - We want a *real* lift in the level of abstraction
  - How often do you reverse engineer assembler to code?
- Separation of concerns
  - You are the experts in your domain and code (not the vendor)
- DSM is agile: as much or as little as you want

* official definition, www.omg.org

SF: Software Factories

- Strongly Microsoft-oriented
  - But main figures from outside Microsoft:
    Greenfield: Rational, Short: TI, Cook: IBM, Kent: Kent
- Grand Unified(?) Theory
  - 666 pages
  - Patterns, AOP, reuse, platforms, components, services
  - DSLs, generators, frameworks
- Vision varies under commercial pressure
  - First own languages, then wizards, now MS designers
- Focus on MS & partners building and selling DSLs
  - ISV sells same DSM solution to many companies
    - Less domain-specific, companies have less control
  - Offsets the "massive effort"* of using their tools
    - *Quote from Prashant Sridharan, lead product manager

SF Pros & Cons

+ Microsoft: Massive resources, will get it made:

Windows announced	1.0 released	2.0 released	3.0 released	3.1 released
1983	1985	1987	1990	1992

– Microsoft: too many cooks and agendas
  – Building meta-tools requires strong leadership, focus
  – Will the project be continued (remember Rose in VS?)
– MS team lacked real-world experience in DSM
  – Will need a rewrite, but will it happen?
+ Basic ideas are sound
  + Book mostly better than later marketing

DSM Pros & Cons

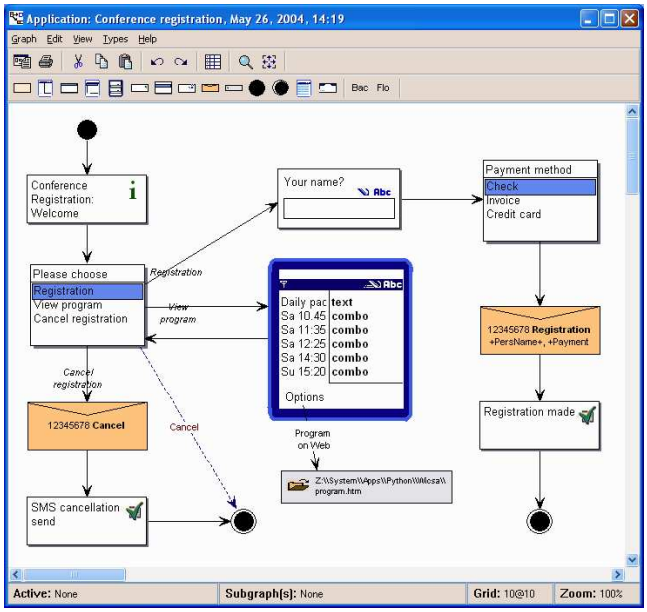
- \+ Fundamental productivity and quality improvements
  - \+ 300% faster in scientific study
  - \+ 500-1000% reported by companies
  - \+ 50% less errors in scientific study
- \+ Gives full control to the company
  - \+ Their experienced developers are in the driver's seat
- – Requires expertise and resources from the company
- \+ Minimal vendor lock
  - \+ Metamodel-driven tools are open
  - \+ You can translate & transform models to other tools and formats
- – Most tools not mature or "industrial strength"
  - – Do not scale to multiple developers or models
  - – Do not handle evolution and maintenance

Outline

- Introduction
- The vision of Model Driven Development
- Examples and case studies
  - Smartphone applications
  - eCommerce marketplace

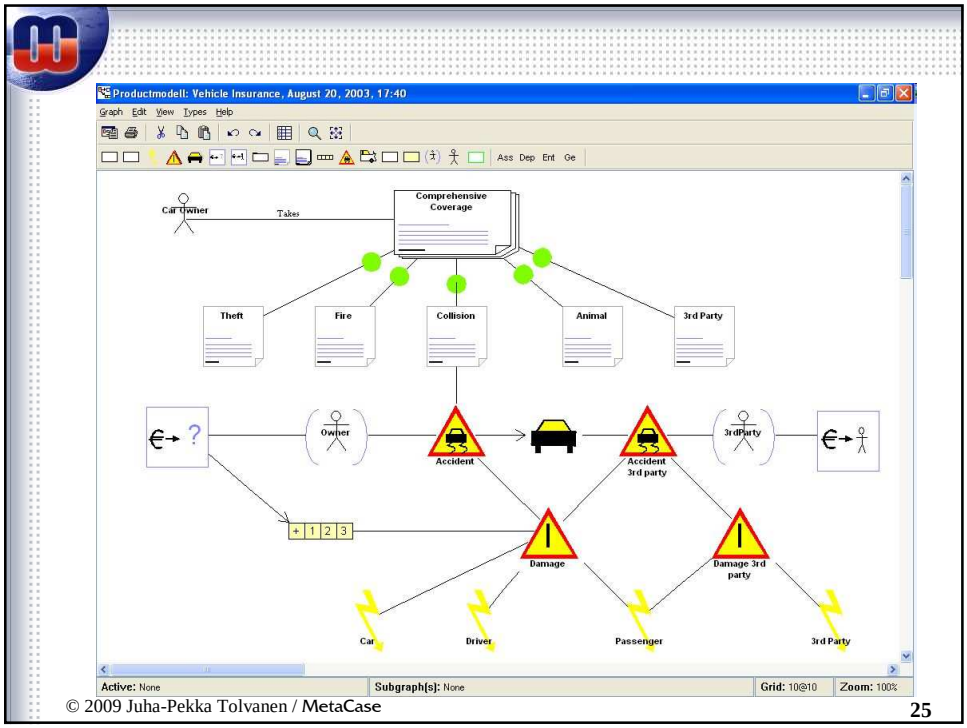
Enterprise apps in smartphones

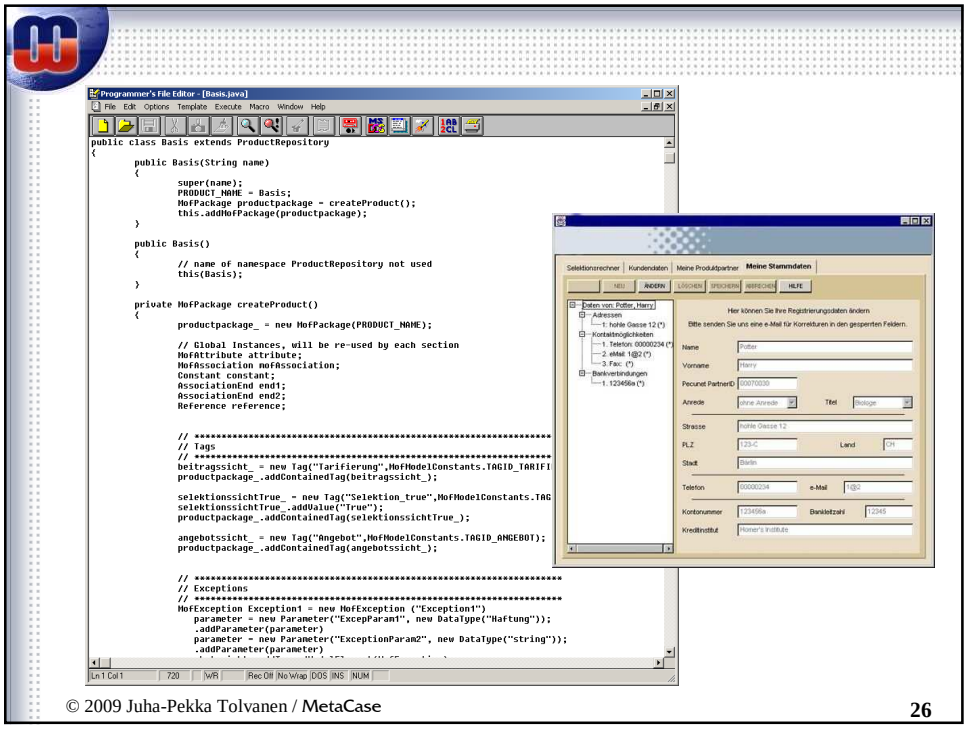
- Symbian/Series 60 for enterprise application development
- Platform provides basic services
- Modelling language to define application logic using basic widgets and services
- Code generator produces 100% of implementation
- Complete chain from model to running app

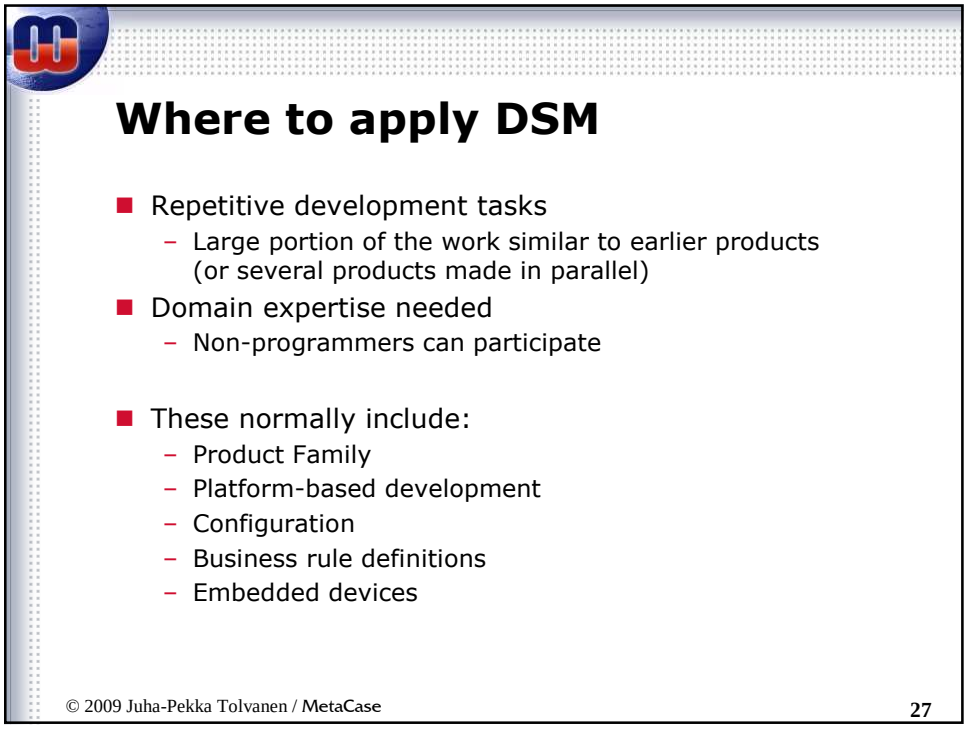


Insurance products & eCommerce

- 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 writing directly Java after first 30 products = DSM at least 3 times faster, fewer errors

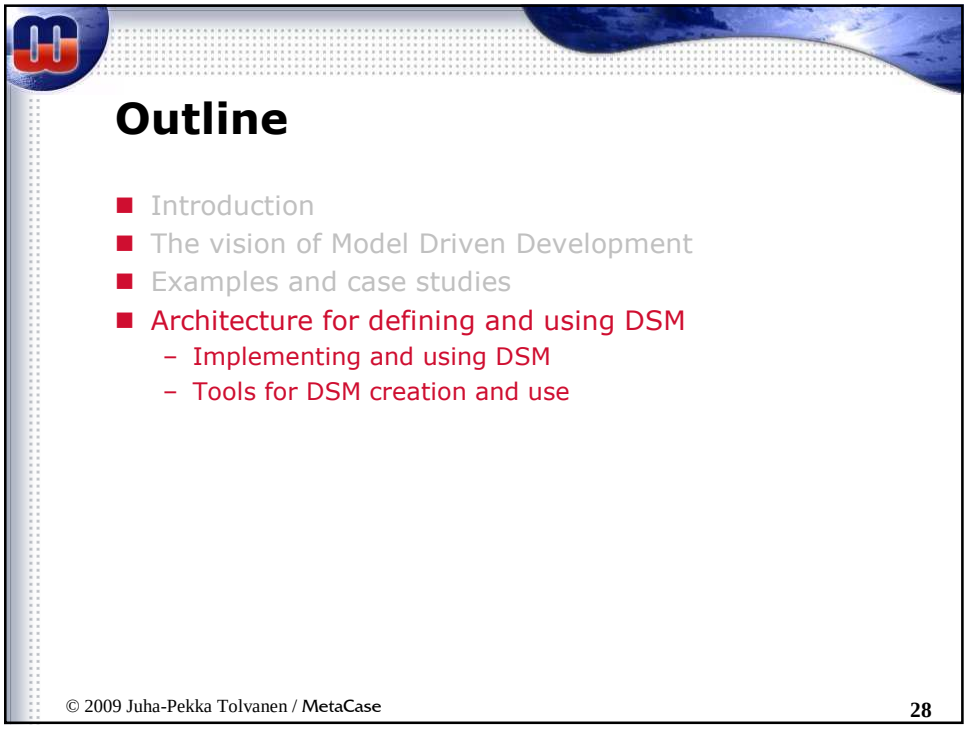






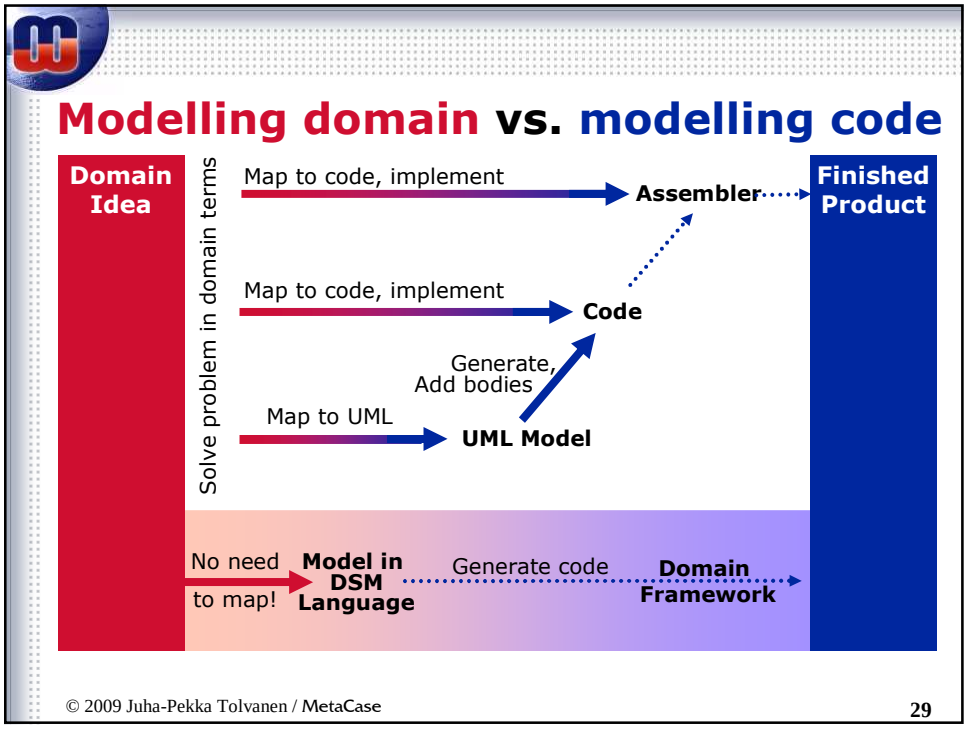
Where to apply DSM

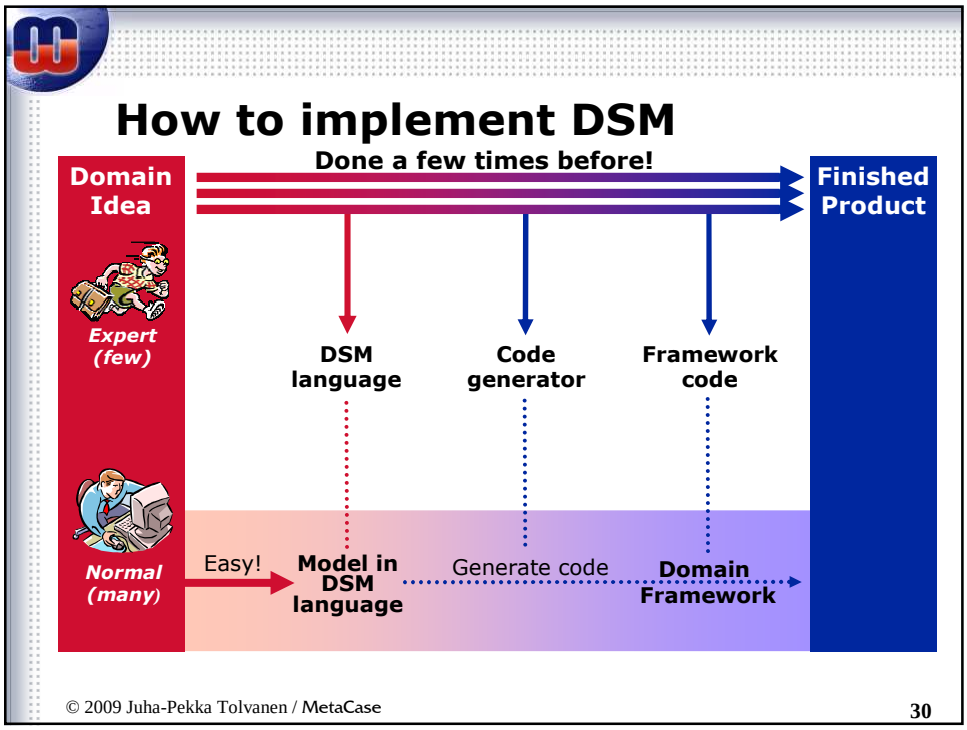
- Repetitive development tasks
  - Large portion of the work similar to earlier products (or several products made in parallel)
- Domain expertise needed
  - Non-programmers can participate
- These normally include:
  - Product Family
  - Platform-based development
  - Configuration
  - Business rule definitions
  - Embedded devices



Outline

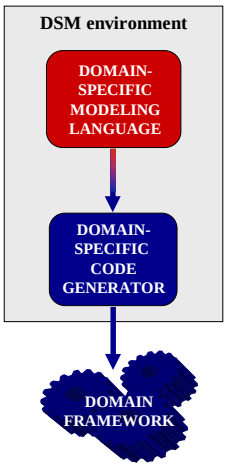
- Introduction
- The vision of Model Driven Development
- Examples and case studies
- Architecture for defining and using DSM
  - Implementing and using DSM
  - Tools for DSM creation and use


Modelling domain vs. modelling code
Domain Idea
Solve problem in domain terms
Map to code, implement
Assembler
Finished Product
Map to code, implement
Code
Generate, Add bodies
Map to UML
UML Model
No need to map!
Model in DSM Language
Generate code
Domain Framework
© 2009 Juha-Pekka Tolvanen / MetaCase
29


How to implement DSM
Domain Idea
Done a few times before!
Finished Product
Expert (few)
DSM language
Code generator
Framework code
Normal (many)
Easy!
Model in DSM language
Generate code
Domain Framework
© 2009 Juha-Pekka Tolvanen / MetaCase
30

DSM environment

- Three things are required for a complete DSM environment:
  1. Domain-specific modelling language
     - Metamodel (of the language) maps to problem domain (not to coding concepts)
     - Metamodel bounds allowed design space
  2. Code generator(s)
     - Generators read models to produce code
     - Provide variation for output formats
  3. Domain framework
     - Include common aspects used as primitives/components/platform services
     - Called by the generated code



Defining a DSM solution: steps

1. Identify abstractions
   - Concepts and how they work together
2. Specify the metamodel
   - Language concepts and their rules
3. Create the notation
   - Representation of models
4. Define the generators
   - Various outputs and analysis of the models

- Apply and refine existing components and libraries
- The process is iterative: try solution with examples
  - Define part of the metamodel, model with it, define generator, extend the metamodel, model some more, ...

Tools for DSM creation and use

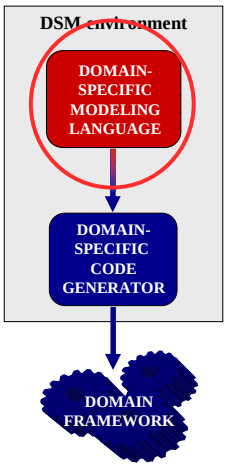
- 6 ways to get the tools we need for DSM
  1. Write own modelling tool from scratch
  2. Write own modelling tool based on frameworks
  3. Metamodel, generate modelling tool skeleton, add code
  4. Metamodel, generate full modelling tool over a framework
  5. Metamodel, output configuration for generic modelling tool
  6. Integrated modelling and metamodeling environment
- Good tools minimize resource use (few man-weeks)
  - creating modelling tools and generators data-like, not code
  - guide in DSM creation
  - allow you to test DSM throughout domain design process
- Good tools allow DSML to change, and reflect changes:
  - to modelling tools
  - to design models already made

Outline

- Introduction
- The vision of Model Driven Development
- Examples and case studies
- Architecture for defining and using DSM
- Implementing DSM
  - Identifying modelling concepts and rules
  - Building generators
  - Building a domain framework

Implementing modelling languages

- The most important asset of a DSM environment
  - application engineers use it
  - generator and framework largely invisible
- Often includes elements of familiar modelling paradigms
  - state machine
  - flow model
  - data structure, etc.
- Language specified as a metamodel



Identifying DSM constructs

- Use domain concepts directly as modelling 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 modelling
  - do not visualize product code!
    - better to ”forget” your current code
- Apply suitable computational model(s) as a starting point

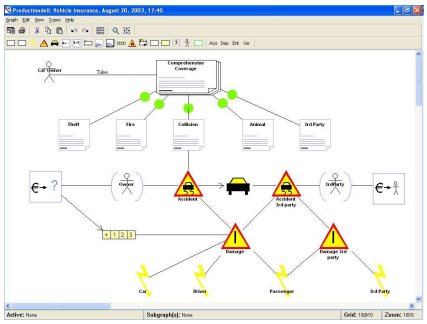
Approaches to identify concepts

- "How do I start to do DSM?"
  - Hard problem for DSM beginners
  - Analyzed over 20 cases to find good toolbox of approaches
- Initial analysis suggested five approaches:
  1. Domain expert's or developer's concepts
  2. Generation output
  3. Physical structure
  4. Look and feel of the system built
  5. Variability space

Problem domain	Solution domain/ generation target	Approach
Telecom services	Configuration scripts	1
Insurance products	J2EE	1
Business processes	Rule engine language	1
Industrial automation	3 GL	1, (2)
Platform installation	XML	1, (2)
Medical device configuration	XML	1, (2)
Machine control	3 GL	1, 2
Call processing	CPL	2, (1)
Geographic Information System	3 GL, propriety rule language, data structures	2
SIM card profiles	Configuration scripts and parameters	2
Phone switch services	CPL, Voice XML, 3 GL	2, (4)
eCommerce marketplaces	J2EE, XML	2, (4)
Automation network	C	3, 4
Crane operations	C/C++	3, (5)
SIM card applications	3 GL	4
Applications in microcontroller	8-bit assembler	4
Household appliance features	3 GL	4
Smartphone UI applications	Scripting language	4
ERP configuration	3 GL	4, 5
ERP configuration	3 GL	4, 5
Handheld device applications	3 GL	4, 5
Phone UI applications	C	5, (4)
Phone UI applications	C++	5, (4)

1. Domain expert's concepts

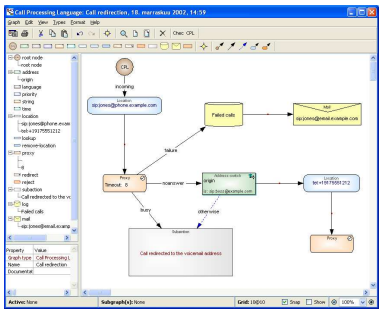
- Concepts from domain
- Mostly made without help
- Simple MoC
- Simple code generation
- OK in established domain
- Usable by non-coders



Insurance products/J2EE

2. Generation output

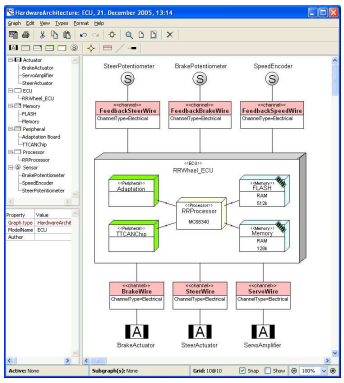
- Modelling constructs come from code artefacts
- Static parts are easy
  - Data structures
  - Core XML elements
- Dynamic behaviour hard
  - Full programming language?
  - Need domain framework
- Danger: low level of abstraction
  - Little productivity gain
- But works well with DSL or XML
  - As opposed to generic 3GL



Internet telephony/CPL

3. Physical structure

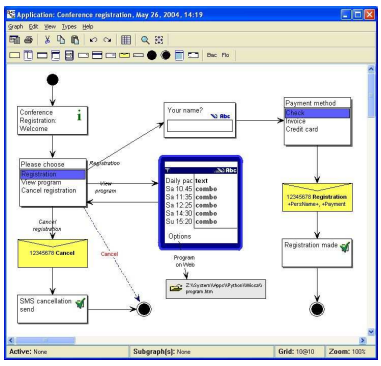
- Best for physical systems
  - Networks, logistic systems, HW architecture, train control, factory automation, etc.
- Often static data model MoC
  - Also describes connections and dependencies
  - May include behavioural elements
- Visible domain concepts
  - Easy to identify
  - High level of abstraction
- Usually linked to other models (and DSLs) to achieve more comprehensive code generation



Automotive HW architecture

4. Look and feel of the system

- Best for physical end product
  - UI on PC, embedded, speech
- Often state machine MoC
  - Also data & control flow
  - Power of relationships
- Visible domain concepts
  - Easy to identify
  - High level of abstraction
- Domain framework hides code
  - Don't write code in models…
  - …unless you really have to!
- Generators considered easy



Smartphone apps/Python

5. Variability space

- Language concepts capture variability space
- Modeller makes variant choices
  - Composition, relationships, values
- Infinite variability space (Czarnecki)
  - Not just feature tree: unbounded product family
- Used to create hardest DSM languages
  - Handled most complex domains, kept modelling simple
- Static variance easy, dynamic harder
- Consultant should be good coder
- Customer expert in his domain and code
  - Consultant should also be able to program
- Predict future variability ⇒ high level of abstraction

Evaluation of the Approaches

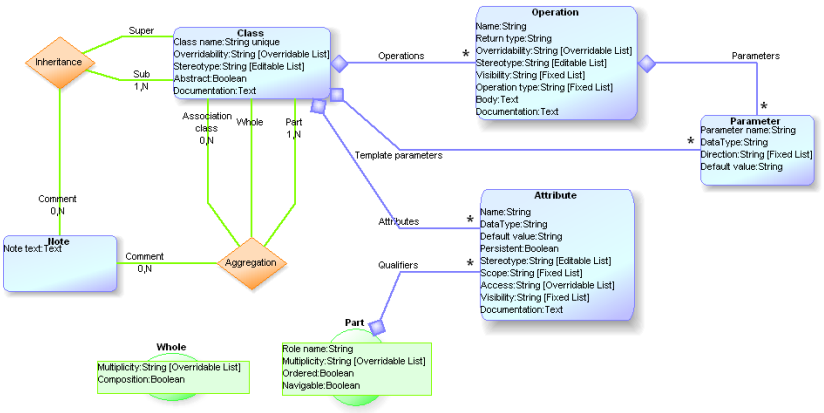
- Only certain pairs of approaches occurred
- Hierarchy of approaches
  - From less to more experienced DSM practitioners

1. Domain expert's concepts – "we just did it"
2. Generation output
   - Generic/ad hoc language not so good
   - Established DSL good
3. Physical structure
   - To support specifications in other DSM languages
4. Look and feel: common, easy, true DSM
5. Variability space: adds power to handle complexity
   - Found in very different domains

- Best results combined 4 (L&F) and 5 (Variability)
  - 4 gives objects, 5 gives relationships and properties

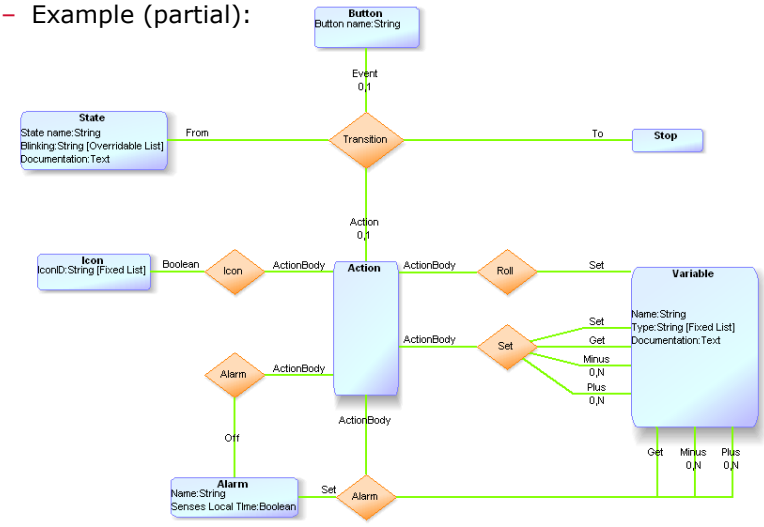
Defining a metamodel

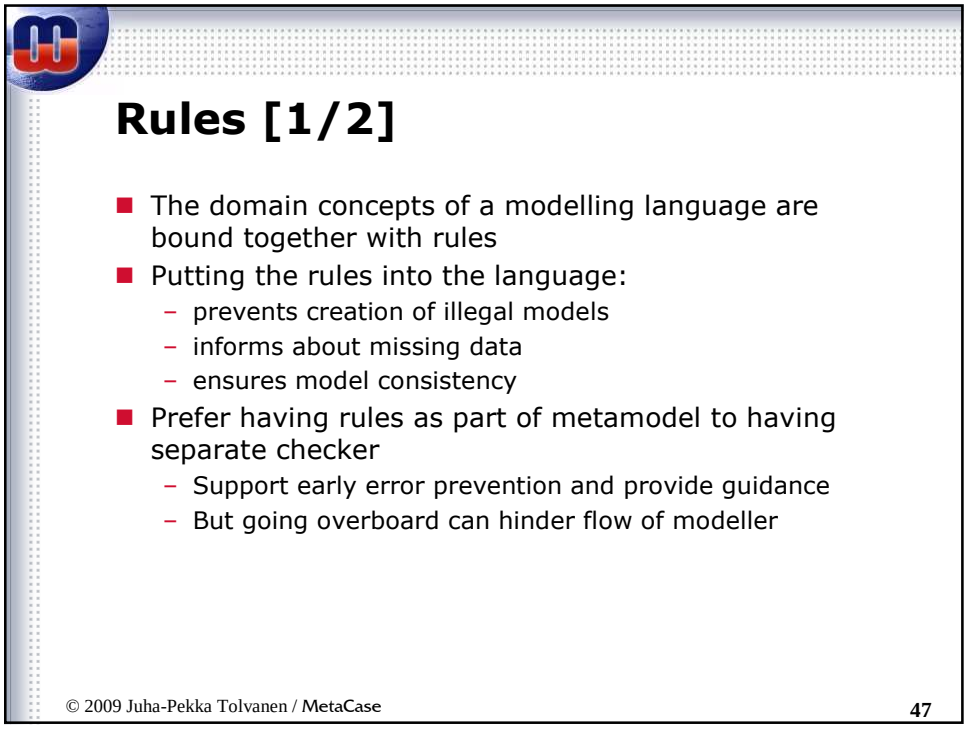
- Metamodel describes the modelling language
  - e.g. class diagram (partially below) or any other language



Metamodel of wristwatch apps

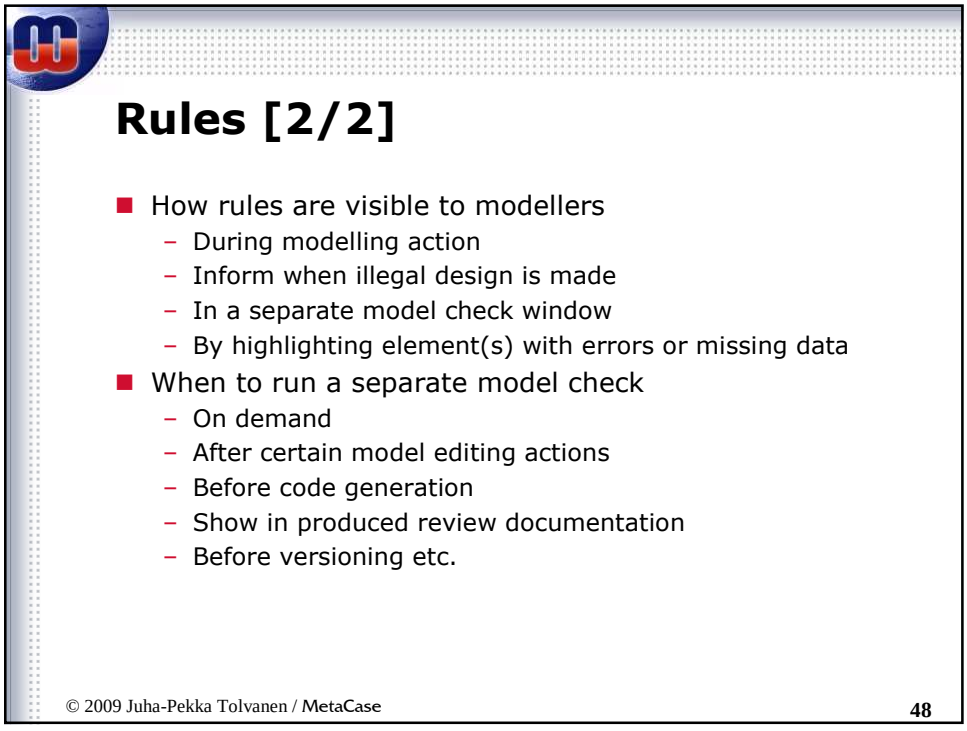
- Example (partial):





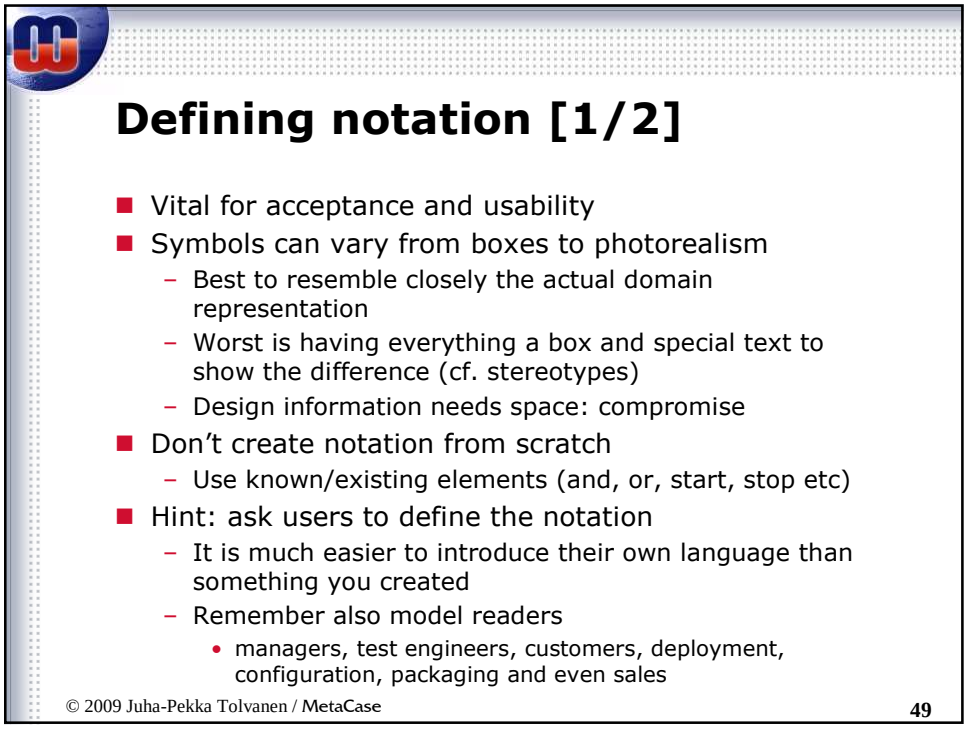
Rules [1/2]

- The domain concepts of a modelling language are bound together with rules
- Putting the rules into the language:
  - prevents creation of illegal models
  - informs about missing data
  - ensures model consistency
- 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 modeller



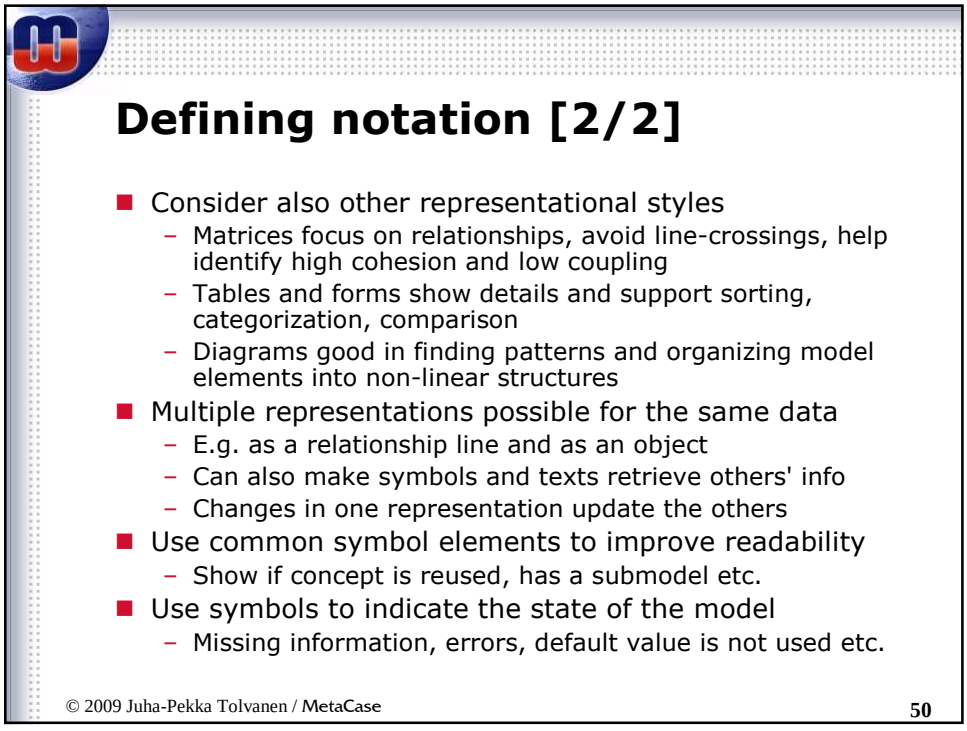
Rules [2/2]

- How rules are visible to modellers
  - During modelling action
  - Inform when illegal design is made
  - In a separate model check window
  - By highlighting element(s) with errors or missing data
- When to run a separate model check
  - On demand
  - After certain model editing actions
  - Before code generation
  - Show in produced review documentation
  - Before versioning etc.



Defining notation [1/2]

- 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

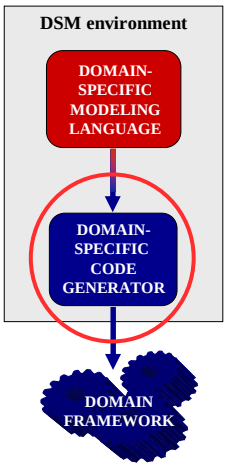


Defining notation [2/2]

- Consider also other representational styles
  - Matrices focus on relationships, avoid line-crossings, help identify high cohesion and low coupling
  - Tables and forms show details and support sorting, categorization, comparison
  - Diagrams good in finding patterns and organizing model elements into non-linear structures
- Multiple representations possible for the same data
  - E.g. as a relationship line and as an object
  - Can also make symbols and texts retrieve others' info
  - Changes in one representation update the others
- Use common symbol elements to improve readability
  - Show if concept is reused, has a submodel etc.
- Use symbols to indicate the state of the model
  - Missing information, errors, default value is not used etc.

Generator

- Generator translates the computational model into a required output
  1. crawls through the models
     → navigation according to metamodel
  2. extracts required information
     → access data in models
  3. translates it into the code
     → translation semantics and rules
  4. using some output format
     → possibility to define output format

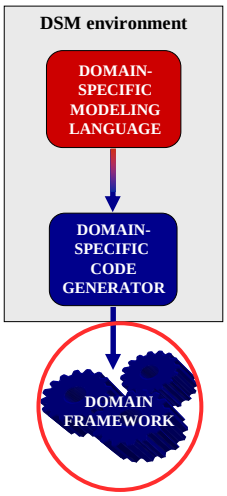


How to design a generator

- Make the generator just for your situation
  - Trying to make general purpose generator often fails
- Make generation process complete, target 100% output
  - Never edit generated code: edit generator or framework
  - Do you try to edit Assembler and keep C in synch with it?!
- Don't visualize code
  - Generating one class header from one class in a diagram helps very little, if at all…
- Put domain rules up-front to the language
  - Generator definition becomes easier when the input is correct
- Keep generator and generated code as simple as possible
  - Raise variation to the modelling language
  - Push low-level implementation issues to the framework

Domain framework

- Provides an interface for the target platform and programming language
- Raise the level of abstraction on the platform side
- Achieved by atomic implementations of commonalities and variabilities
  - especially for behaviour
  - implementation as templates and components
- Include **interface** for the code to be generated
  - often the only needed part for static variation (e.g. for XML schema)



Generators aren't just for code…

- Checking completeness and uniformity
- Configuration
- Testing and analysis
- Automated build → automating compile and execution
- Help text
- User guides
- Documentation and review

Outline

- Introduction
- The vision of Model Driven Development
- Examples and case studies
- Architecture for defining and using DSM
- Implementing DSM
- Concluding remarks
  - Summary
  - Q & A

Summary

- Productivity can be improved by a raise in abstraction
- DSM solves the pitfalls of CASE and UML:
  metamodel and generators can be custom built
- DSM has a big organizational impact
  - Experts make the DSM environment
  - Other developers do model-driven development
- Building your own tool is hard, but meta-tools exist
  - Meta-tools make moving to DSM feasible
- DSM makes the best possible use of your expert(s)
  - And they'll love it!

Thank you!

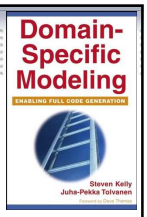
Free evaluation download: www.metacase.com
Build your first DSM language in an hour!

<plug>If you like it after 31 days, see 150€ Intro offer</plug>

MetaCase
Juha-Pekka Tolvanen
jpt@metacase.com

www.metacase.com/blogs

Literature and further links



- DSM Forum, www.dsmforum.org
- Blogs: www.metacase.com/blogs
- Brinkkemper, S., Lyytinen, K., Welke, R., Method Engineering - Principles of method construction and tool support, Chapman & Hall, 1996
- Czarnecki, K., Eisenecker, U., Generative Programming, Methods, Tools, and Applications, Addison-Wesley, 2000.
- Gray, J., Rossi, M., Tolvanen, J-P, (eds.) Special issue of Journal of Visual Languages and Computing on Domain-Specific Modelling with Visual Languages, Vol 15 (3-4), 2004
- Kelly, S., Tolvanen, J.-P., Domain-Specific Modeling: Enabling Full Code Generation, Wiley, 2008. http://dsmbook.com
- 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, 1996.
- Pohjonen, R., Kelly, S., Domain-Specific Modelling, Dr. Dobb's, 8, 2002
- Tolvanen, J.-P., Kelly, S., Defining Domain-Specific Modelling Languages to Automate Product Derivation: Collected Experiences. Procs of the 9th International Software Product Line Conference, Springer-Verlag, 2005.
- Weiss, D., Lai, C. T. R., Software Product-line Engineering, Addison Wesley Longman, 1999.