



# Domain-Specific Modeling: Enabling Full Code Generation

*27 November, 2008*

*Juha-Pekka Tolvanen, Ph.D.*

 MetaCase



# Outline

- Why Domain-Specific Modeling
- Industry examples
- How to define modeling languages and code generators
- Industry experiences
- How you can start
- Conclusions

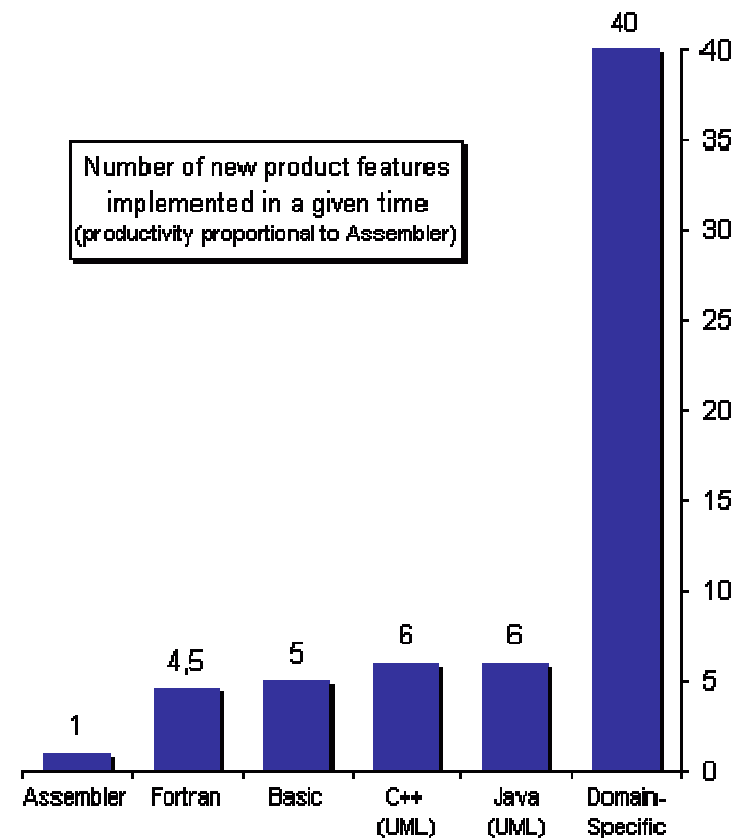


# How has productivity improved?

- "The entire history of software engineering is that of the rise in levels of abstraction"

Grady Booch

- 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

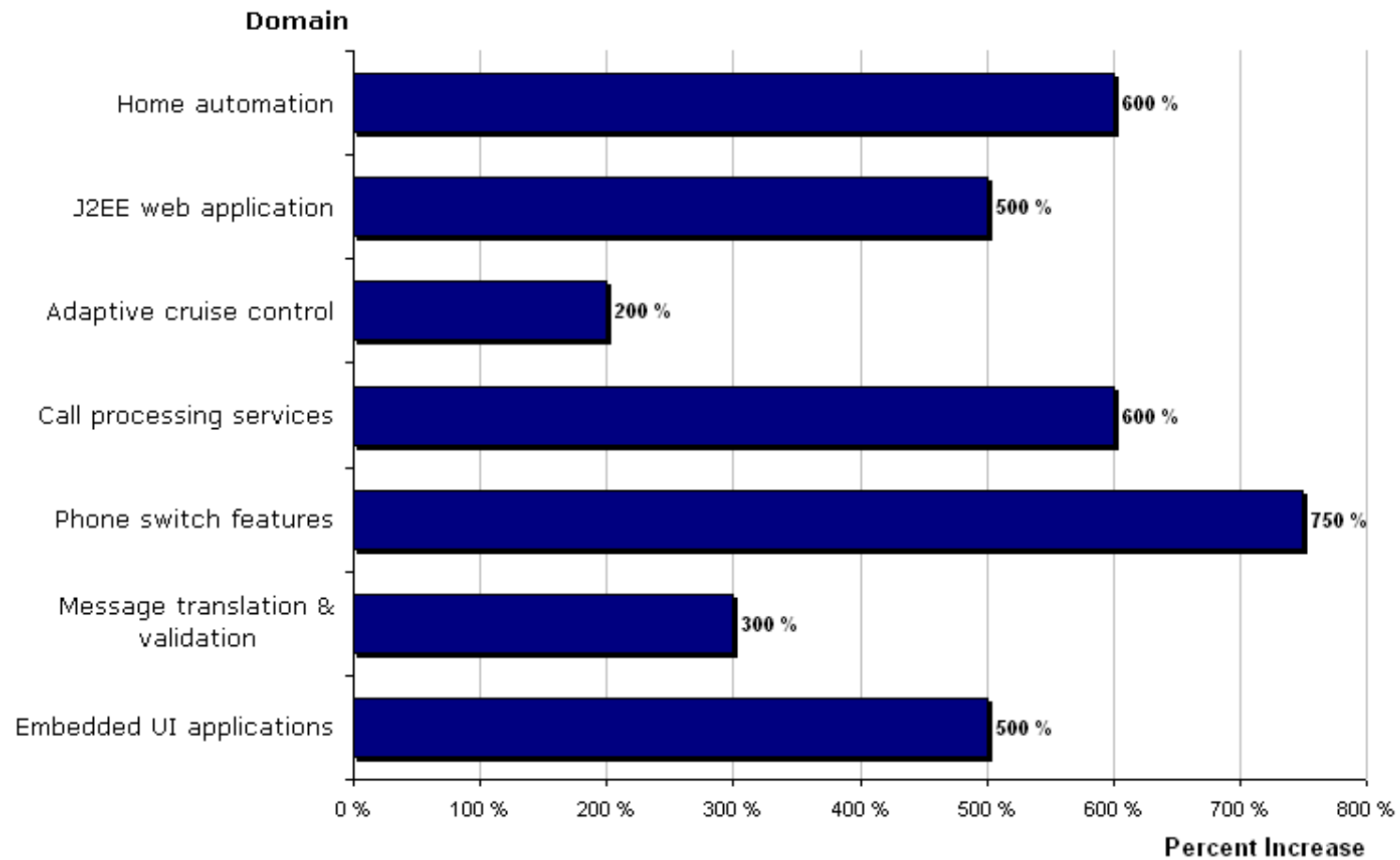


# DSM examples

- See 6 examples in the following 7 minute flash video at:  
[http://www.metacase.com/papers/DSM\\_Examples.html](http://www.metacase.com/papers/DSM_Examples.html)
- Examples cover different domains:
  - Financial/banking
  - IP telephony
  - Home automation
  - Automotive architectures
  - Telecom services
  - Business rules and workflow



# Productivity increase from DSM



\* Productivity proportional to earlier practice

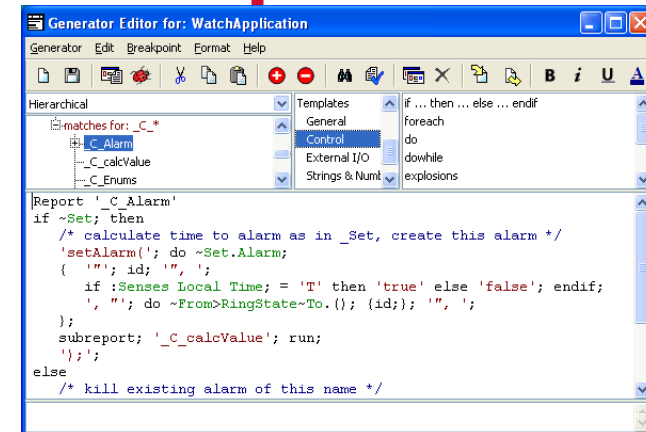
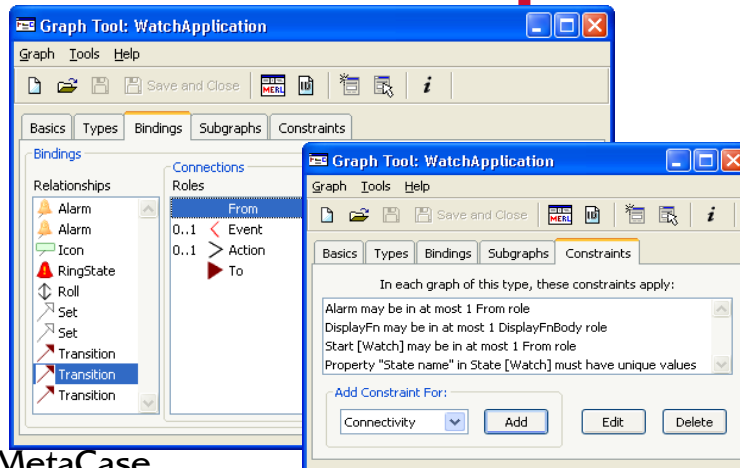
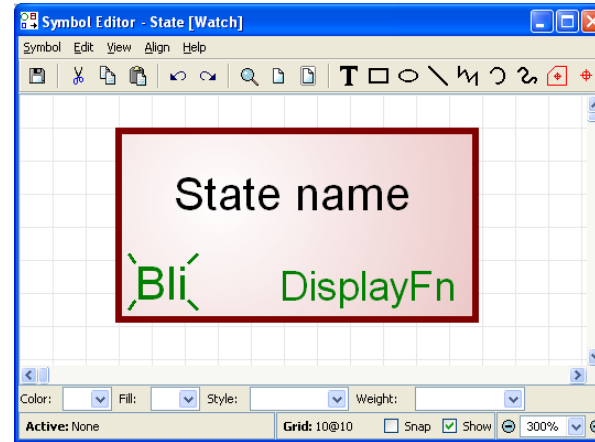
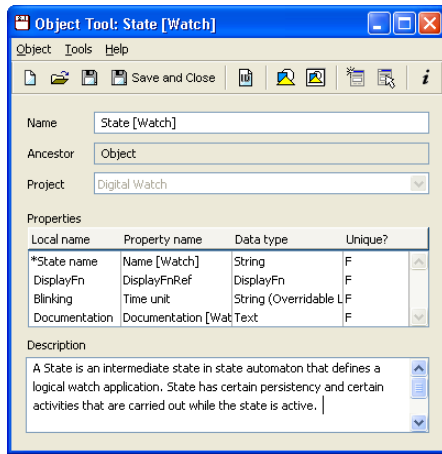


# The steps of defining a DSM solution

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, ...



# MetaEdit+ tool for DSM definition





# MetaEdit+ delivers modeling tools needed

- Editors (diagram, matrix, table), browsers, generators, multi-user, multi-project, multi-platform environment
- Method easy and safe to maintain and share
  - Shared new language versions, updates models already made

The image displays three windows from the MetaEdit+ environment:

- WatchApplication: SimpleTime, August 31, 2003, 14:47**: A state machine diagram with states **Show**, **Mode**, **EditMinutes**, and **EditHours**. Transitions are shown between these states. The **EditMinutes** and **EditHours** states have an **editOffset** property.
- Report Output: Stopwatch: WatchApplication**: A window showing a code snippet for a **case Running** state, including a **switch (button)** and a **case Up** block with assignments for **stopTime**, **icon**, and **state**.
- C:\MetaEdit\MetaEdit MWB 3.0\Reports\Stopwatch.html - Microsoft...**: A window showing the definition of the **stopTime (Variable)**. It lists properties: **Name [Watch]** is **stopTime**, **Type** is **METime**, and **Documentation [Watch]** is "Variable that stores the current stop time: how many seconds had elapsed when the stopwatch was stopped."



# Demonstration on defining languages and generators

- See 15 minute video at:

[http://www.metacase.com/papers/DSM\\_Definition.html](http://www.metacase.com/papers/DSM_Definition.html)



# How to define modeling languages

- Use domain concepts directly as modeling constructs
  - already known and used
  - established semantics exist
  - natural to operate with
  - easy to understand and remember
- Apply suitable computational model(s) as a starting point
- Define as a metamodel, test with a tool
- Build iteratively: define part of language, model with it, modify and extend further, model again...



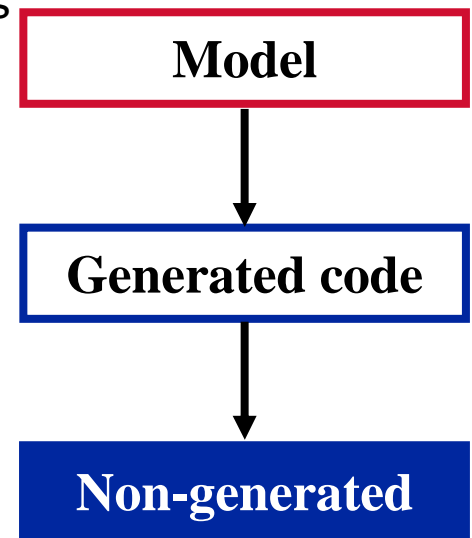
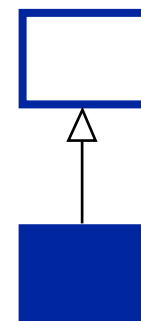
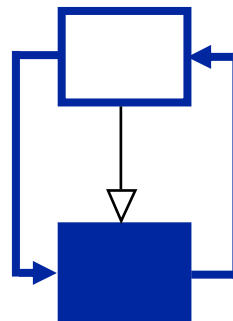
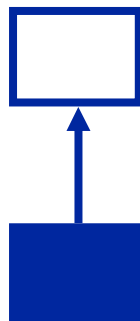
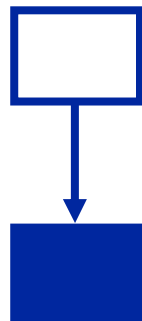
# How to design generators

- Make generator for your situation only
  - Trying to make general purpose generator often fails
- Put domain rules up-front to the language
  - Generator definition becomes easier when the input is correct
  - Models should be impossible to draw wrongly for generation
- Keep generator modular to reflect changes
  - e.g. structure generator based on modeling languages, generated files, modeling concepts
- Make generated code readable (“good looking”)
  - To be used later while debugging the code, executing it in a simulator, and while implementing the generator
  - Follow good coding standards, include comments, have data to link back to models (e.g. in comment or via e.g. simulator)



# Combining generated code and other code

- Different levels of code to integrate
  - Other generated code
  - Domain framework, components, platform functions
  - Hand-made code
- Separation of generated and non-generated code
  - Best to keep them in separate files (or sections in files)
- Generated code...
  - can call existing code, instantiate data structures
  - can be called from existing code
  - can be subclassed from existing code
  - can form base classes



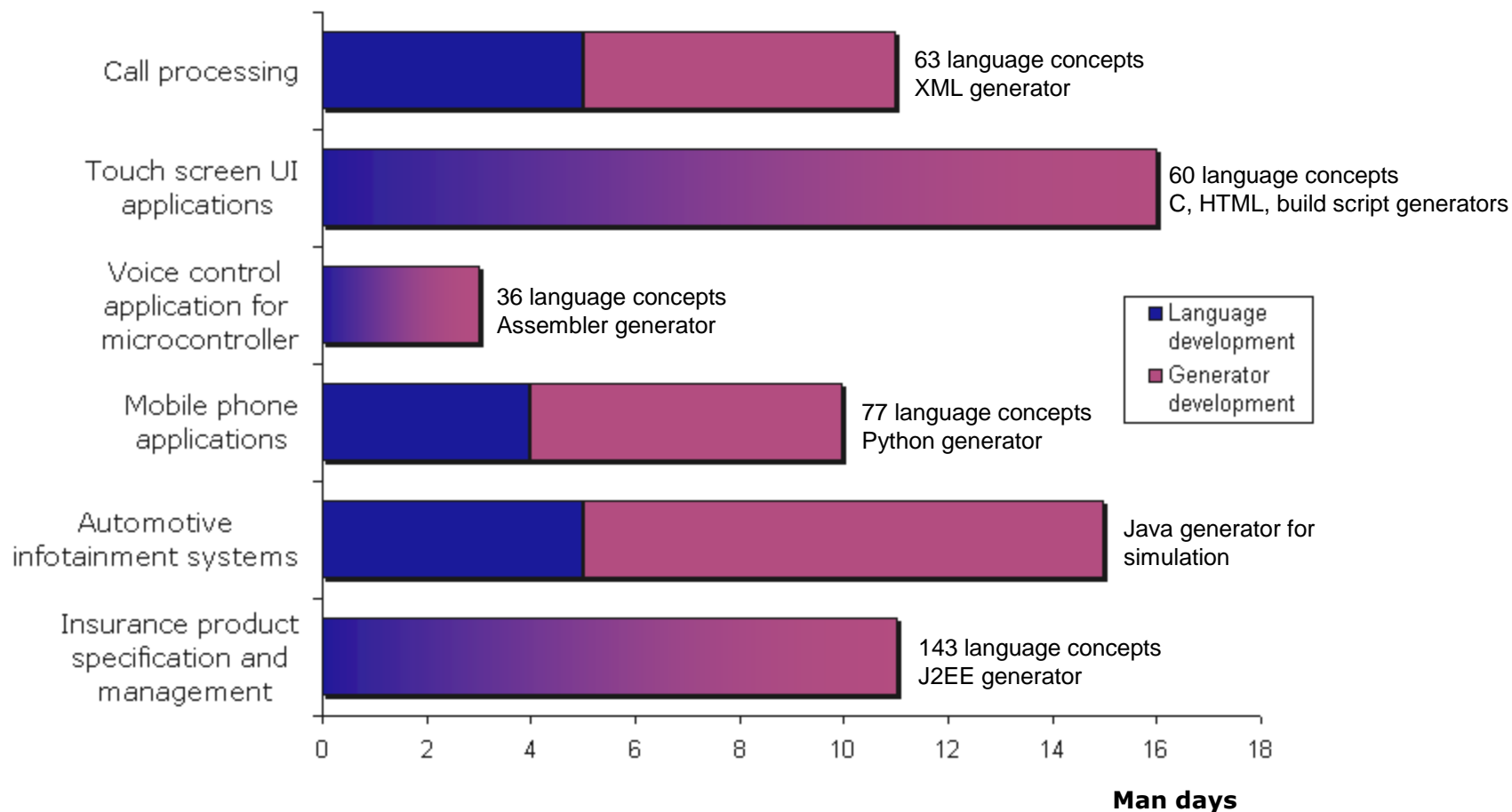


# Generators aren't just for code...

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



# DSM Solution Development Time





# Defining DSLs with MetaEdit+: Engineer testimonials

**Matsushita  
Electric  
Works, Ltd.**

"I could define a domain-specific **language in about six hours** – design, testing and one failed trial included," Laurent Safa

**SIEMENS**

"Modeling with MetaEdit+ was indeed quite convenient and **a lot easier** to accomplish than trying to achieve the same results with Eclipse GMF," Ulf Hesselbarth



"MetaEdit+ is by far the **most advanced** tool in the DSL category, Markus Voelter, Author of Model-Driven Software Development

**DENSO**

"It's now **easy** to make modifications when the AUTOSAR version is changed – implementation and testing requirements are both reduced," Yohsuke Satoh



# How to start: Proof of concept

- Implement small but significant part of DSM solution
  - Modeling language, generator, model for simple app.
- Shows concrete results to management
  - Not just glossy brochures, books – or slide sets!
- Preparation:
  - Outline DSM solution scope, and part to be covered now
  - Pick DSM creation team
    - Add internal/external DSM expert, if possible
  - Set workshop time (2 days), meeting to present results
    - Ideally meeting is straight after
    - Hard deadline focuses group on concrete results, not meta



# Pilot project

- When language & generators ready for first real case
  - Also need prototype documentation and training
- Separate team to test DSM solution
  - Must be on a real application (not time critical)
- Opens up modeling language to change again
  - Modeling language was stable when making generator
- Gather rough productivity statistics
  - They prove nothing, but are evidence nonetheless



# Summary

- Domain-specific modeling radically improves productivity (5-10x)
- DSM leverages expert developers' abilities to empower other developers in a team
- Metamodel-based tools provide a cost-effective way to create DSM infrastructure
  - A variety of tools is available
- Building DSM is great fun
  - The ultimate refactoring!

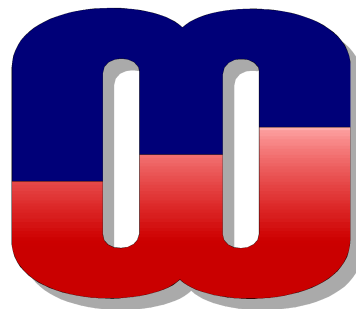


# Thank you!

Free evaluation download: [www.metacase.com](http://www.metacase.com)

Build your first DSM language in an hour!

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



[www.metacase.com](http://www.metacase.com)  
[info@metacase.com](mailto:info@metacase.com)

Europe:

MetaCase

Ylistönmäentie 31

FI-40500 Jyväskylä, Finland

Phone +358 14 4451 400

Fax +358 14 4451 405

USA:

MetaCase

5605 North MacArthur Blvd.

11th Floor, Irving, Texas 75038

Phone (972) 819-2039

Fax (480) 247-5501



## For more information...

- Chat and see a demo in room 103
  
- Read the book:  
Domain-Specific Modeling,  
Wiley, 2008

