Defining Domain-Specific Modeling Languages

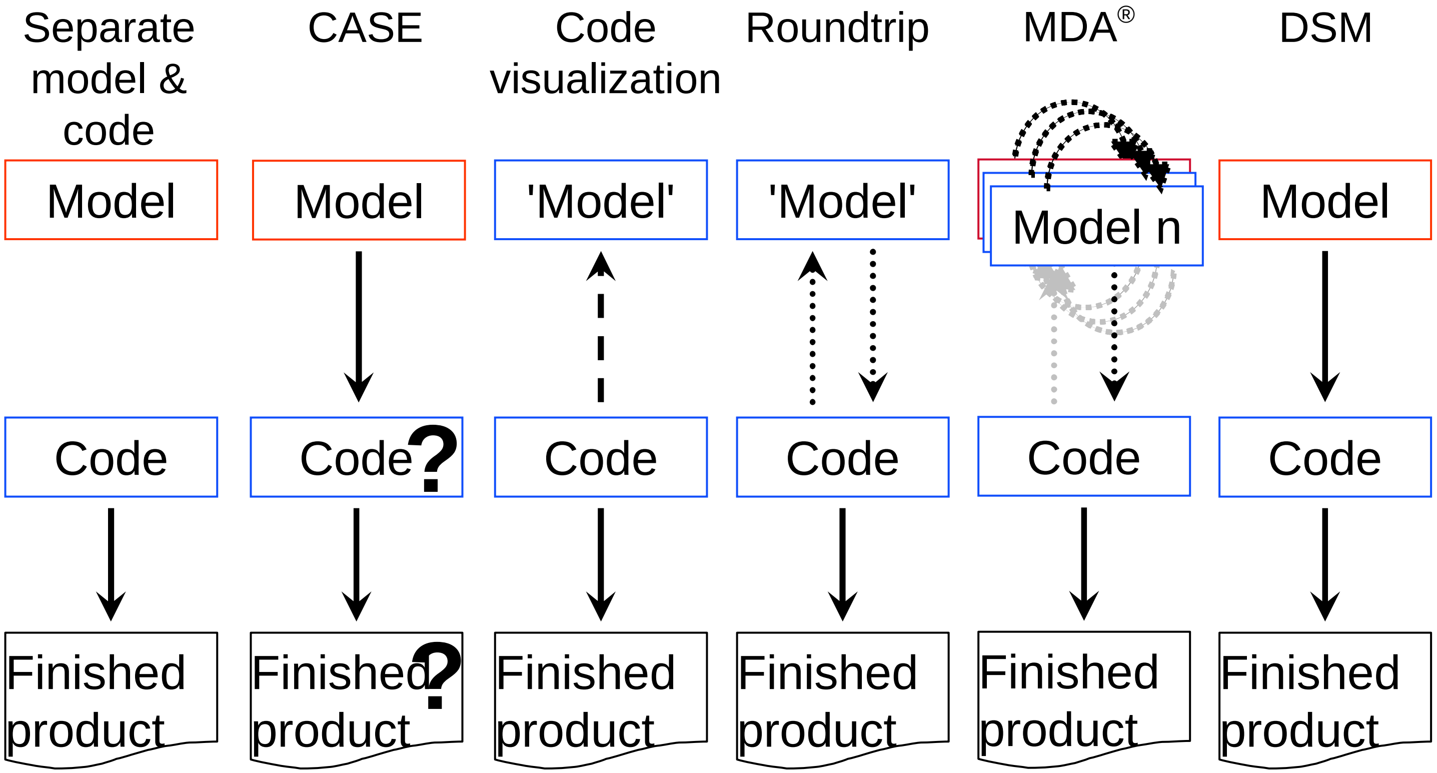
1st Oct 2008

Juha-Pekka Tolvanen
MetaCase

Relevant language classifications to start with

- General-Purpose / Domain-Specific
  - Narrow area of interest
  - Often inside one company only
- Problem Domain / Solution Domain
  - Higher abstraction leads to improved productivity
- External / Embedded (internal)
  - Guide developers
  - Scalable to a larger potential developer base
- Graphical / Text / Matrix / Table / Map etc.
  - Easy to read, understand and communicate with
  - Humans are good at spotting visual patterns
- Static structures / Behavior

How do we use models?



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

Demo:
Domain-Specific Modeling

Does it work? Some industry examples

- Automotive infotainment MMI
- E-commerce visitor navigation
- Financial contracts
- IMS Service Creation
- Insurance products
- Message translation and validation in C^3 system
- Pacemaker product line
- Phone switch configuration
- Phone UI applications
- Professional radio applications
- SIM applications
- Train interlocking control
- Voice control of home automation

Experiences from practice

Panasonic

“**5-fold** productivity increase when compared to standard development methods”

EADS

“The quality of the generated code is clearly better, simply because the modeling language **rules out errors**”

DENSO

“**Eliminated our need to outsource** software development activities"

NOKIA
Connecting People

“A module that was expected to take 2 weeks now **took 1 day** from the start of the design to the finished product”

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 modeling 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

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 changed under commercial pressure
  - MS modeling tools immature ⇒ de-emphasize models
- Focus on MS partners building and selling DSLs
  - ISV sells same DSM solution to many companies
  - Offsets the "massive effort"* of using their tools
    - *Quote from Prashant Sridharan, MS 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 (moving back to UML?)

– 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

Domain-Specific Modeling

- Focus on a narrow area of interest => Domain
- Language 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.

DSM Pros & Cons

- + Fundamental productivity and quality improvements
  - + 300% faster in academic study, 1000% reported by companies
  - + 50% less errors in an academic study
- + Gives full control to the company
  - + Experienced developers are sitting in the driver’s seat
- – Requires expertise and resources from the company
- + Minimal vendor lock
  - + You can translate & transform models to other tools and formats
- – Only few industry strength tools available
  - – Scalability to a larger number of developers
  - – Do not handle evolution and maintenance

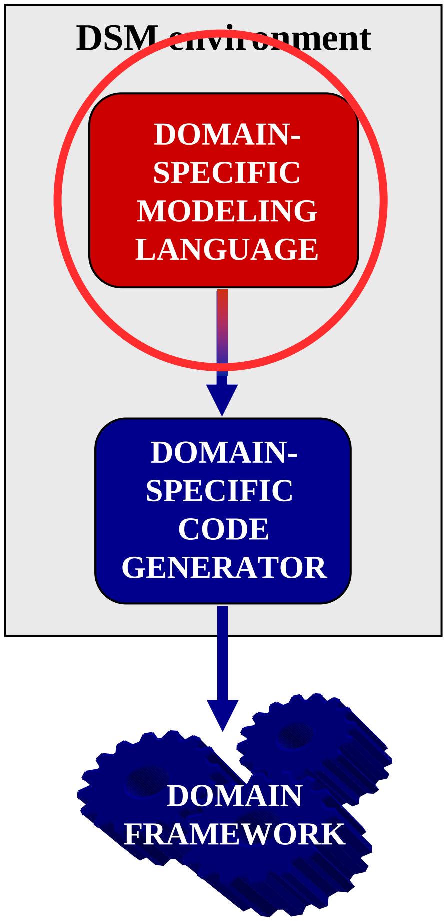
The steps of defining a DSM solution

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, run generator to compare with reference code, extend the metamodel, model some more, ...

Implementing Domain-Specific Modeling languages

- The most important asset
  - becomes the ”source (code)”
  - application engineers use it
  - generator and framework largely invisible
- Often includes elements of familiar modeling paradigms
  - state machine
  - flow model
  - data structure, etc.
- Language specified as a metamodel
- Profiles are reduced form of metamodeling
  - Poor man’s solution

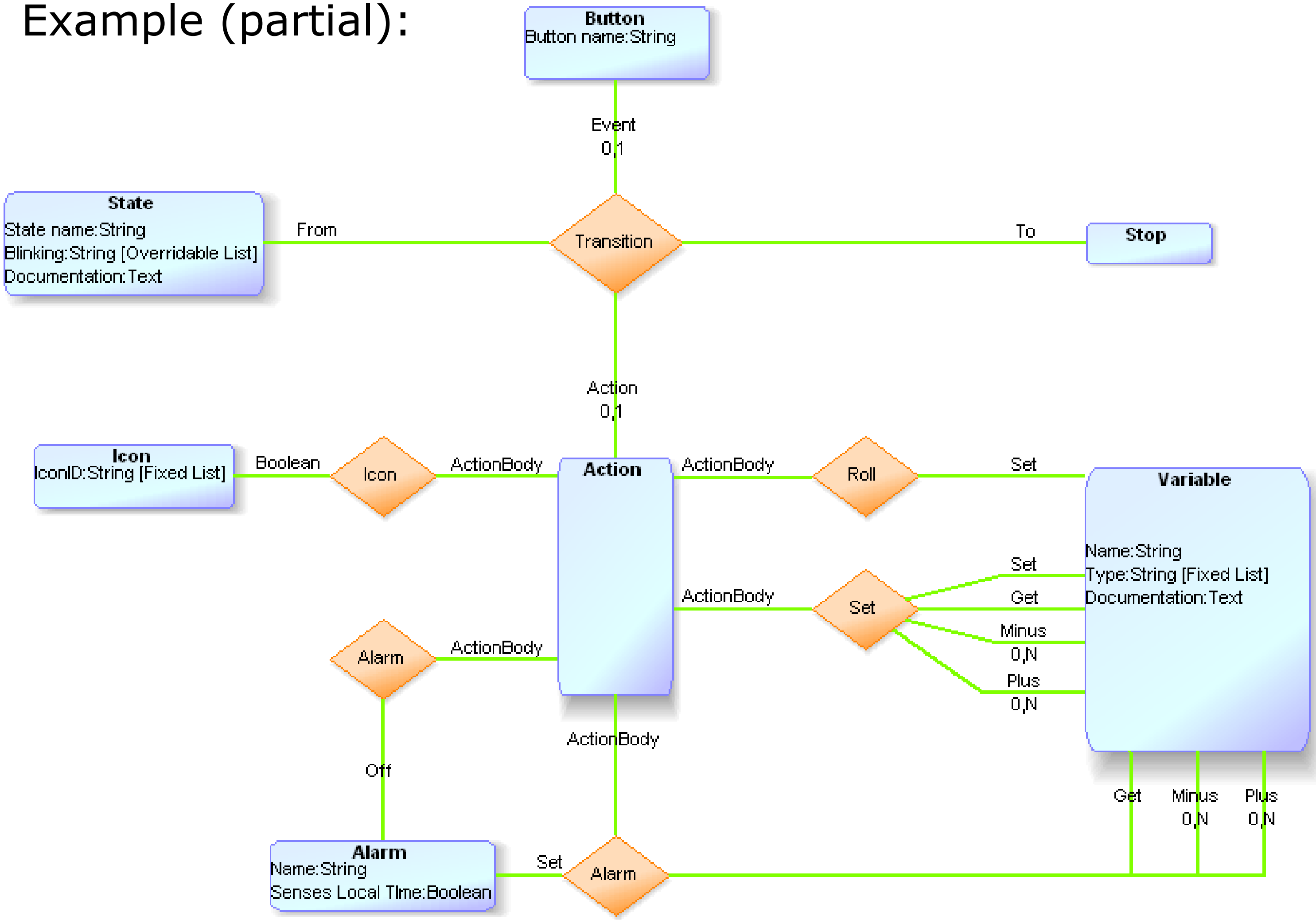

DSM environment
DOMAIN-SPECIFIC MODELING LANGUAGE
DOMAIN-SPECIFIC CODE GENERATOR
DOMAIN FRAMEWORK

Identifying language constructs

- Use domain concepts directly as modeling constructs
  - already known and used
  - established semantics exist
  - natural to operate with
  - easy to understand and remember
- Focus on expressing design space with the language
  - use parameters of variation space
  - try to minimize the need for modeling
- Apply suitable computational model(s) as a starting point
- Build iteratively: define part of language, model with it, modify and extend further, model again...

Metamodel of wristwatch apps

– Example (partial):



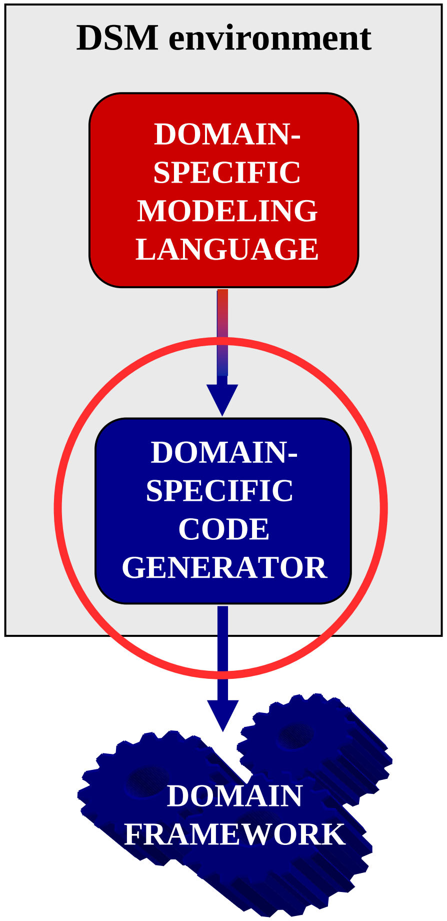
Demo on language definition

- Demo available as flash video at:

http://www.metacase.com/papers/DSM_Definition.html

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


DSM environment
DOMAIN-SPECIFIC MODELING LANGUAGE
DOMAIN-SPECIFIC CODE GENERATOR
DOMAIN FRAMEWORK

Types of generator facilities

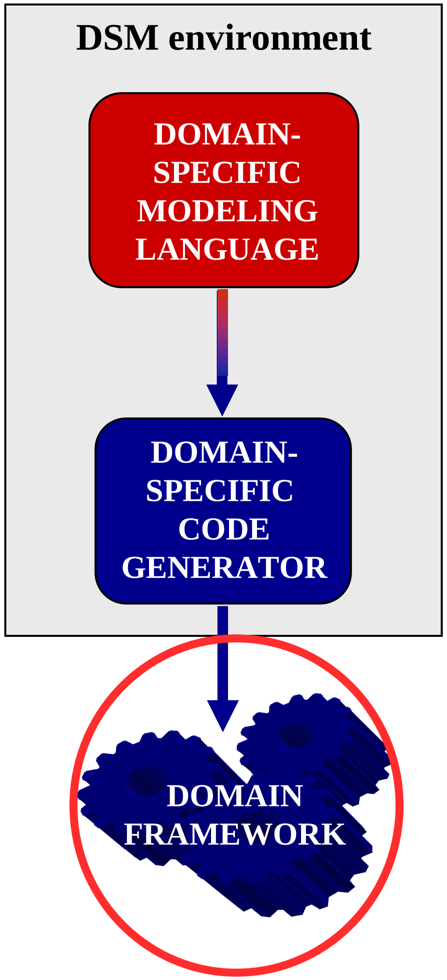
- Programming language accessing model through API
  - Direct access, but low level, high coupling with tool
  - Need something better, designed just for generation
- Model visitor
  - Map each model structure to a code structure
  - Limited to simple one-to-one mappings
- Output template
  - Single file, code + escaped <%generator commands%>
- Crawler: Model navigation and output streams
  - Multi-file, code quoted, native generator commands
- Generator generators
  - Tempting, alluring, mostly unnecessary

How to design a generator

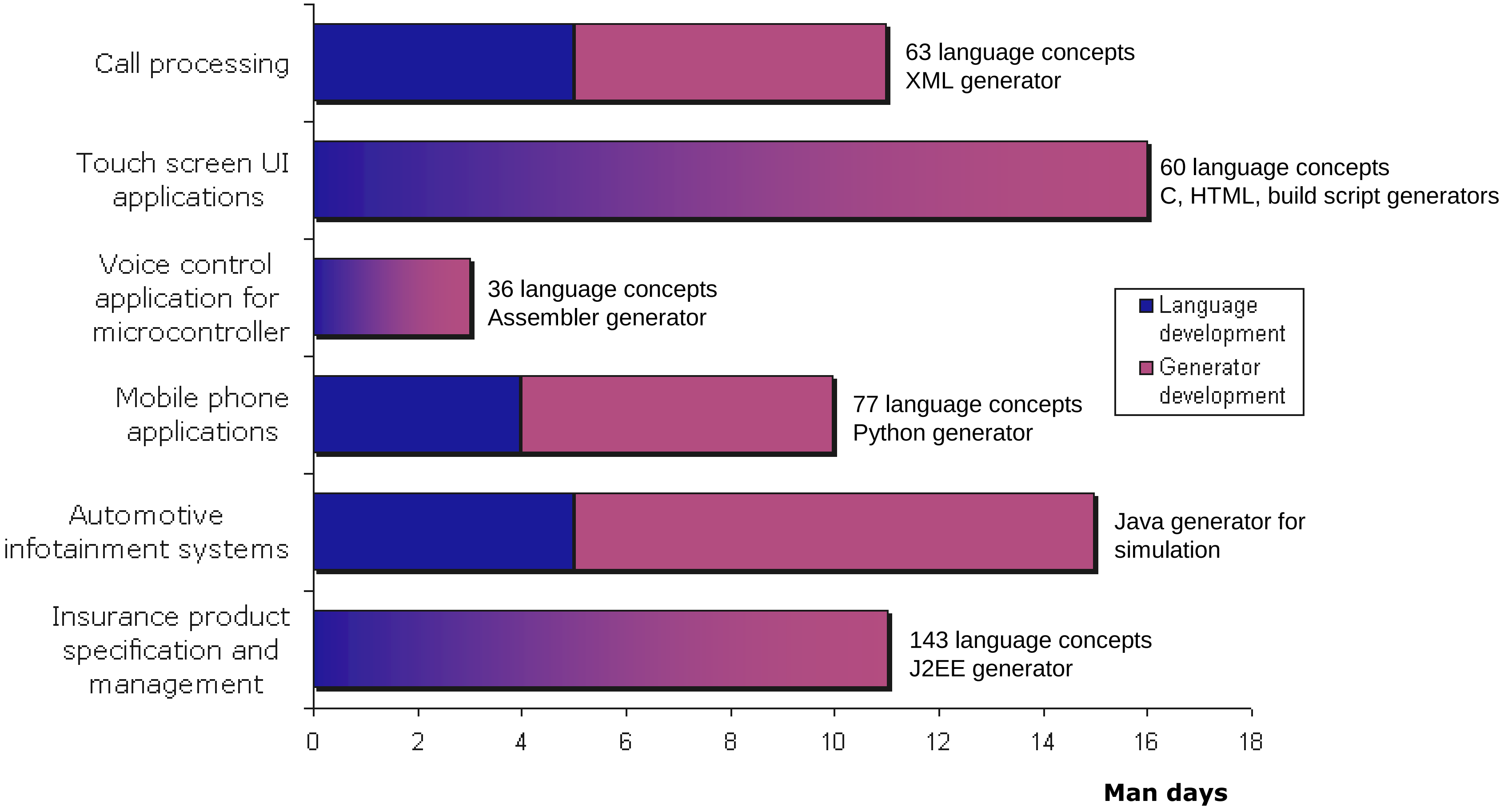
- Make generator for your situation only
  - Trying to make general purpose generator often fails
  - Have (or create) reference implementation
- Put domain rules up-front to the language
  - Generator definition easier when the input is correct
  - Models should be impossible to create wrongly for generation
- 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)

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 behavior
  - 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)


DSM environment
DOMAIN-SPECIFIC MODELING LANGUAGE
DOMAIN-SPECIFIC CODE GENERATOR
DOMAIN FRAMEWORK

DSM Solution Development Time



63 language concepts
XML generator

60 language concepts
C, HTML, build script generators

36 language concepts
Assembler generator

77 language concepts
Python generator

Java generator for
simulation

143 language concepts
J2EE generator

Choosing a good domain for DSM

- If you have more than one candidate, pick best first
  - Maturity of target business area in your company?
  - Low coupling with external organizations?
  - Related to other candidate domains?
  - Extent of customization per customer?
  - Good existing source code examples?
  - In-house framework?
  - Software development process maturity?
    - Should be mature, but not *too* locked down
  - Availability of architect / expert developers?
    - Never, ever try building DSM with summer interns!

Organizing for DSM

- Building a DSM solution requires few resources
  - Quality not quantity
  - For lower-level DSM tools, may need basic coders too
    - May be best to start with high-level, go low later if needed
- Team needs both domain and code expertise
  - Best candidates are normally expert developers
- Each member must balance elegance with pragmatism
- Look for people who build macros and templates:
  - For the whole team, not just themselves
  - Prepared to maintain and document them

Success factors

- A narrow focus
  - the narrower the better, later easier to extend
- Own the framework (or part of it)
  - e.g. in-house application framework
- Apply on domains where ”repetition” (ROI)
  - multiple products or features, developers, targets
- Have experience in the domain
  - have made several similar kind applications already
- Have expertise
  - one of the top three who built the first products
  - an experienced developer (author): makes the generator
- Tools that support both language definition and use
  - evolution and iterative development

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

For more information...

- Tutorial tomorrow (13:00 - 17:00)
- Domain-Specific Modeling, Wiley-IEEE, 2008