MetaCase

Choosing the Best Level of Abstraction for Your Domain-Specific Language

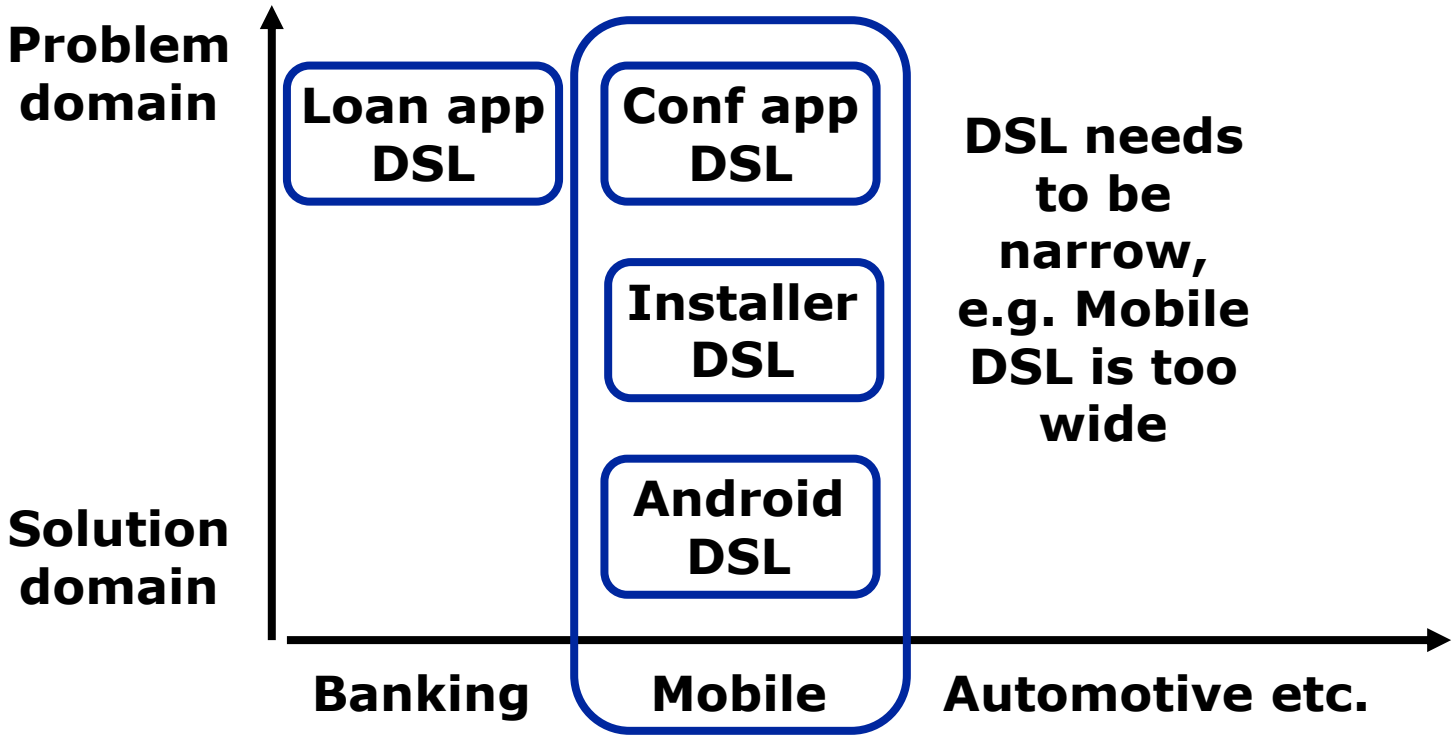
1 July 2013

Juha-Pekka Tolvanen, PhD
jpt@metacase.com

Some (repeated) definitions

- General-Purpose / Domain-Specific
- External / Embedded / Internal Languages
- Problem Domain / Solution Domain
- Graphical / text / matrix / table / maps etc. as concrete syntax
- Static / behavior
- Turing complete / in-complete
- Domain knowledge as first class citizen / naming convention

(Your) area of interest = domain



Where at the main benefits?

- More empirical research is needed, but studies show improvements in:
  - Productivity
  - Quality
  - Process
  - Maintenance tasks
  - Understanding and communication
  - Easier introduction of new developers etc.
- While time-to-market is often the most significant improvement, many industry cases shows 5-10x (**500-1000%**) productivity improvements (see references)

Industry experiences

"The setup effort to create the languages was a couple of weeks and provided more than **ten times faster speed**"

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

"The DSML solution makes development **significantly faster** and easier than the old manual coding practices"

Controlled empirical studies

Panasonic

- Built the same system twice: 425% faster
- Built code generator for a second platform: a fraction of time

POLAR
LISTEN TO YOUR BODY

- Lab study: 6 engineers develop typical features: > 750% faster
- Built the same system twice: 900% faster

HOFERNET.com

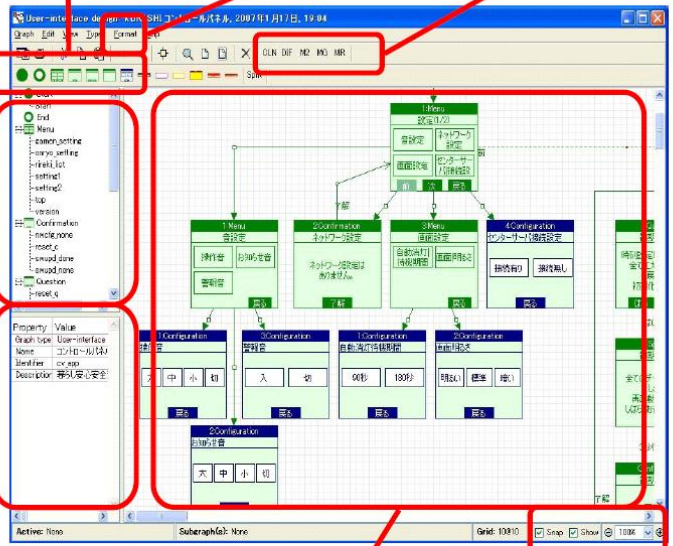
- Built the same production system in parallel
- Built several similar systems: Break Even Point = 3,14

Panasonic

Standard pages and connectors are available through the icon bar

On-site tool configuration facilities

Code generators and Work flow automations

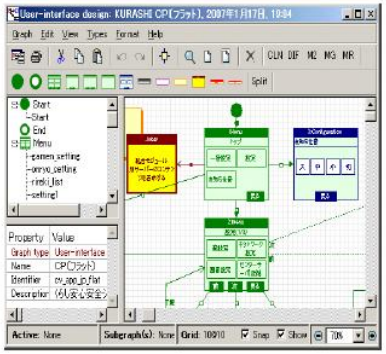


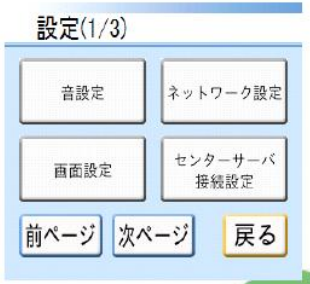
Concepts browser

Detailed view

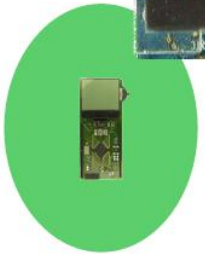
Visual modeling workspace

Zoom and other view modifiers



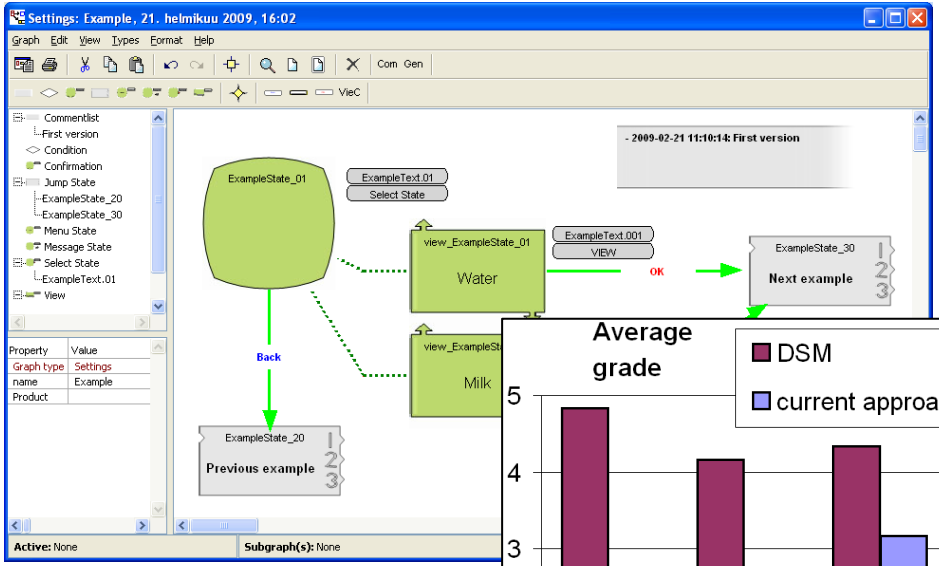


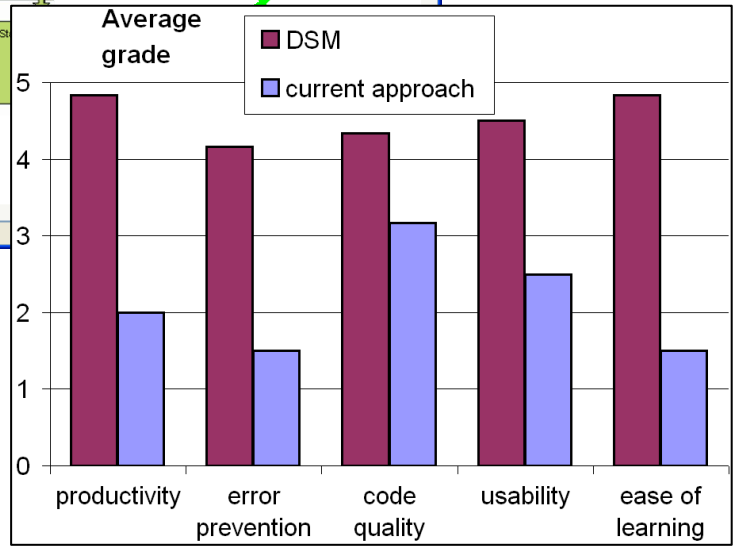
Embedded controller



Microcontroller

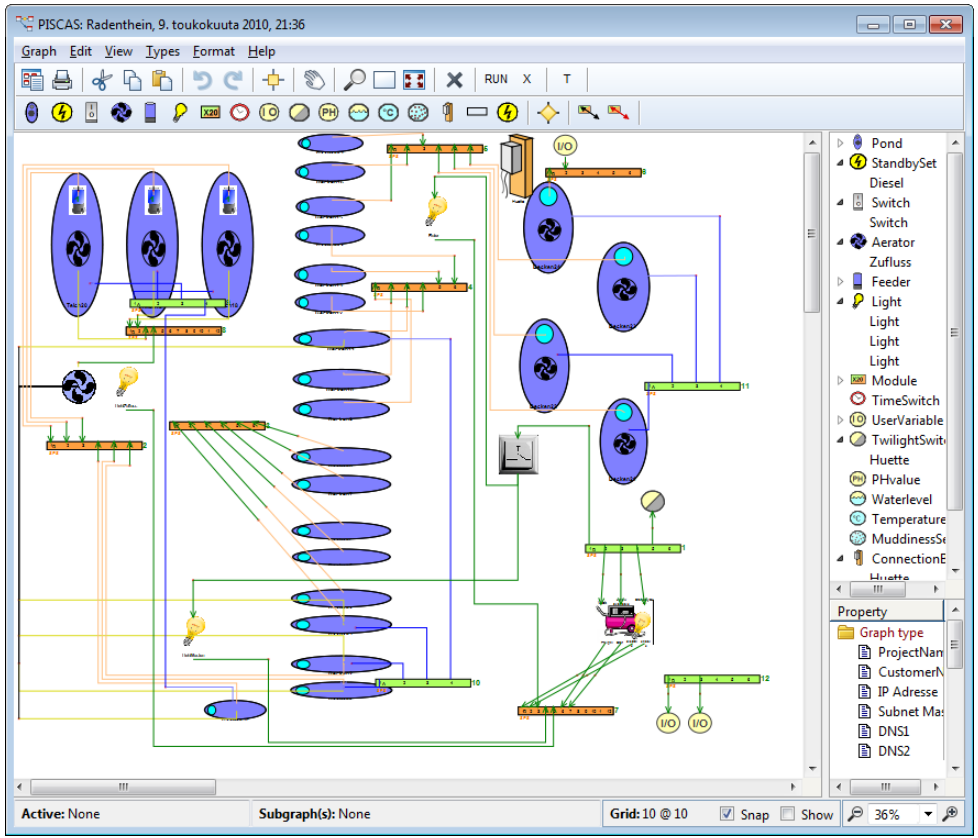
POLAR
LISTEN TO YOUR BODY

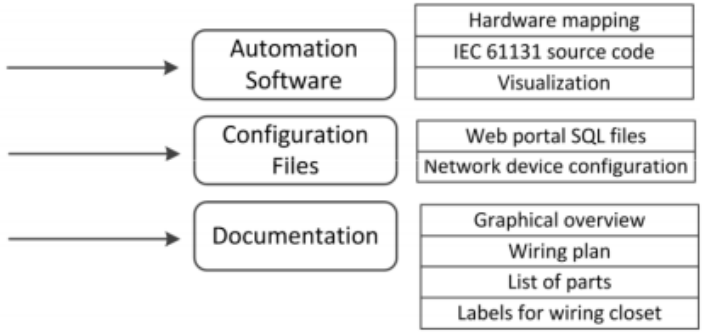




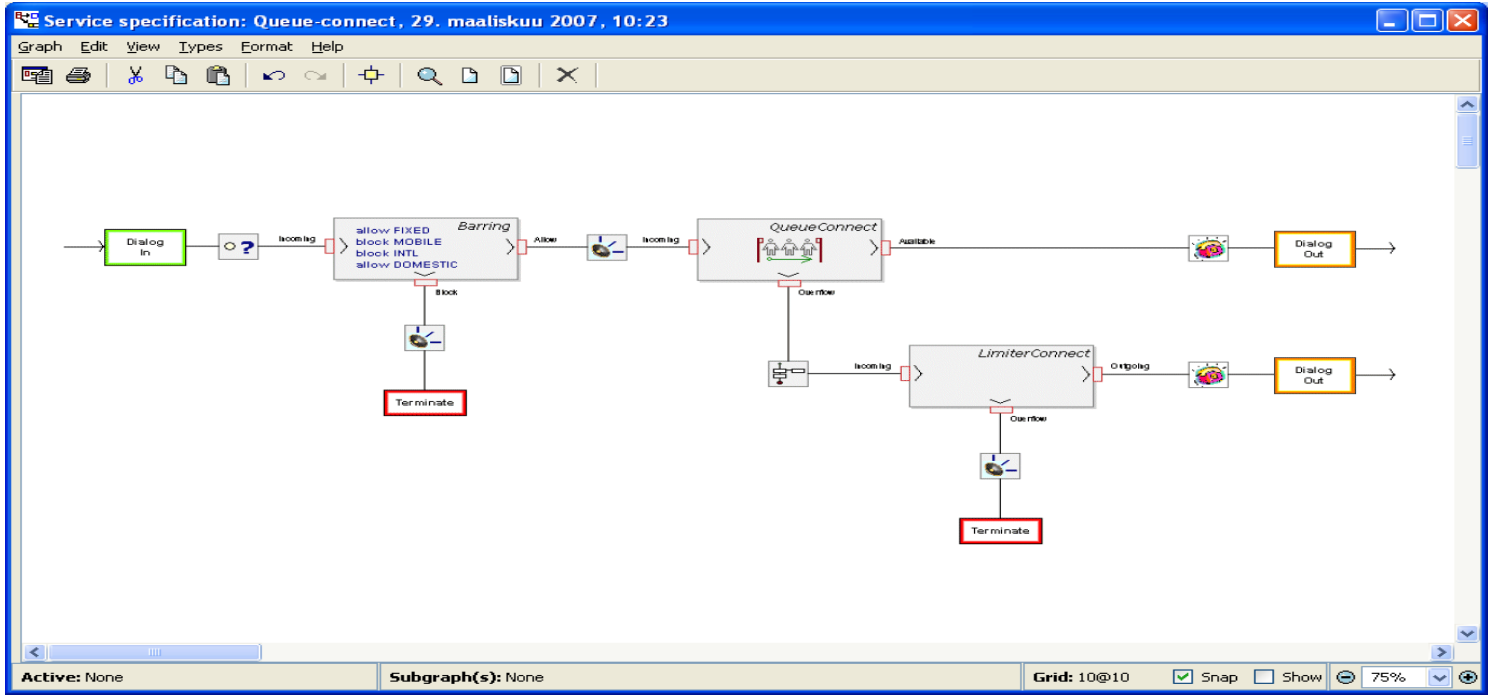
www.metacase.com

HOFERNET.com

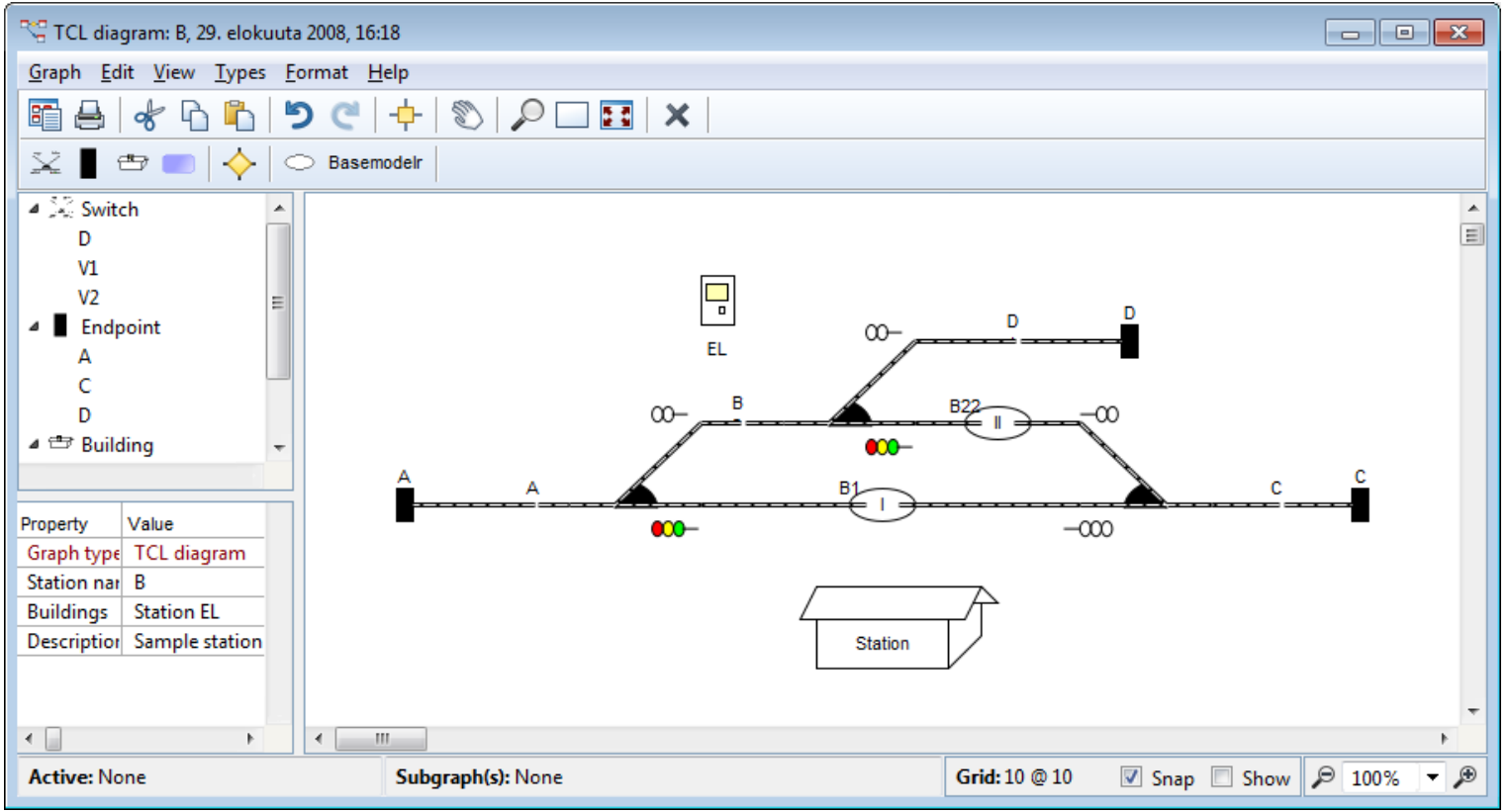




Telecom services

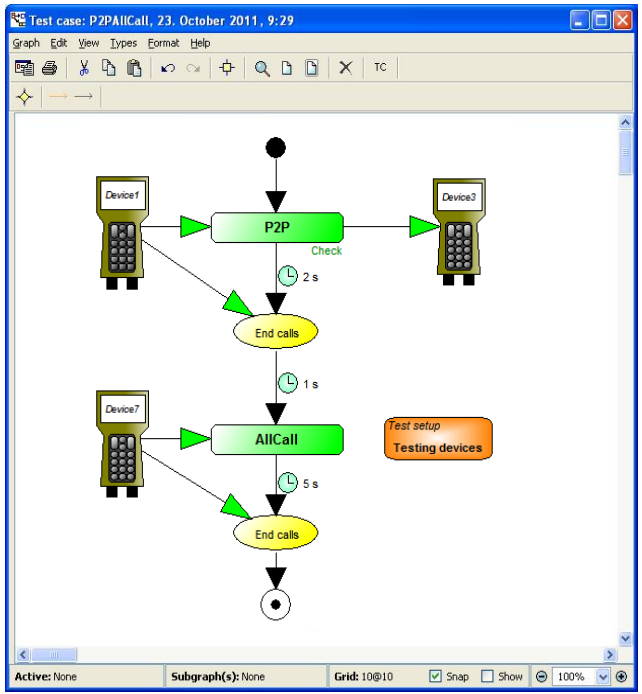


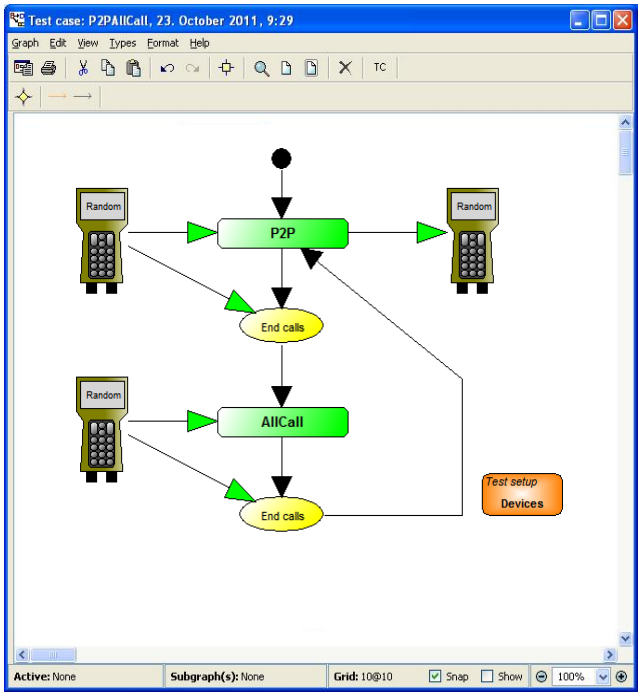
Railway interlocking



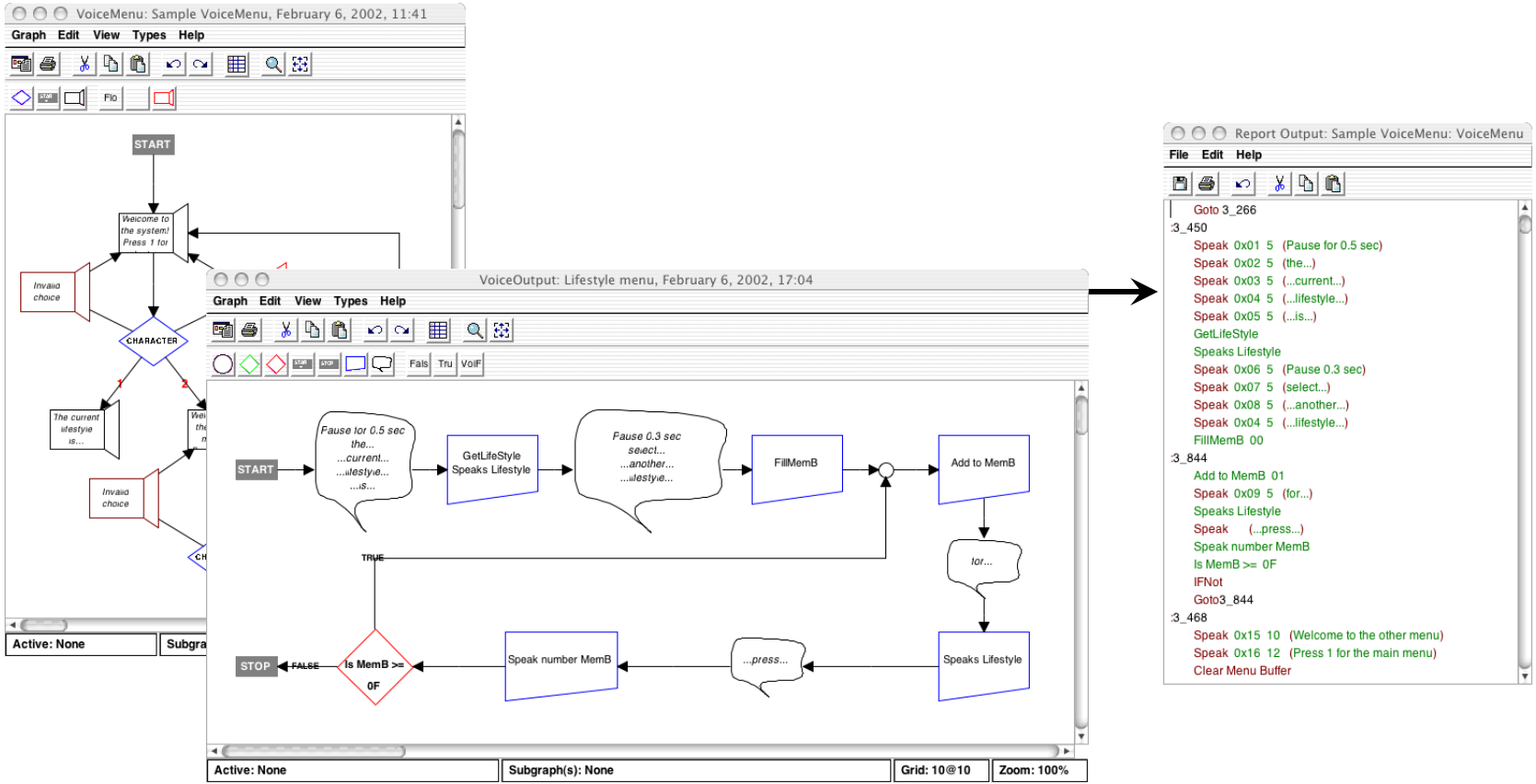
Radio network testing

- Modeling test cases/test logic and generating test data

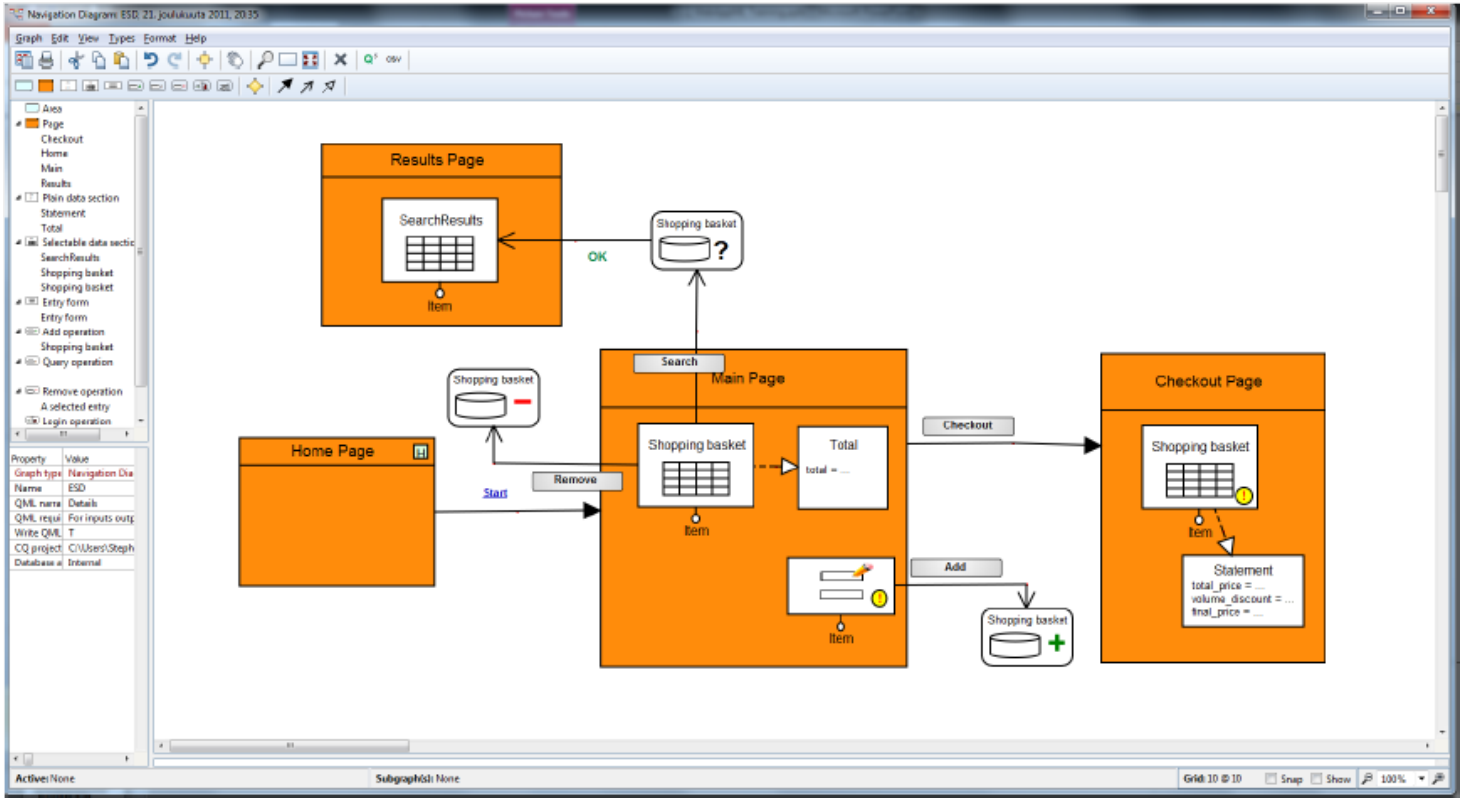




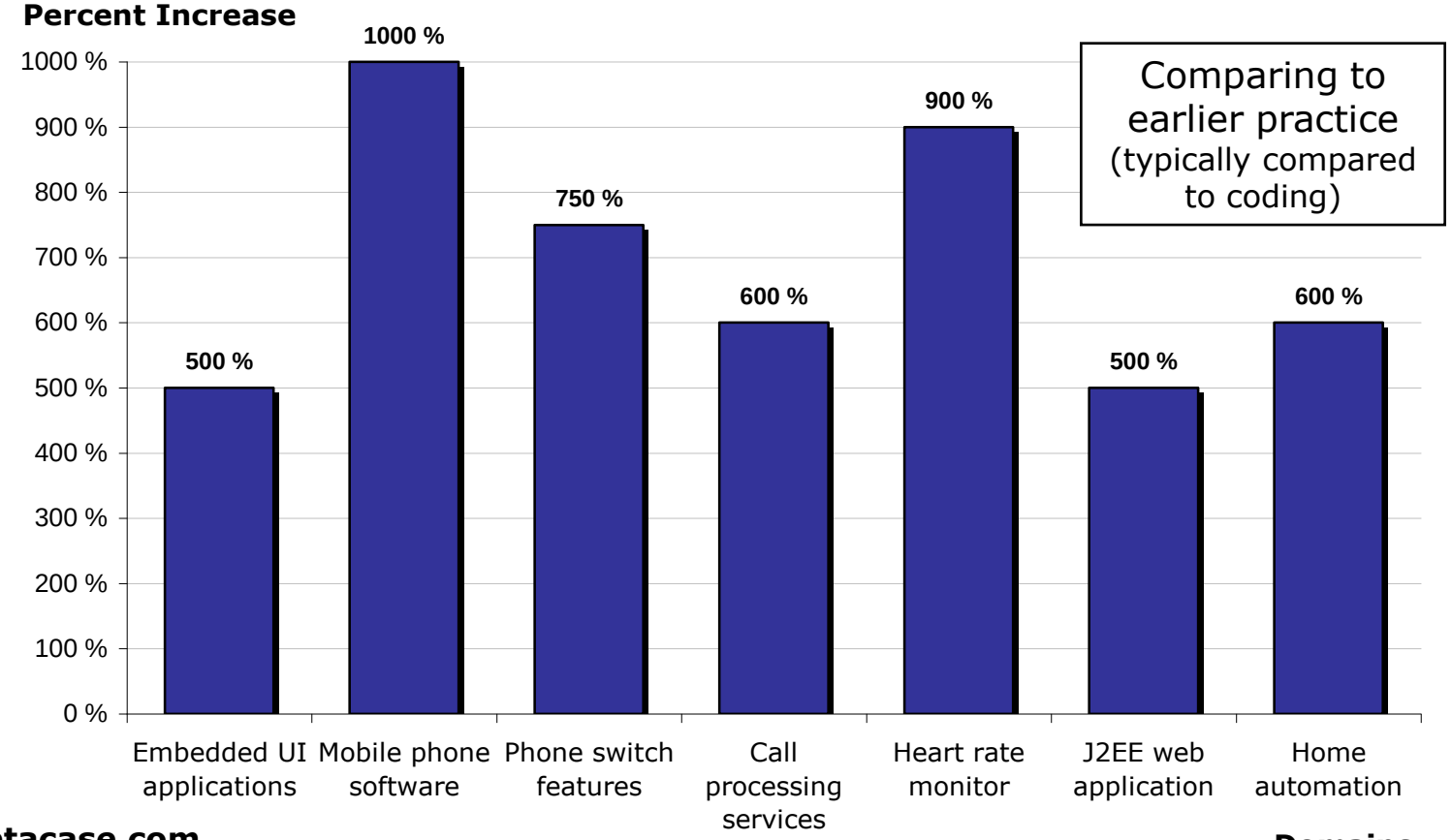
Voice control



Web app testing



Productivity increase measured



Analysis of the cases indicates

- DSL should raise the level of abstraction above the code
- DSL is not using necessarily linear text representation
  - Domain and problem solving dictates
  - Maps, diagrams, matrixes, tables etc.
- DSL focus on narrow domain
  - Exclude outside as much as possible
- DSL for other than producing the code
  - Higher abstraction is not applicable only for producing the code but can also be used for testing, deployment, interaction design, localization etc.

Tooling

- Some sort of tooling is always used, but big differences on tools, see
  - series of Language Workbench Comparison, LWC 2011-2013
  - Comparison reports, e.g. tinyurl.com/gerard12: Language implementation (Eclipse GMF 25 days vs. MetaEdit+ 0.5 days)
- Language creation is the first task but also other issues need to be handled, like
  - integrating multiple languages
  - sharing languages
  - maintaining languages
  - updating specifications made with earlier version of language
  - collaboration, like multiple language engineers

Summary

- Raise the abstraction as high as possible:
  ideally 1:1 to problem domain
- Exclude outside as much as you can
- Refine languages as needed (keep it flexible)
- Use tools that support “agile” language definition
  - And allow also models to automatically update to new language version

Thank You!
Questions?

To see various cases, examples, and download MetaEdit+ tool, visit http://www.metacase.com

References

- Kelly, S., Tolvanen, J.-P., Domain-Specific Modeling: Enabling Full Code Generation, Wiley, 2008. DSMbook.com
- El Kouhen, A., Dumoulin, C., Gérard, S., Boulet, P., Evaluation of Modeling Tools Adaptation, tinyurl.com/gerard12
- Kärnä, J., et al. Evaluating the Use of Domain-Specific Modeling in Practice, 9th DSM Workshop (2009)
- Puolitaival, et al. Utilizing Domain-Specific Modeling for Software Testing, Proceedings of VALID, (2011)
- Preschern et al. Domain Specific Language Architecture for Automation Systems: An Industrial Case Study, Procs of Graphical Modeling Language Development, DTU (2012)
- Safa, L., The Making Of User-Interface Designer, A Proprietary DSM Tool, Procs of 7th DSM Workshop (2007)
- Sprinkle et al. (eds) IEEE Software, DSL&M special issue, July/Aug, 2009, including: Kelly & Pohjonen, Worst Practices for DSM, tinyurl.com/worstDSM