



MetaCase

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

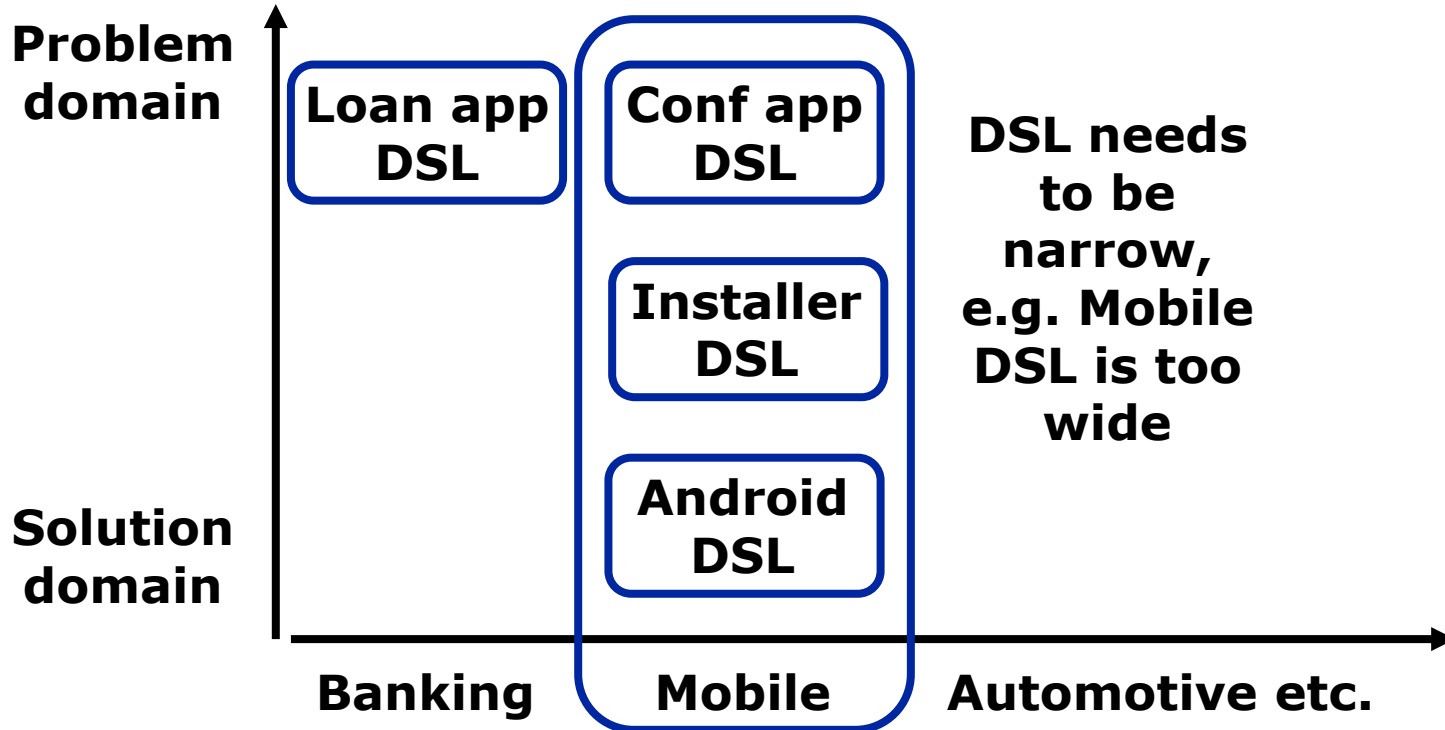
1 July 2013

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



Elektrobit

"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



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



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

Standard pages and connectors are available through the icon bar

On-site tool configuration facilities

Code generators and Work flow automations

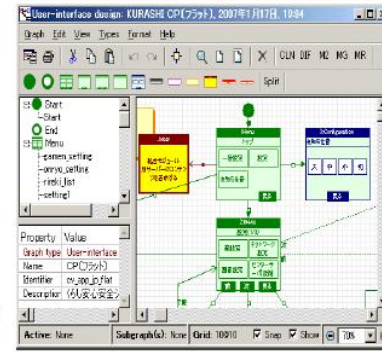
Concepts browser

Detailed view

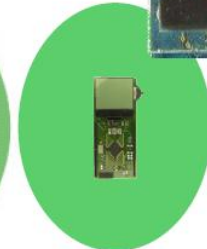
Visual modeling workspace

Zoom and other view modifiers

The screenshot shows a software development environment for user interfaces. It features a 'Concepts browser' on the left with a tree view of elements like 'Start', 'Menu', and 'Confirmation'. A 'Detailed view' pane shows properties for the selected element. The main workspace is a 'Visual modeling workspace' containing a flowchart of menu items and settings, such as '音設定' (Sound Settings), 'ネットワーク設定' (Network Settings), and 'センターサーバ接続設定' (Center Server Connection Settings). A toolbar at the top includes icons for standard actions and a menu bar with options like 'OLN DF ME MG MR'. A status bar at the bottom shows 'Grid: 10010' and 'Snap' options.

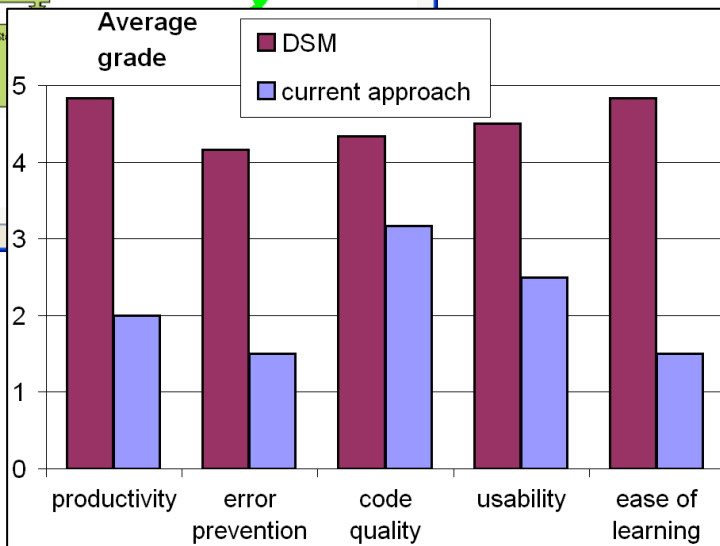
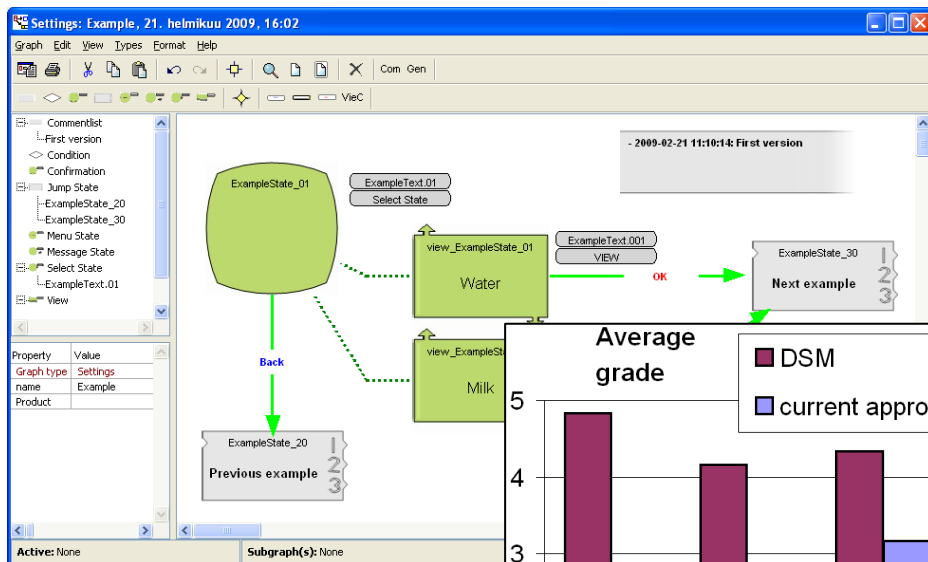


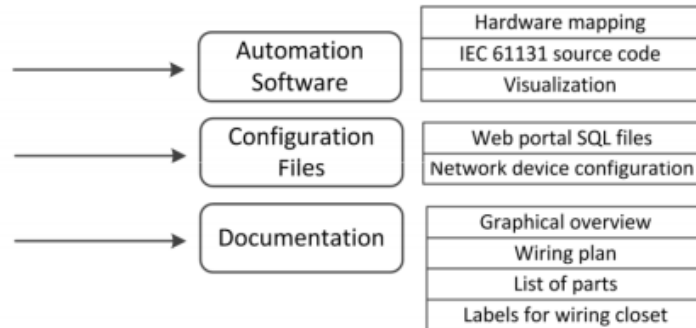
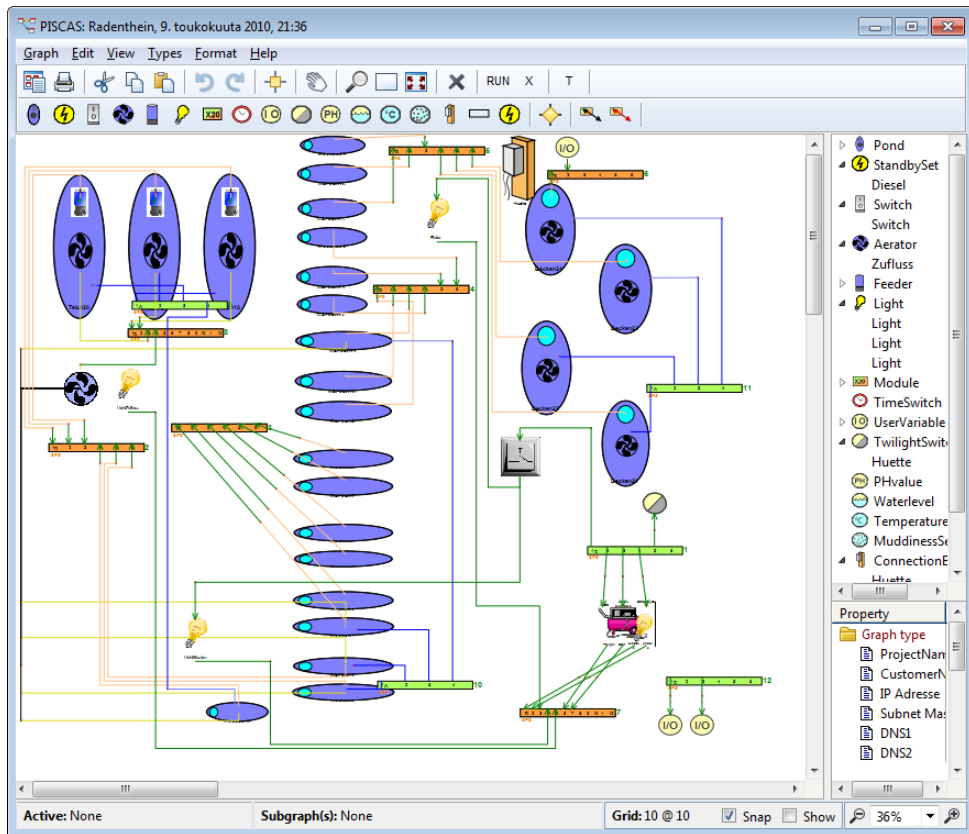
Embedded controller



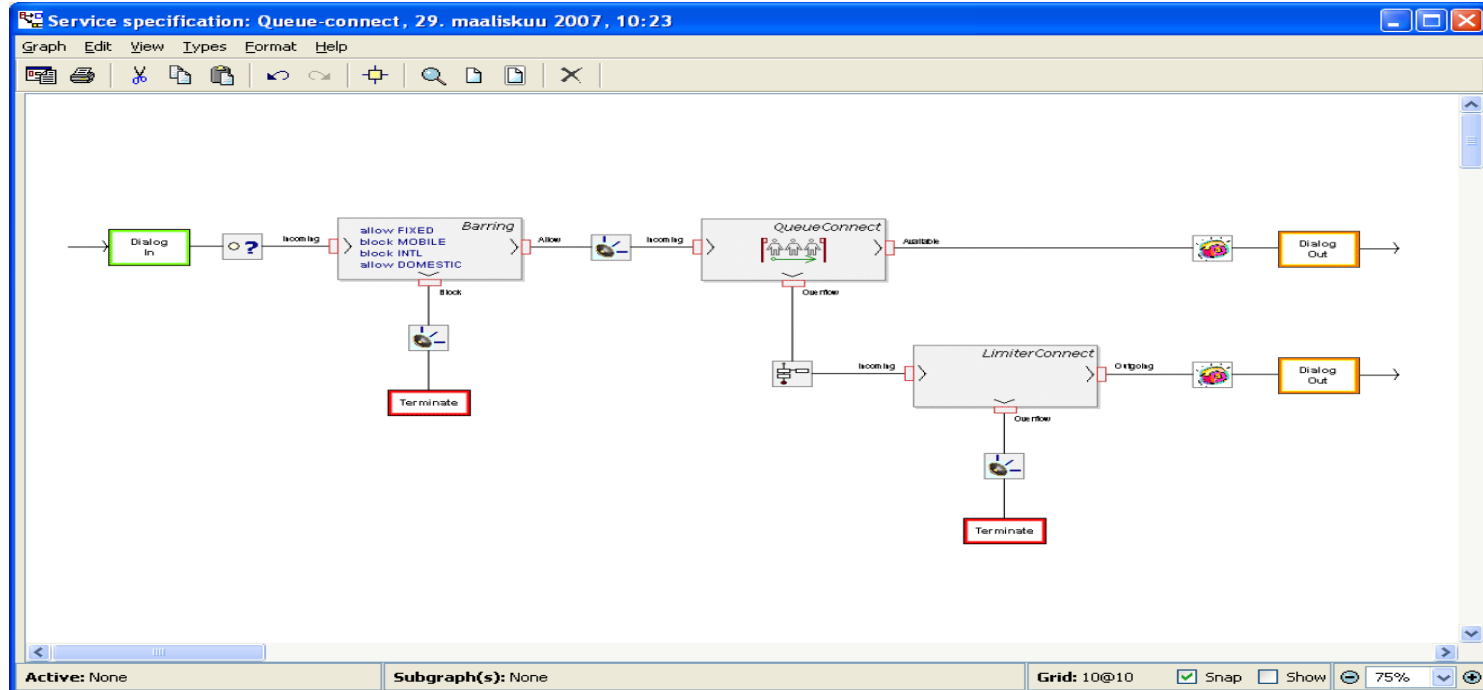
Microcontroller



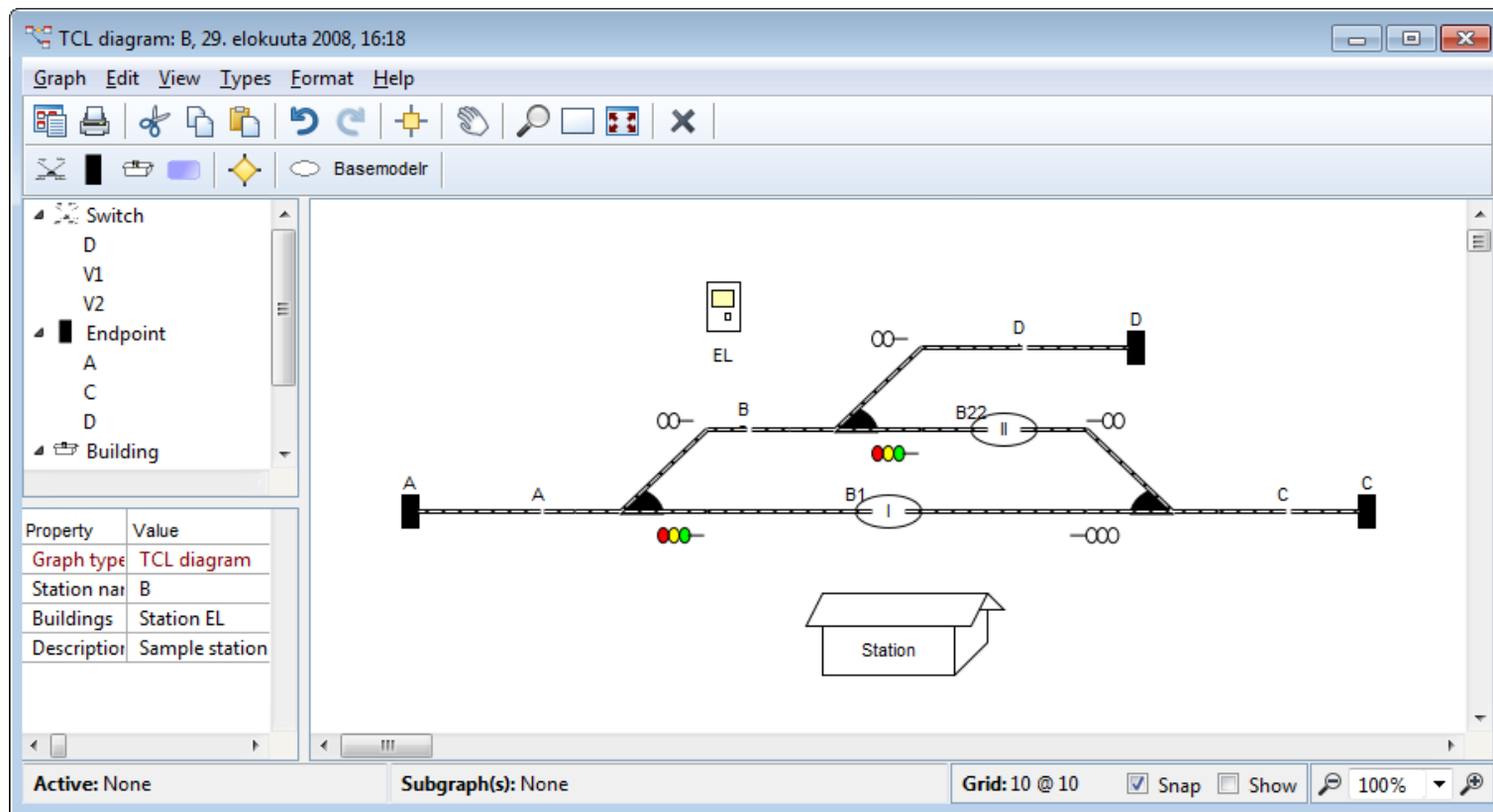




Telecom services

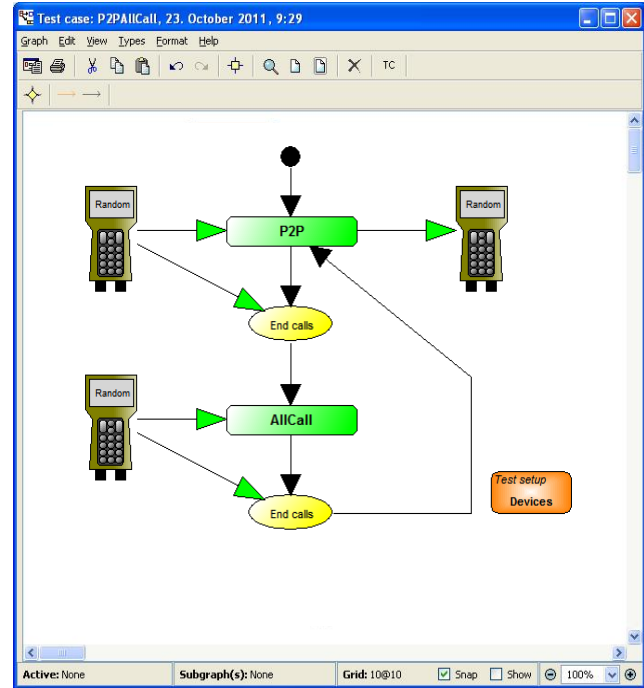
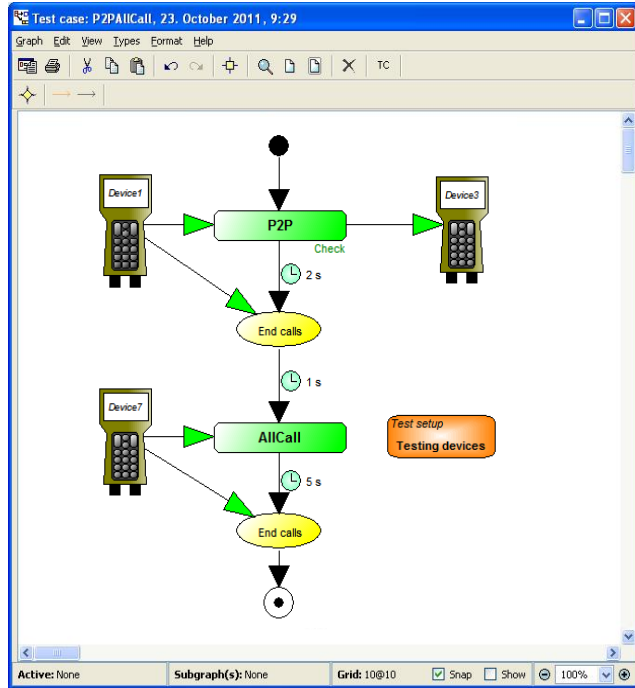


Railway interlocking

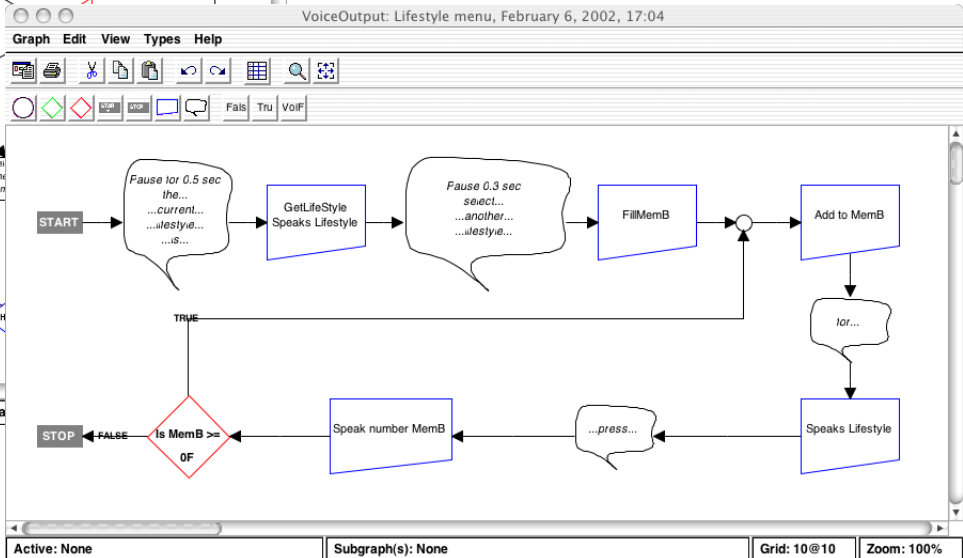
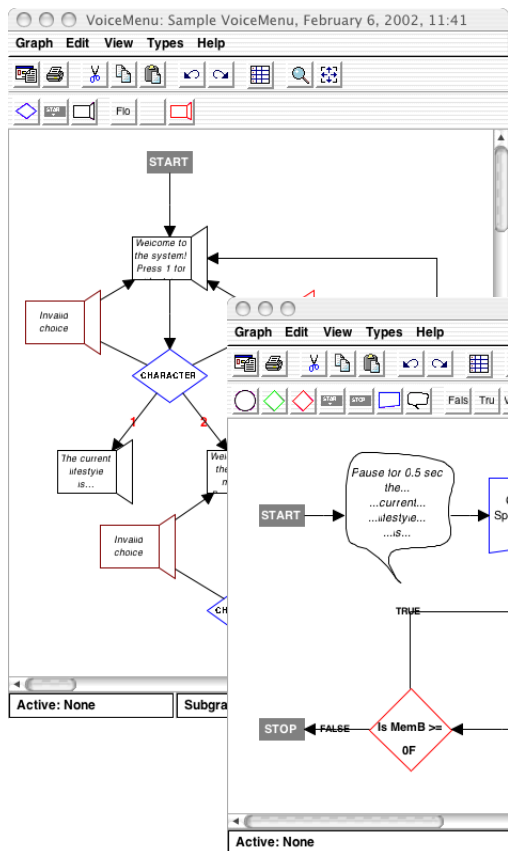


Radio network testing

- Modeling test cases/test logic and generating test data

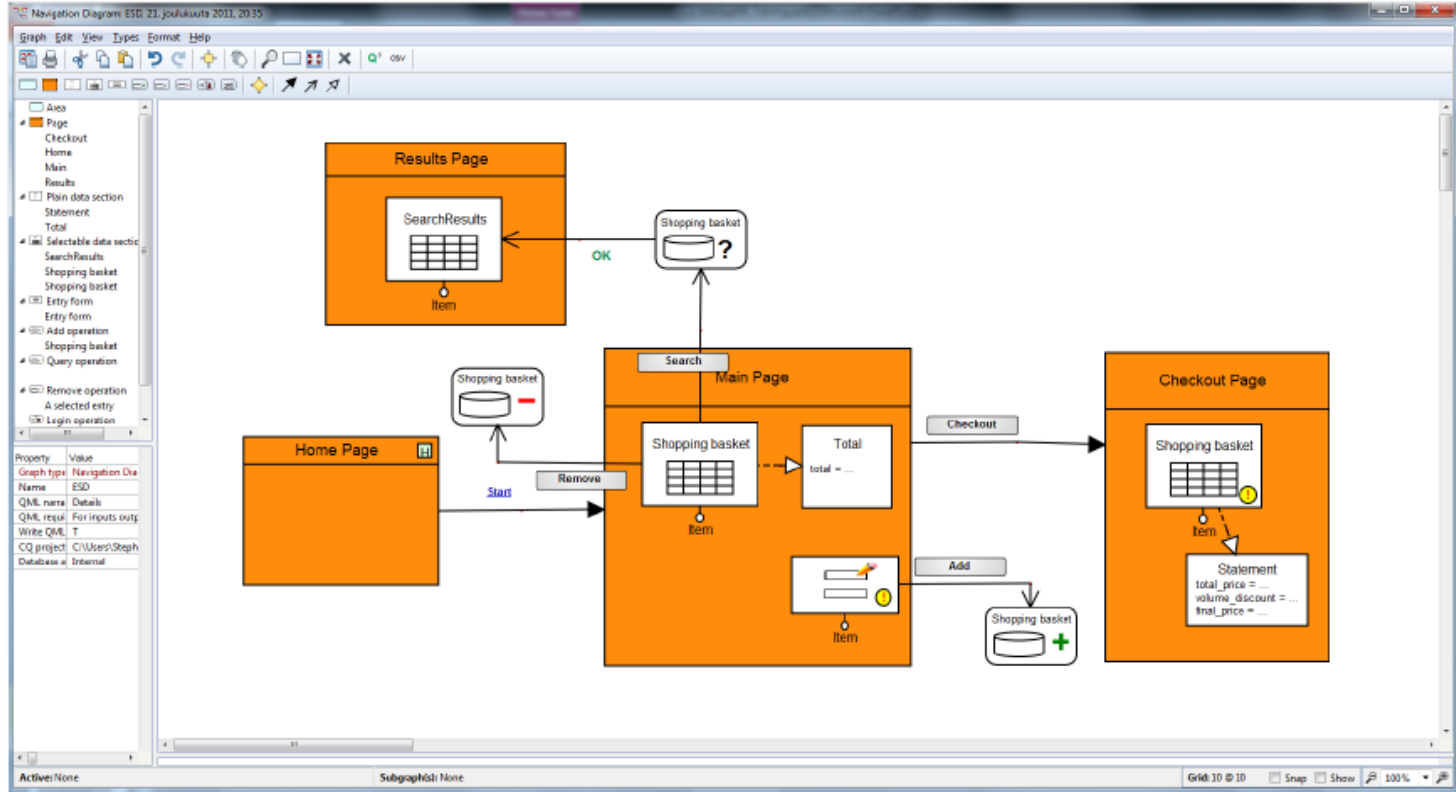


Voice control



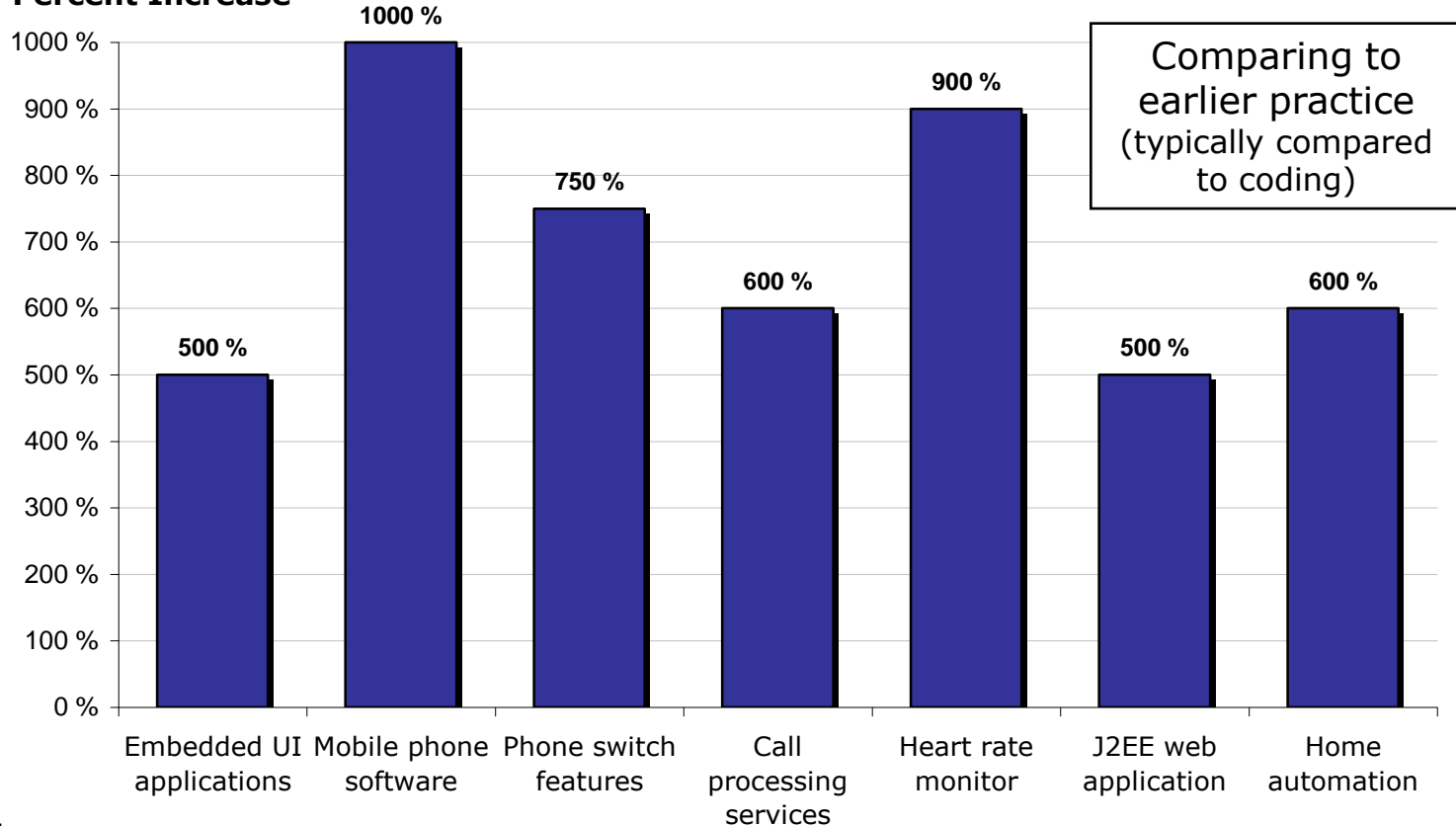
```
Report Output: Sample VoiceMenu: VoiceMenu
File Edit Help
Goto 3_266
:3_450
Speak 0x01 5 (Pause for 0.5 sec)
Speak 0x02 5 (the...)
Speak 0x03 5 (...current...)
Speak 0x04 5 (...lifestyle...)
Speak 0x05 5 (...is...)
GetLifeStyle
Speaks Lifestyle
Speak 0x06 5 (Pause 0.3 sec)
Speak 0x07 5 (select...)
Speak 0x08 5 (...another...)
Speak 0x04 5 (...lifestyle...)
FillMemB 00
:3_844
Add to MemB 01
Speak 0x09 5 (tor...)
Speaks Lifestyle
Speak (...press...)
Speak number MemB
Is MemB >= OF
IFNot
Goto3_844
:3_468
Speak 0x15 10 (Welcome to the other menu)
Speak 0x16 12 (Press 1 for the main menu)
Clear Menu Buffer
```

Web app testing



Productivity increase measured

Percent Increase



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



MetaCase

**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