SIEMENS

Anatomy of a domain-specific language project in an industrial context

Development and examination of a DSL demonstrator for elevator controllers

Christoph Wienands & Michael Golm
Software Engineering, Architecture and Platform Technologies
Siemens Corporate Research, Princeton, NJ

SIEMENS

Overview

- Motivation and goals
- Elevator controllers as research subject
- Domain analysis
- Inception of new, abstract domain concepts
- DSL Tooling Architecture
- Meta-model
- Encountered challenges
- Support for another target platform (C) and code generator optimization
- Collected metrics

SIEMENS

Background of DSL project for elevator controllers

Motivation

- Introduction of DSLs into organizations is difficult.
  - Learning curve
  - Language and model evolution
  - Initial investment high
- Development of visual DSLs typically labor-intensive task.
  - Limited experience of DSL authors
  - Difficulties finding appropriate visual representations
  - Complexity: Multi-diagram DSLs, constraints, customization, visual-text hybrids
- Generated code said to perform poorly.

Pretty much everything Steve Mellor said in Keynote

Goals

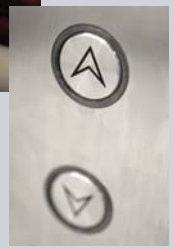
- Gain more experience building DSL, especially using Eclipse GMF.
- Collect metrics throughout development and evaluate them.
- Examine challenge of generation of performant code.
- Develop a fully documented, easy-to-understand, publishable DSL and tool chain for research purposes and as a showcase.

SIEMENS

Elevator controllers – Research subject for DSLs

- Elevators are grouped in elevator banks
- High-level control consists of: Opening/closing doors, accelerating/decelerating individual cars, and reacting to floor calls/car calls (buttons pressed inside cars and outside on floors.
- Floor calls require the concept of committed direction: The direction an elevator car will continue in after opening and closing the doors at the next stop.
- Support for multiple special modes: Maintenance, fire-fighter mode, etc.

→Elevator controller domain is non-trivial and provides enough variability to justify the development of a DSL, yet it is strictly scoped.



SIEMENS

Domain analysis of elevator controllers

Elevator controller functionality is separated into:

- Common behavior shared across all controllers
  - Accelerating / decelerating to reach target floor
  - Closing of doors: Closing delay, reopen if obstruction detected
  - Releasing of pressed buttons upon reaching target floor
- Variable, mode-specific behavior:
  - Determining next target floor
  - Setting of committed direction
  - Decision to open doors (only if stopped)

→Common behavior is mostly safety-critical and will not be part of the modeling language.

→Actions strongly depend on state of elevator components and events from subsystem.

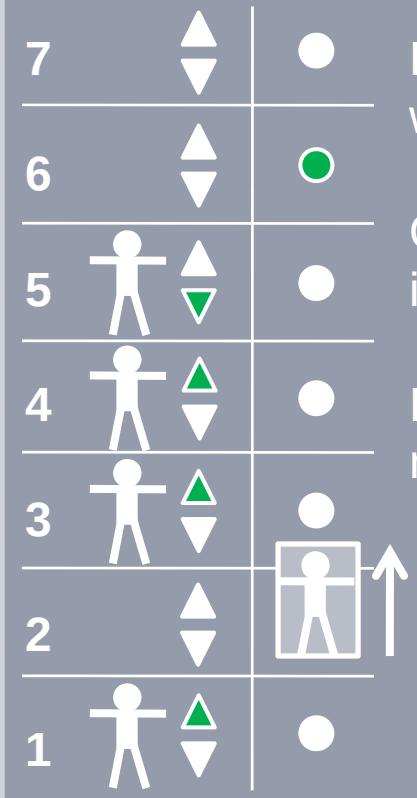
→Modeling language based on state machines seems most suited.

SIEMENS

Advanced domain concepts in elevator controller DSL

- **2 named call lists**
  - Dynamic lists providing trigger events 'Added' and 'Removed' to state machine transitions
  - *Elevator calls*: Buttons pressed inside car
  - *Assigned floor calls*: Assigned by runtime
- **Call list projections**
  - Dynamic filter queries.
  - Filter configurations: Calls ahead, calls behind, all calls, same direction, opposite direction.
  - Filter result depends on current position, speed, acceleration and committed direction.
  - Can combine calls from *ElevatorCalls* and *AssignedFloorCalls.*
  - Provide events just like underlying call lists.

Call List Projection Example
"*CallsAhead – SameDirection*"



Four persons waiting on floors

One passenger in car, target floor 6

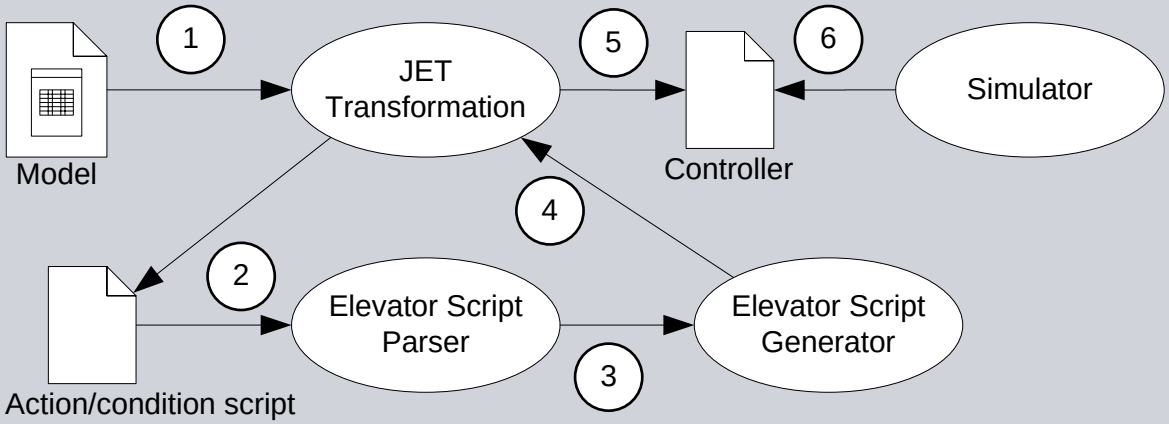
Elevator speed 15ft/s, next possible stop 4

Result of call list projection query: 4, 6

SIEMENS

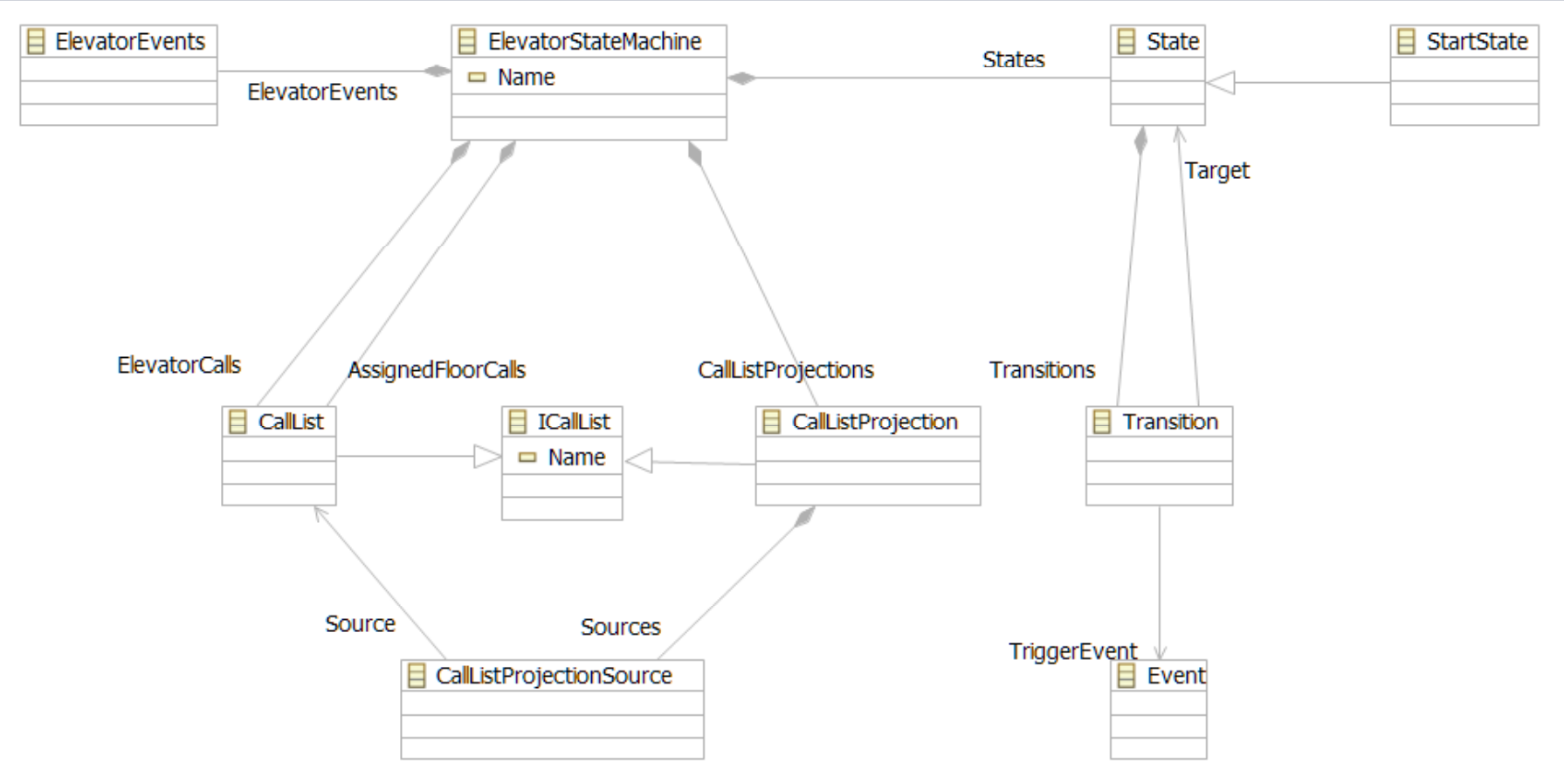
DSL Tooling Architecture

Technology	Description
Graphical Modeling Framework (GMF)	Eclipse-based workbench for visual DSLs
Eclipse Modeling Framework (EMF)	Underlying (meta) model library
Graphical Editing Framework (GEF)	Underlying library for editors
Java Emitter Templates (JET)	Code generation, model-to-text
openArchitectureWare Xtext	Parser for mini scripting language
openArchitectureWare Xpand	Code generation for scripting language



SIEMENS

Meta-model

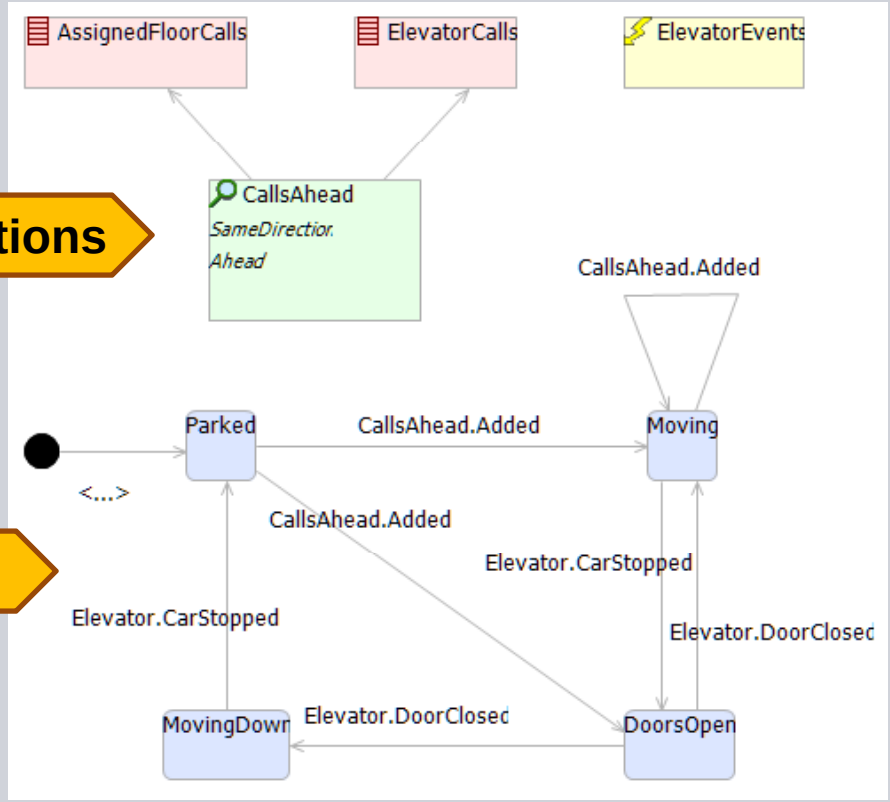


Example of ‘Up-Only’ elevator controller

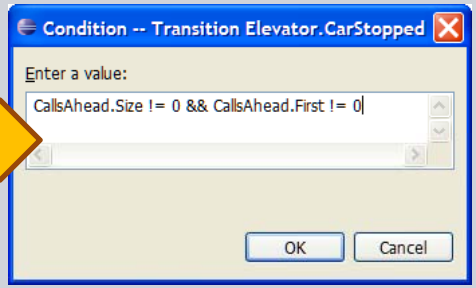
Call lists

Call list projections

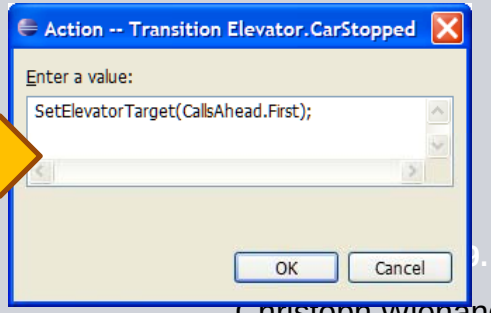
State machine



Conditions



Actions



SIEMENS

Encountered challenges

- Steep learning curve for GMF.
- Implementing an efficient concrete syntax:
  - Weak support for decomposition (subdiagram of different type) in GMF
  - Complexity of mapping ecore meta model to concrete syntax due to EMF flexibility (trade-off)
- Customization of DSL editor:
  - Implemented customizations
    - Read-only, calculated attribute
    - Propagate attribute and reference change notifications
    - Initialize model graphs and prepopulate diagrams upon creation
    - Context menu
    - Integration visual DSL and textual scripting language
  - Most customizations needed to be done manually by modifying or supplementing generated code. Little tool support.

Support for alternative C platform: Optimizations performed by code generator

SIEMENS

- Statically allocated arrays

→Constant memory footprint

- Statically linked event chains
- *Not shown*: Optimized filter queries for call list projections

→ Performant execution of state machine

- Conditional exclusion of status updates

→ Faster execution for simple controllers

```
double* m_ElevatorSpeed;
m_ElevatorSpeed = new double[m_ElevatorCount];
```

```
void tick() {
  … elevatorCallsAdded(i, j); …
}

void elevatorCallsAdded(int elevator, int floor) {
  … elevatorCallsAheadAdded(elevator, floor); …
}

void elevatorCallsAheadAdded(int elevator, int floor)
{
  … if (m_State[elevator] == STATE_MOVING &&
        movingCallsAheadAdded4Condition(elevator)) {
          movingCallsAheadAdded4Action(elevator);
    } …
}
```

```
for (int i = 0; i < m_ElevatorCount; i++) {
#ifdef STATUS_SPEED
  m_ElevatorSpeed[i] = m_Sim->getElevatorSpeed(i);
#endif
}
```

SIEMENS

Some Collected Metrics

Effort for creation of a elevator controller by domain expert vs. domain novice

Development method	Effort expert (hours)	Effort novice (hours)
Using Elevator DSL (therefore works on both target platforms)	1.0	5.0
Manually developed against C domain framework	1.5	-
Manually developed for C platform (no domain framework)	4.0	8.0+ (out of time, believed to be at least 12h until complete)

Efficiency of generated code

Metric	DSL-based controller	Manually dev. controller
Static analysis		
LOC	791 domain framework + 462 generated controller = 1253 total	508 controller
Binary size	56kB	40kB
Implementation effort	4 call list projections, 5 states, 13 transitions and 29 lines of TSL script (conditions, actions)	508 lines of C code
Runtime analysis		
Avg. instructions/cycle	184	223

SIEMENS

Conclusion and future work

Findings

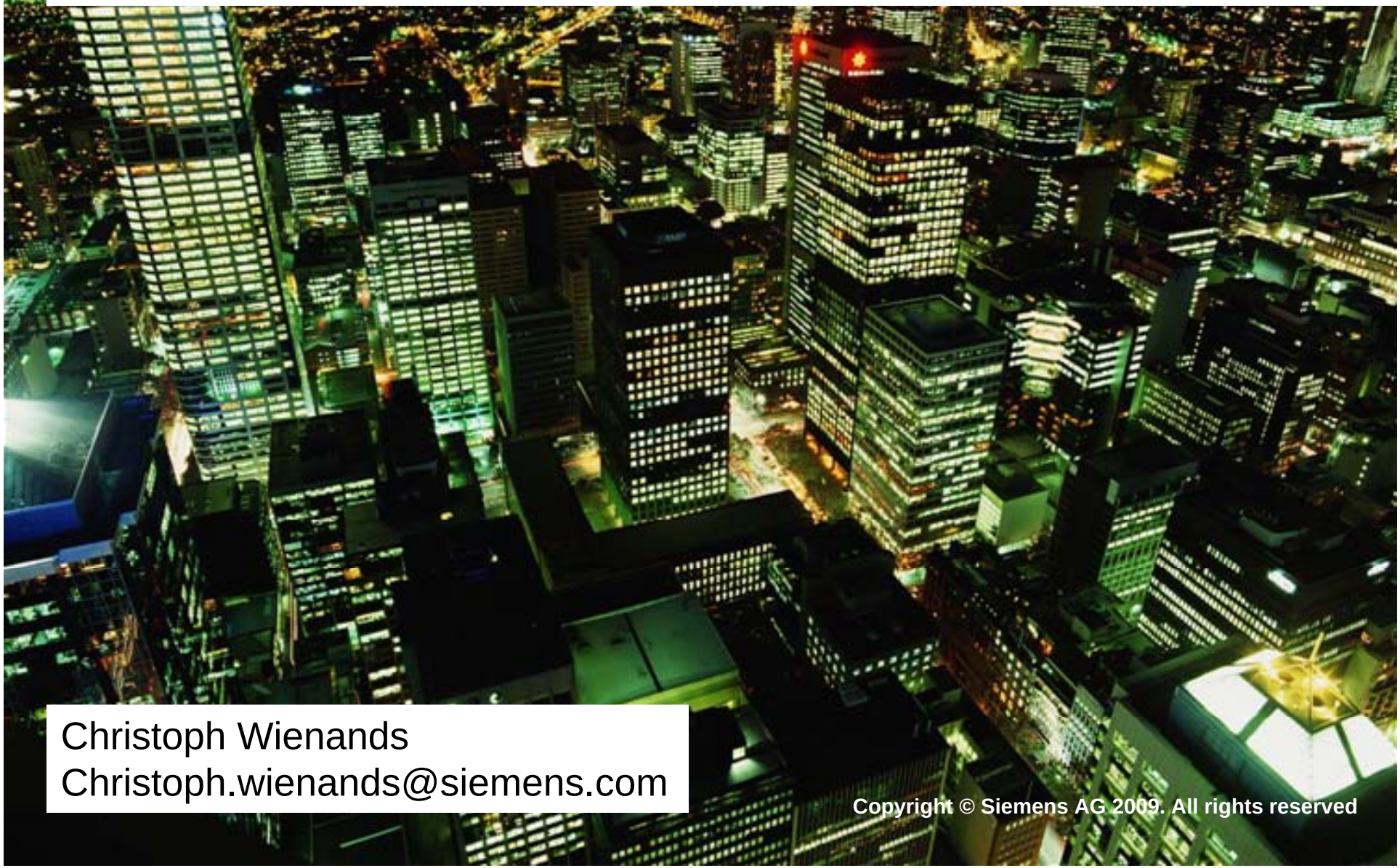
- Domain analysis and prototyping yielded two domain concepts not found in existing elevator controllers
  → raising level of abstraction for users.
- Used many Java/Eclipse modeling technologies to create user-friendly DSL
  → additional experience
- Quantitative analysis of collected metrics support previous findings that DSLs increase productivity.

Future work

- Port DSL to other visual DSL workbenches for comparison
- Extend DSL to support modeling call distribution algorithms across elevator bank.

SIEMENS

Thank you for your attention



Christoph Wienands
Christoph.wienands@siemens.com