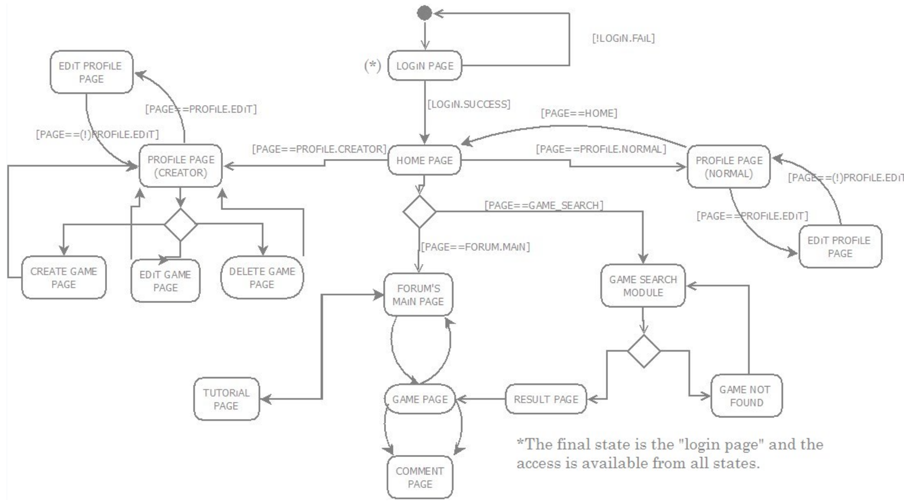
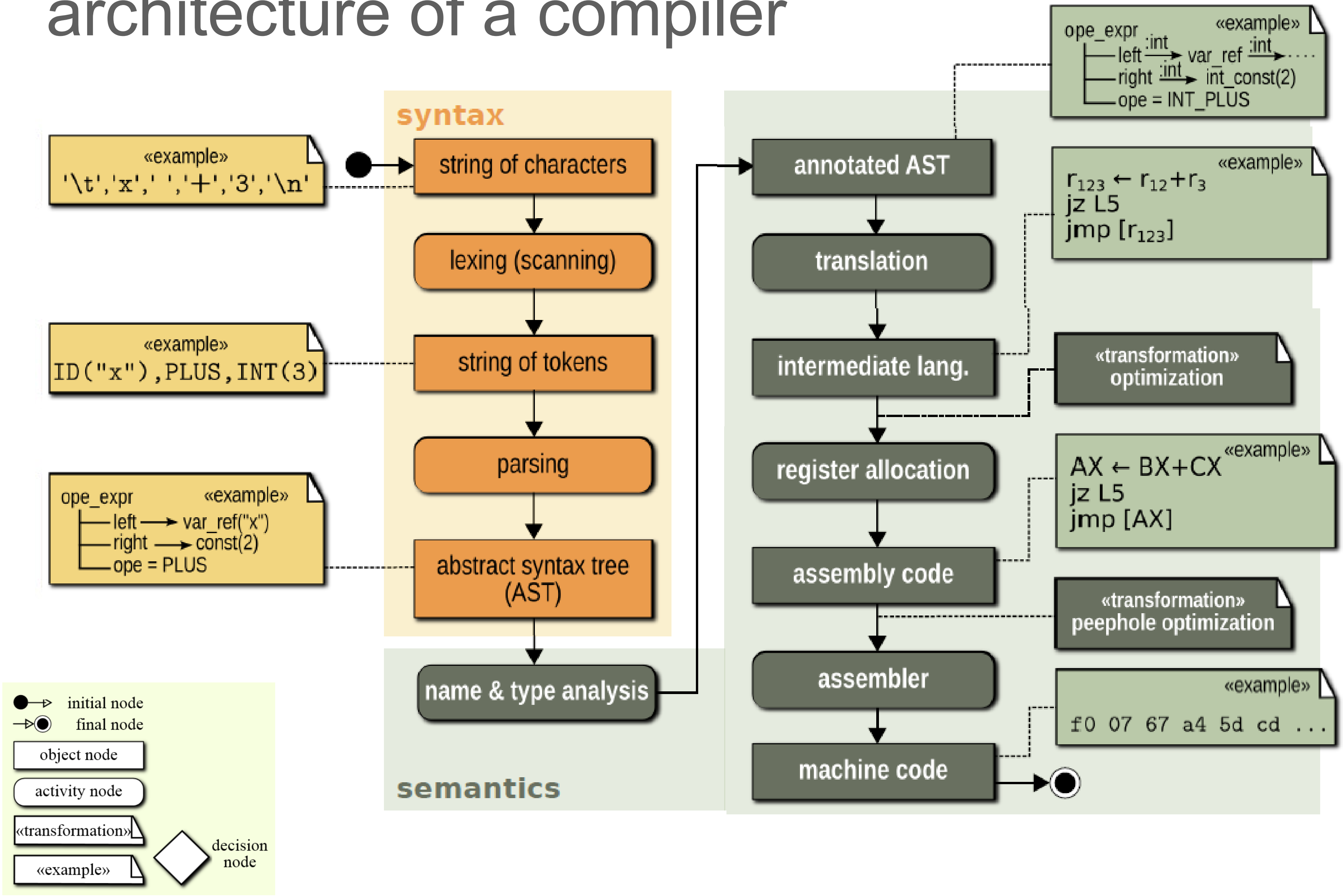


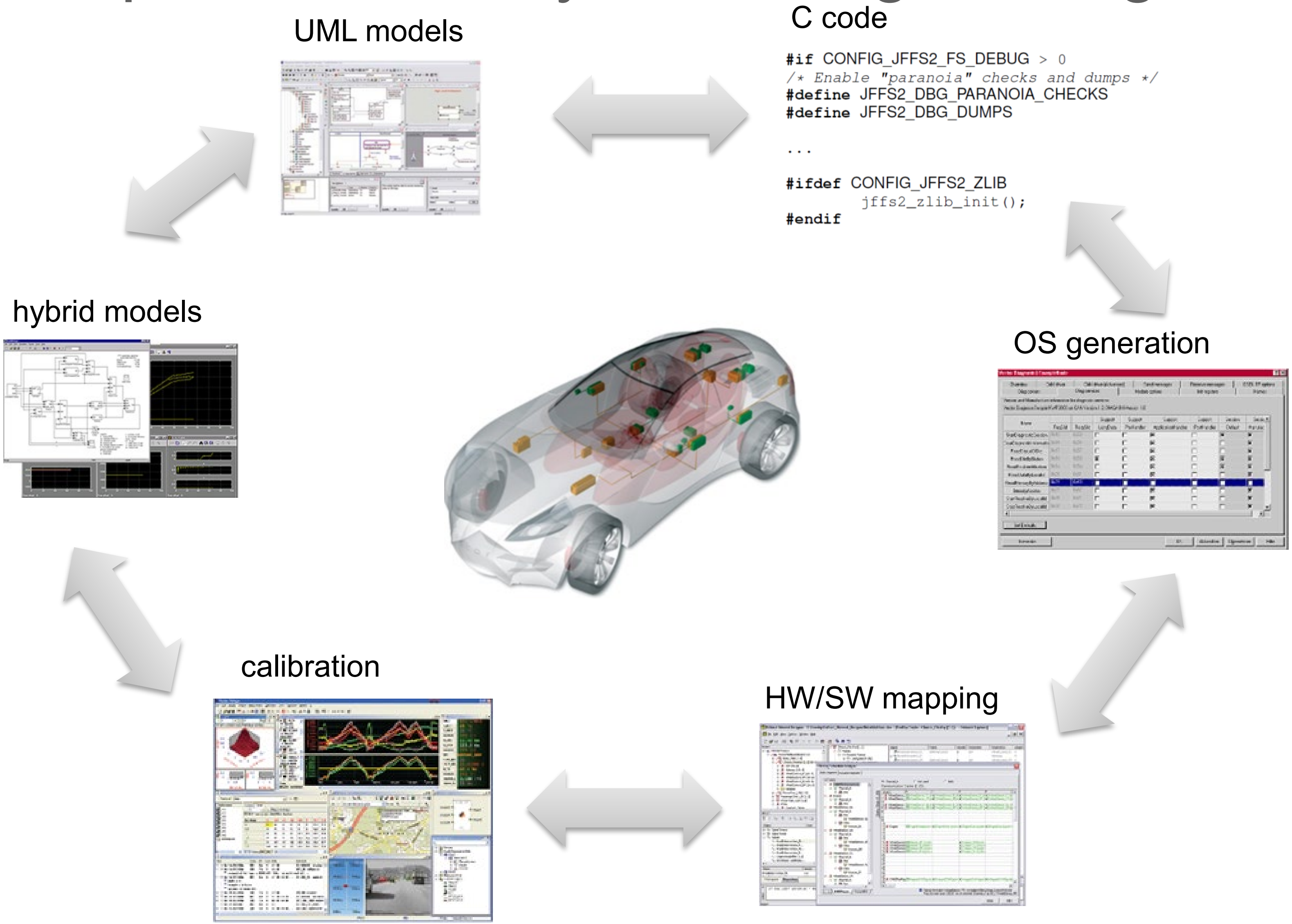
Domain-Specific Languages (DSLs)
motivation, concepts, examples

Thorsten Berger
Associate Professor
thorsten.berger@chalmers.se

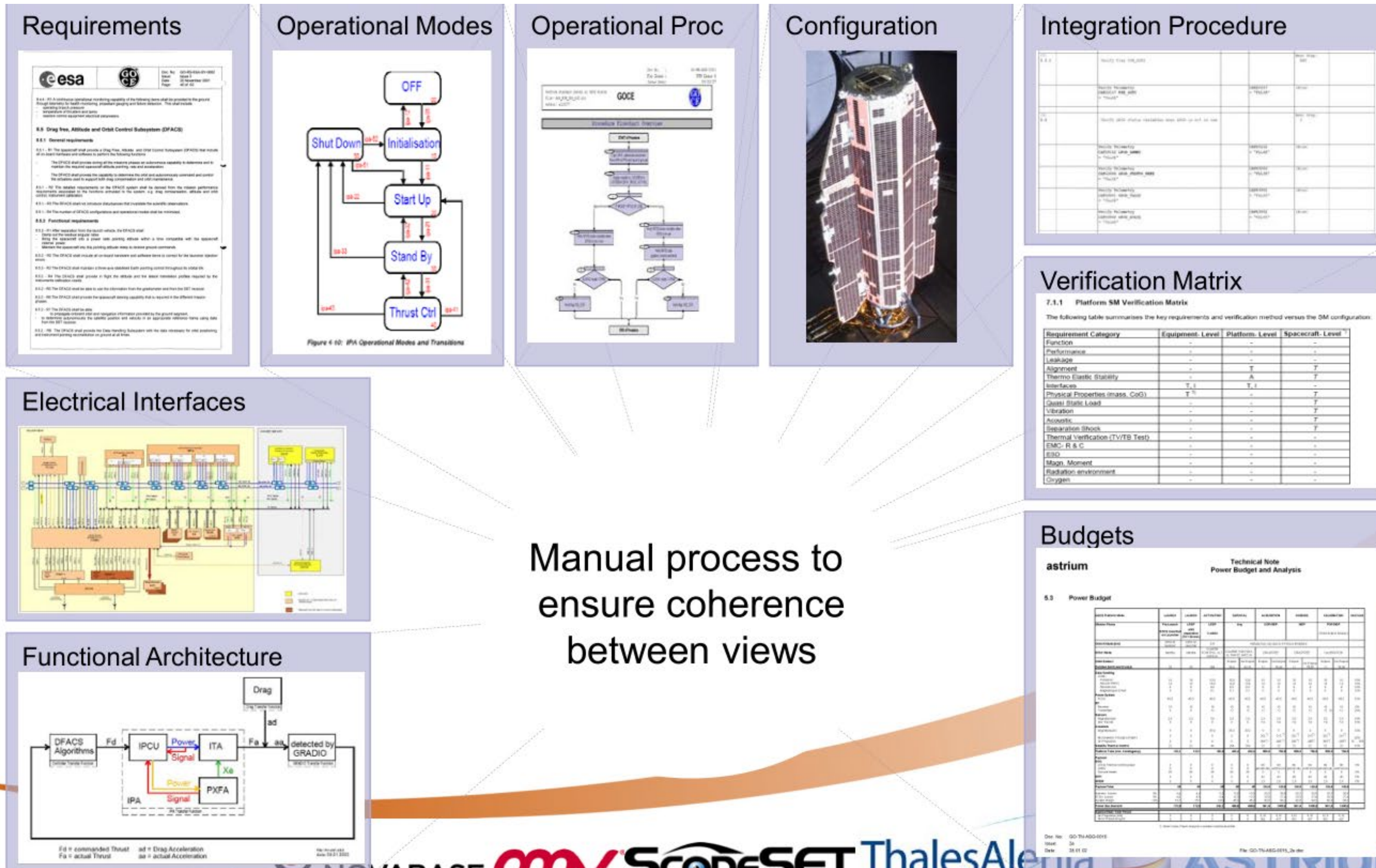
architecture of a compiler



a perspective from systems engineering



www.vsd-project.org

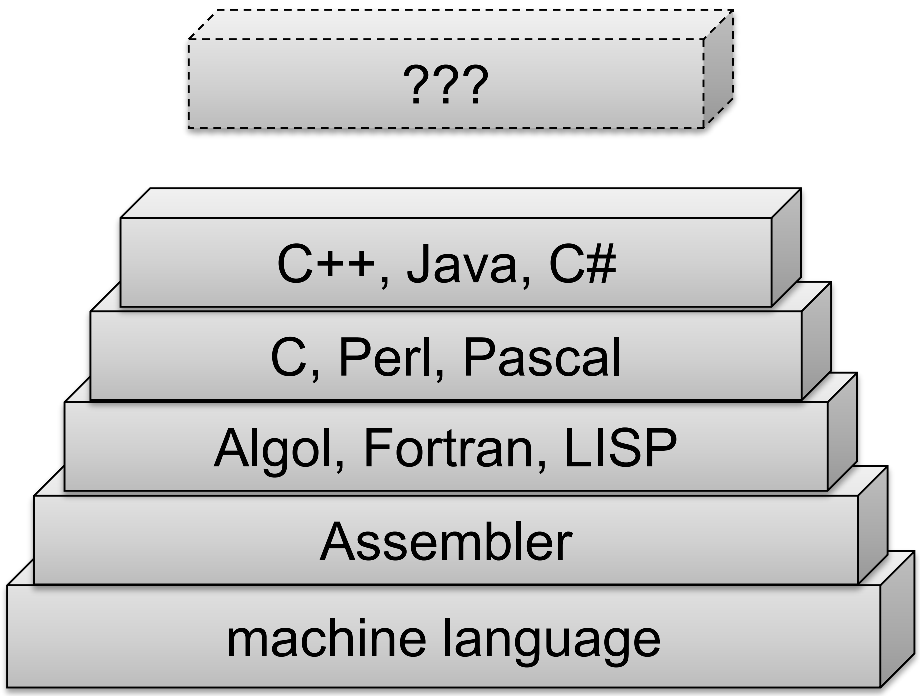


HISTORY OF LANGUAGES

continuous abstractions

language

concepts



objekt-orientation: classes, objects

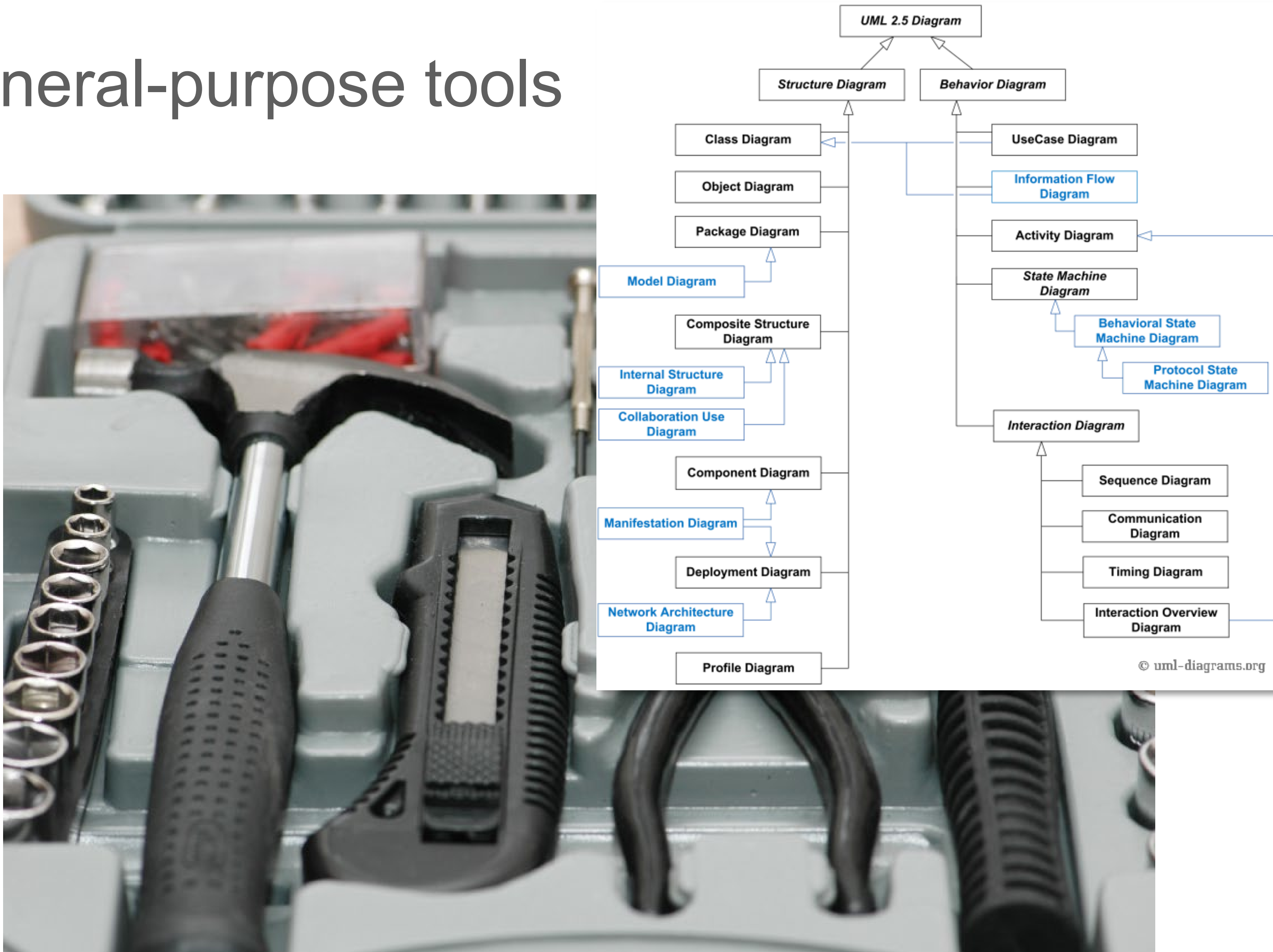
modularization, functions

conditionals, loops

jumps, arithmetic expressions

count, read register, I/O

general-purpose tools



domain-specific tools

focus on the domain!

© Zsolt Bota Finna / fotolia.com

© Europäische Union, [2010] – EP

tailored towards domain requirements
effective within the domain
both specialized and limited
used by domain experts

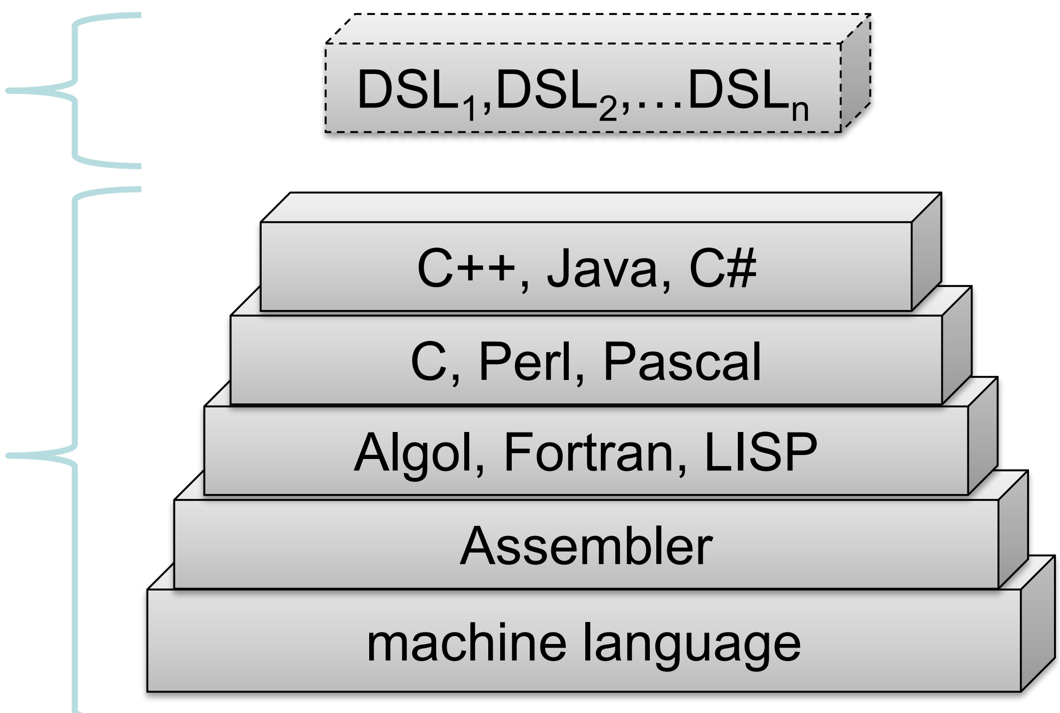
use domain-specific languages!

expressiveness | language | concepts

Domain-Specific Language (DSL)

∧

General-Purpose Language (GPL)



arbitrary domain concepts

objekt-orientation: classes, objects

modularization, functions

conditionals, loops

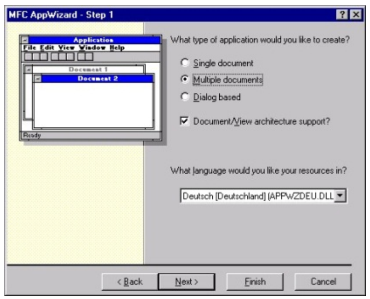
jumps, arithmetic expressions

count, read register, I/O

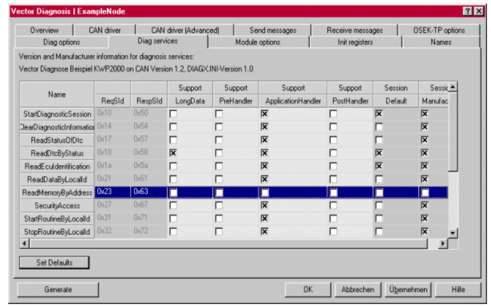
domain-specific language

- advantages
  - less expressive
  - separates domain-related and infrastructure code
  - improves communication with domain experts and customers
- DSLs exist for a long time, such as:
  - regular expressions: `[-+]?[0-9]*\.?[0-9]+`
  - SQL: **`SELECT ... FROM ... WHERE ...`**
  - CSS/HTML: b{ color: #926C41 }
- recent improvements in language technology help
  - to quickly create your own DSL
  - language workbenches instead of CASE tools
  - model-transformation languages
- disadvantage: tools are still somewhat complex

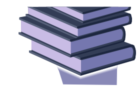
kinds of DSLs



wizard

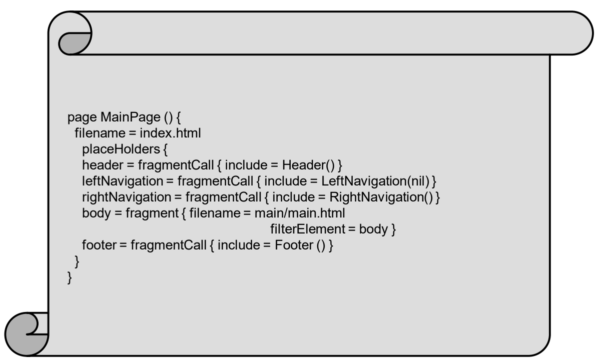


configuration tool

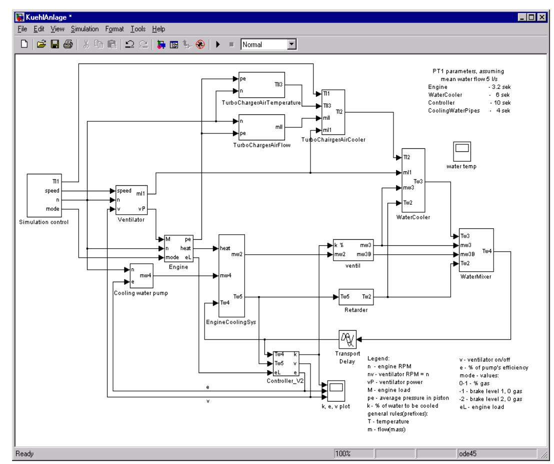


library in a programming language

...



textual DSL



visual DSL

EXAMPLES

INTERNAL (EMBEDDED) DSLS

UI programming in Java

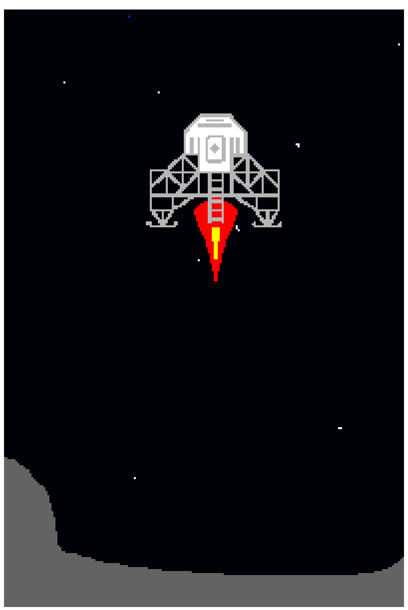
Java

```java
Display display = Display.getDefault();
Shell shell = new Shell(display);
shell.setText("SWT");
shell.setLayout(new FillLayout());
Label label = new Label(composite, SWT.NONE);
label.setText("Hello World!");
shell.pack();
shell.open();
while(!display.isDisposed()){
  if(!display.readAndDispatch()){
    display.sleep();
  }
}
display.dispose();
```

JRuby with Glimmer DSL

```ruby
shell {
  text "SWT"
  label {
    text "Hello World!"
  }
}
```

Lunar lander

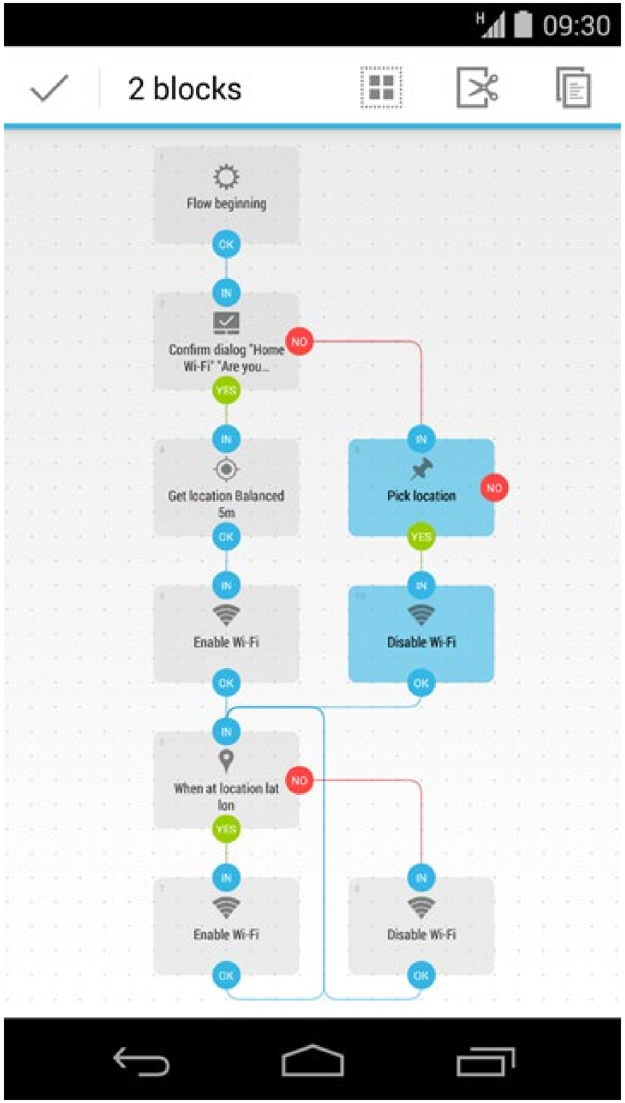


```
object Lunar extends Baysick {
  def main(args:Array[String]) = {
    10 PRINT "Welcome to Baysick Lunar Lander v0.9"
    20 LET ('dist := 100)
    30 LET ('v := 1)
    40 LET ('fuel := 1000)
    50 LET ('mass := 1000)

    60 PRINT "You are drifting towards the moon."
    70 PRINT "You must decide how much fuel to burn."
    80 PRINT "To accelerate enter a positive number"
    90 PRINT "To decelerate a negative"

    100 PRINT "Distance " % 'dist % "km, " % "Velocity
    110 INPUT 'burn
    120 IF ABS('burn) <= 'fuel THEN 150
    130 PRINT "You don't have that much fuel"
    140 GOTO 100
    150 LET ('v := 'v + 'burn * 10 / ('fuel + 'mass))
    160 LET ('fuel := 'fuel - ABS('burn))
    170 LET ('dist := 'dist - 'v)
    180 IF 'dist > 0 THEN 100
    190 PRINT "You have hit the surface"
    200 IF 'v < 3 THEN 240
    210 PRINT "Hit surface too fast (" % 'v % ")km/s"
    220 PRINT "You Crashed!"
    230 GOTO 250
    240 PRINT "Well done"

    250 END

    RUN
  }
}
```

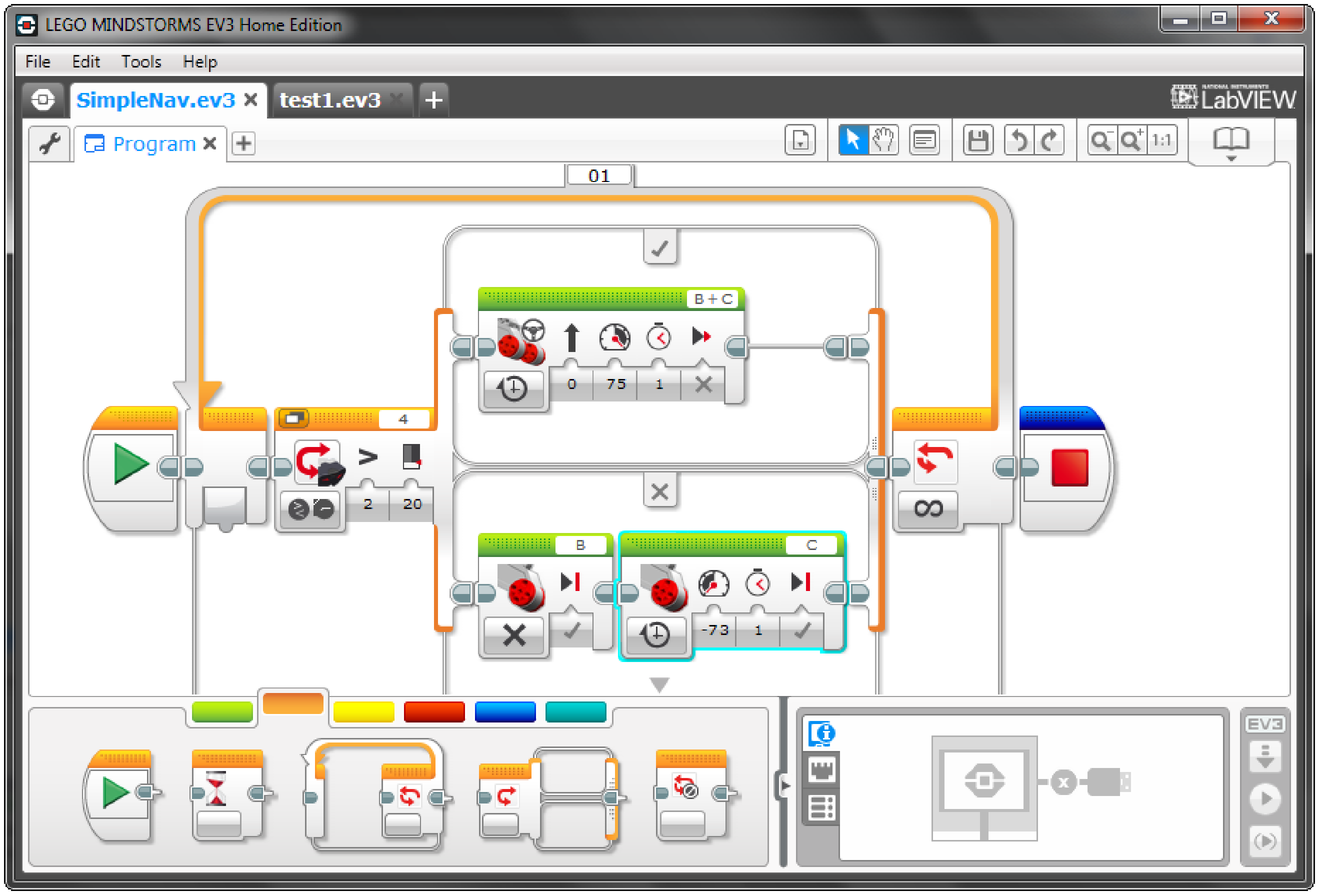
https://www.scala-lang.org/old/node/1403

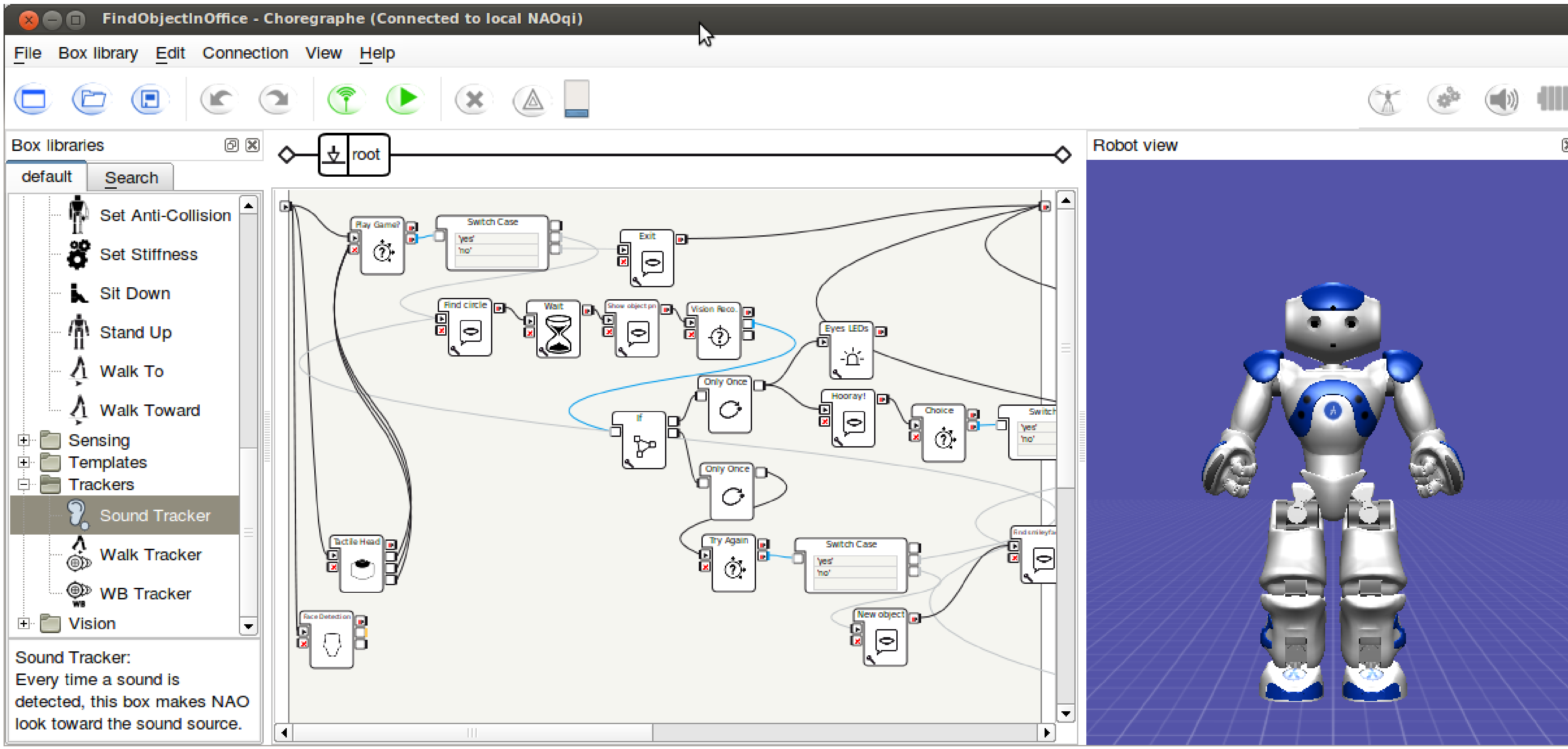
EXTERNAL GRAPHICAL DSLS

Automate (Android App)



Lego Mindstorms Robots



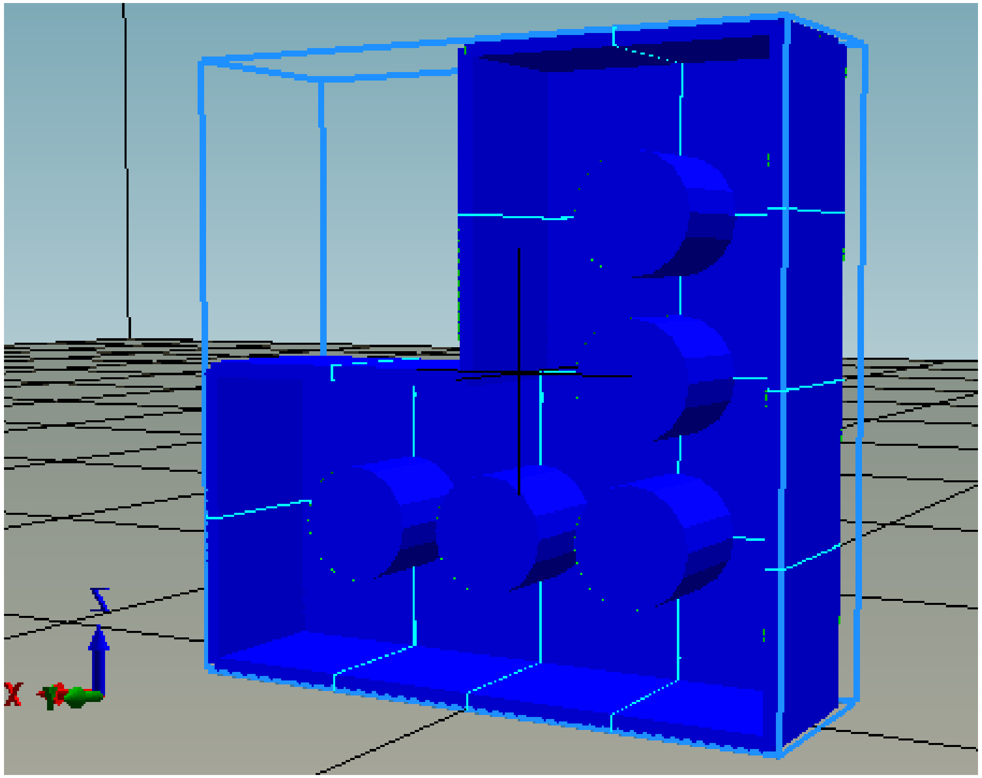

FindObjectInOffice - Choregraphe (Connected to local NAOqi)
File
Box library
Edit
Connection
View
Help
Box libraries
default
Search
Set Anti-Collision
Set Stiffness
Sit Down
Stand Up
Walk To
Walk Toward
Sensing
Templates
Trackers
Sound Tracker
Walk Tracker
WB Tracker
Vision
Sound Tracker:
Every time a sound is detected, this box makes NAO look toward the sound source.
root
Robot view

EXTERNAL TEXTUAL DSLS

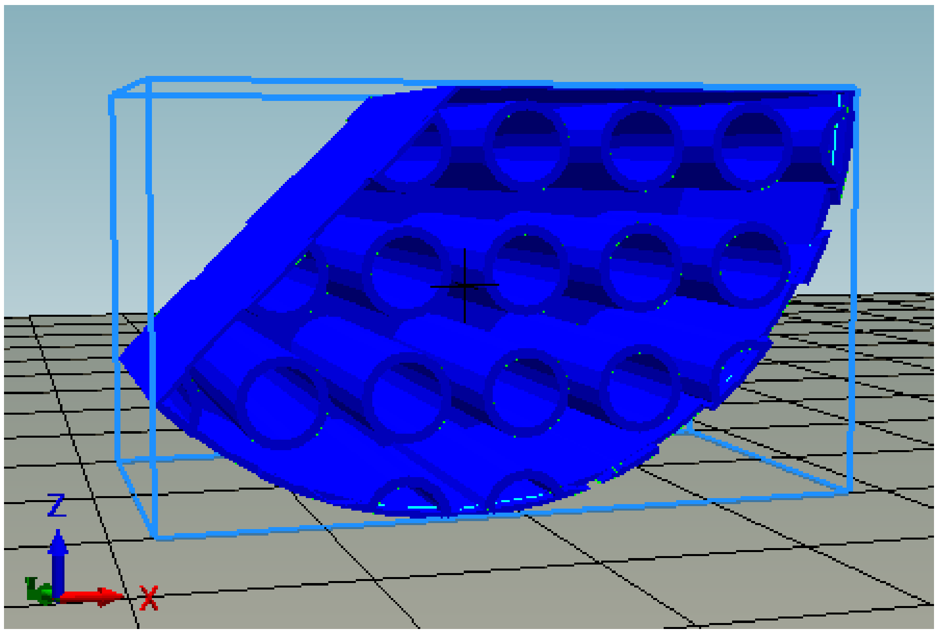
Google Protocol Buffers

```
message Person {

   enum PhType { MOBILE = 0; WORK = 1; }

   message PhoneNo {
      required string no = 1;
      optional PhType type = 2 [default = MOBILE];
   }

   required string name = 1;
   repeated PhoneNo phone = 2;
}
```

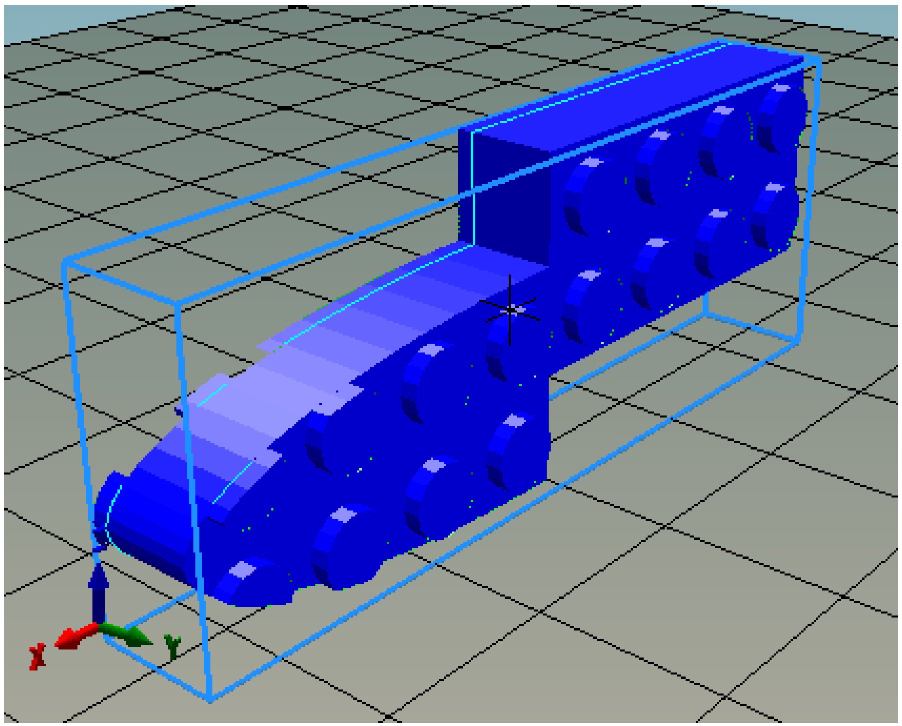
developed in the MDE course



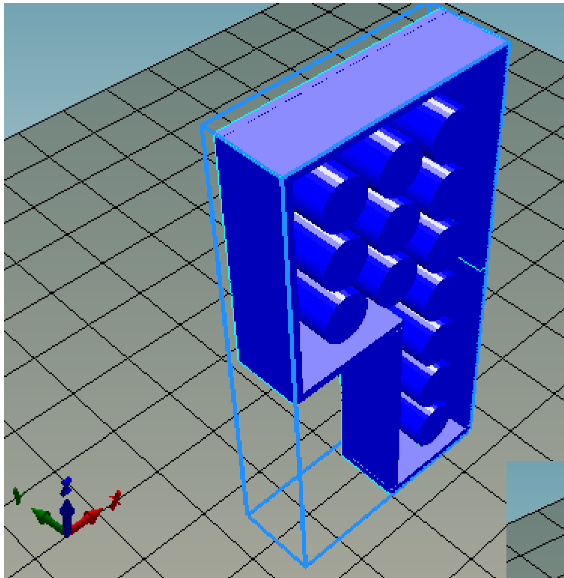

```
{
        "Lego": "Star",
        "Length": 20,
        "Width": 20,
        "Bricks" : [{
                "Brick": "Wars",
                "Width": [4],
                "Length": [2]
        }, {
                "Brick": "Trek",
                "Width": [2],
                "Length": [2]
        }]
}
```



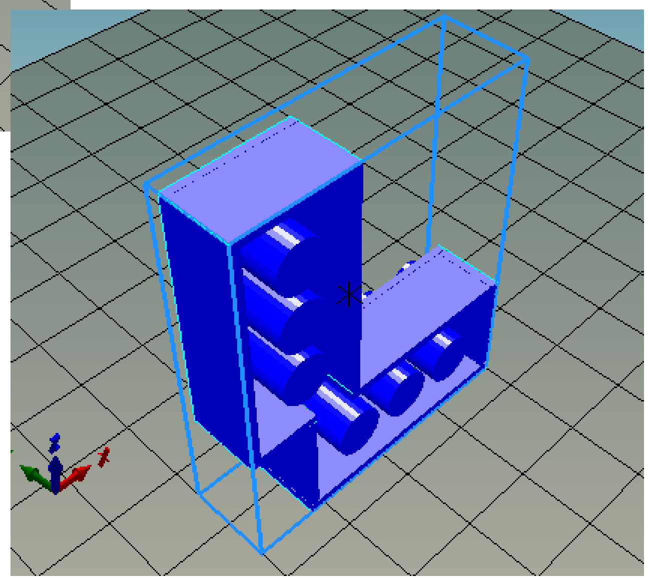

```
AdjLegoSystem {
        thickness 20
        finalBrick Pizza
        abstractlegobrick {
                RoundedBrick Pizza{
                        roundedSide ALL
                        sizeproperties {
                                int length = 7,
                                int width = 7
                        }
                },
                SlicedBrick Slice {
                        portions 3
                        brick Pizza
                }
        }

}
```

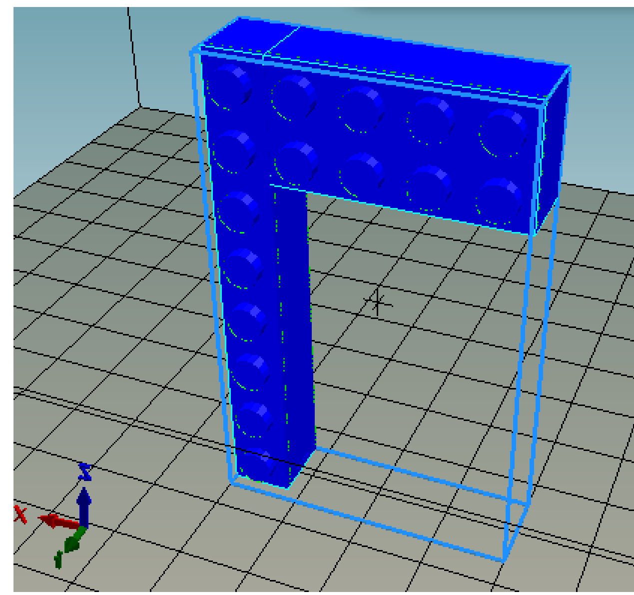



```
AdjLegoSystem {
        thickness 7
        finalBrick Boomerang
        abstractlegobrick {
                RoundedBrick Frisbee{
                        roundedSide RIGHT
                        sizeproperties {
                                int length = 4,
                                int width = 2
                        }
                },

                SquareBrick Stick {
                        sizeproperties {
                                int length = 4,
                                int width = 2
                        }
                },

                Combination Boomerang {
                        mainSide LEFT
                        position 3
                        main Frisbee
                        secondary Stick
                }
}
```

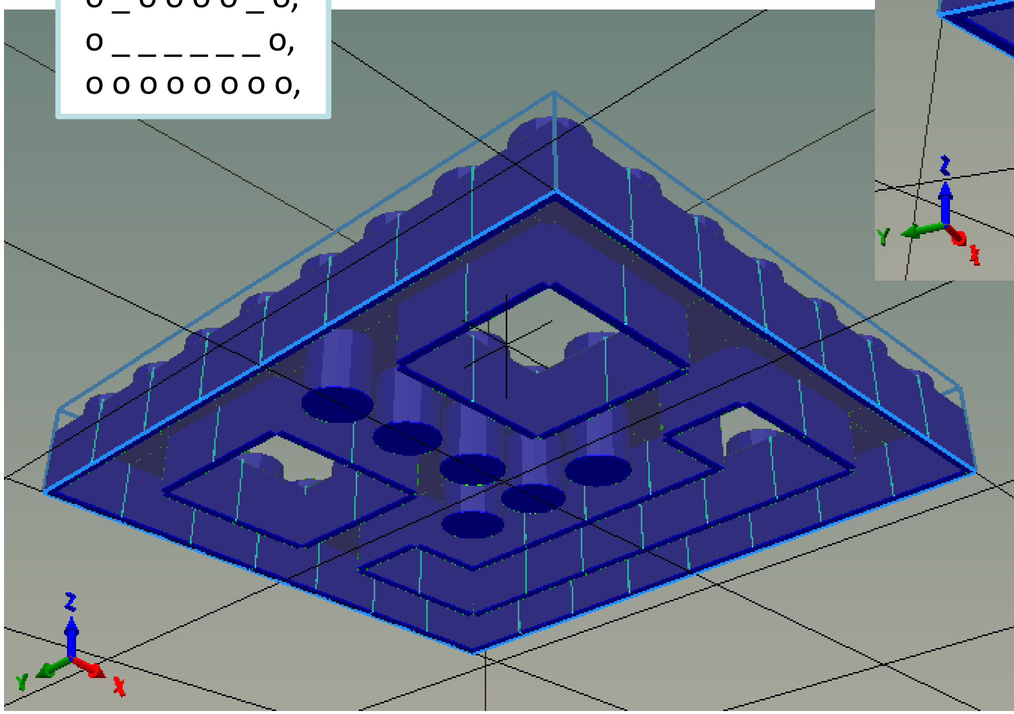


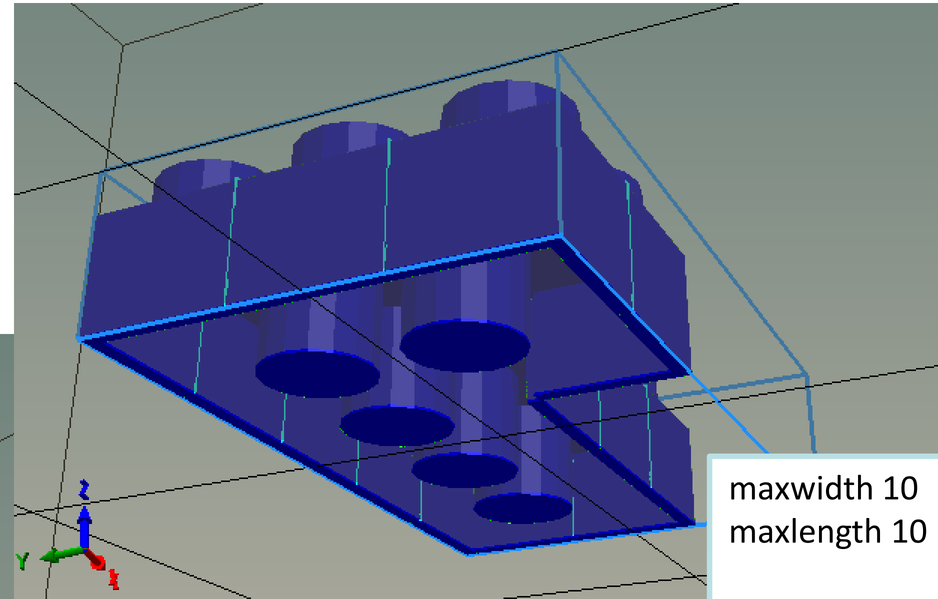
```
dimensions 10 x 10;
"2x4": 2 x 4;
"4x2": 4 x 2;
"1x8": 1 x (2 * "4x4".width);
"4x4": "2x4".height x (2 * "2x4".width);
"Composite Brick 1": "2x4" <- "4x4" TOP: LEFT 1 <- LEFT 1;
"Composite Brick 2": "2x4" <- "4x2" BOTTOM: LEFT 1 <- LEFT 4;
"Composite Brick 3": "1x8" <- "4x2" RIGHT: TOP 1 <- BOTTOM
2;
```





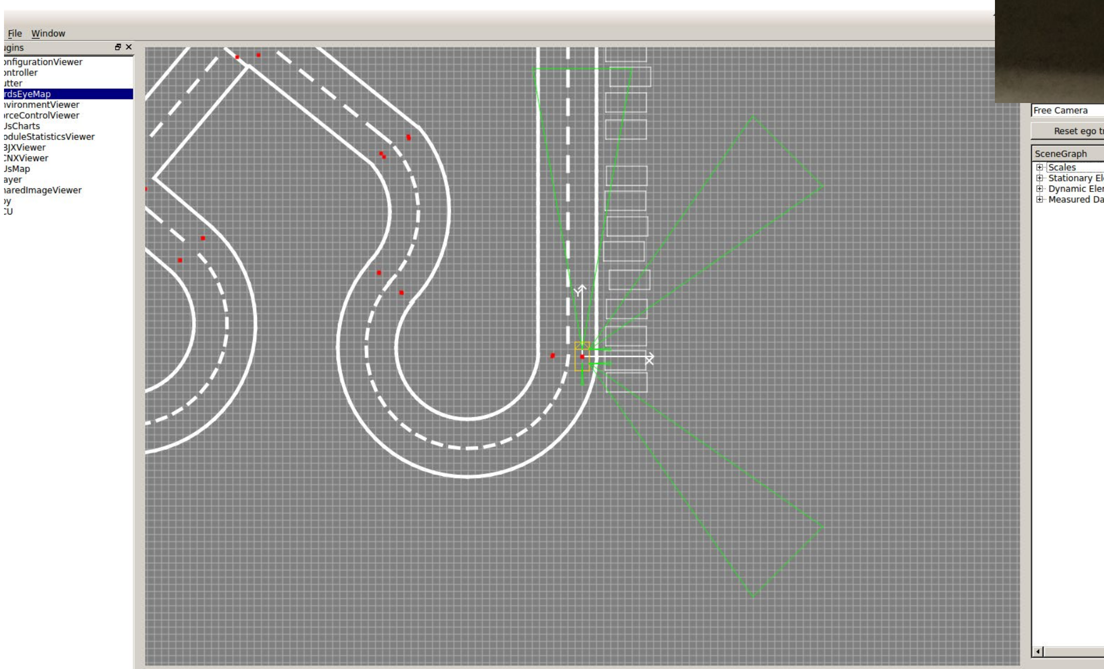
```
brick smiley
o o o o o o o o,
o _ _ o o _ _ o,
o _ _ o o _ _ o,
o o o o o o o o,
o _ o o o o _ o,
o _ _ _ _ _ _ o,
o o o o o o o o,
```

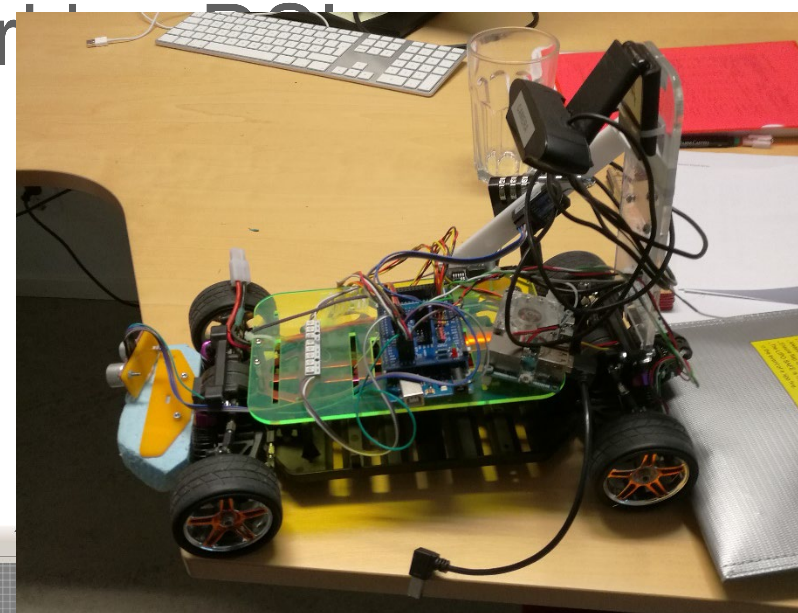




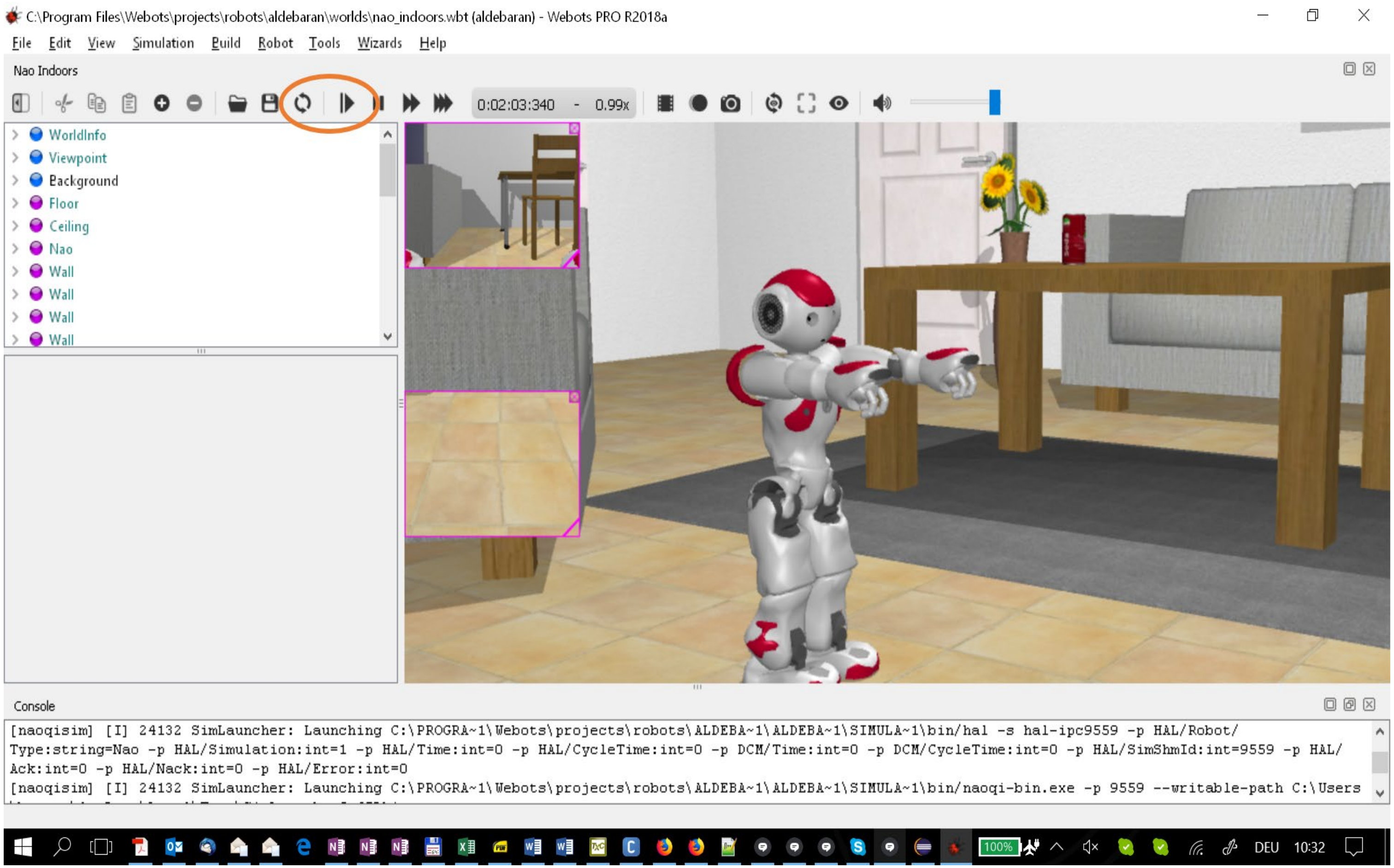
```
maxwidth 10
maxlength 10

abstract brick a
  o o o,
  o o o,

abstract brick b
  _ o o,
  _ o o,
  _ o o,

combo T a over b
```

Autonomous Driving, Par



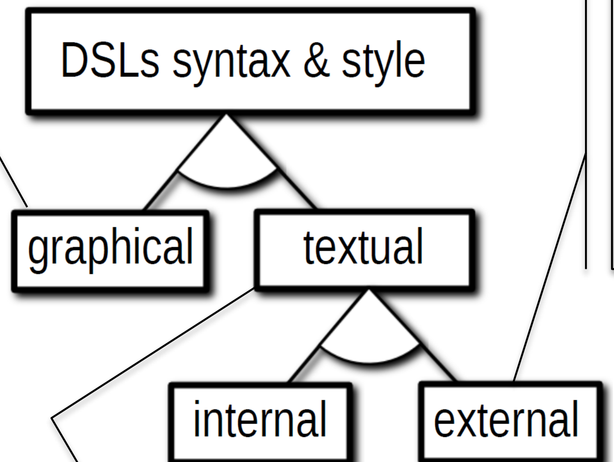


Controlling a NAO Robot



kinds of DSLs

comprehensible
good for **nonprogrammers**
layout conveys information
layout tedious to use
layout **tedious to implement**
syntax **scalability** issues
examples: BPMN, class diagrams, feature models, Yahoo pipes, MIT scratch



implemented as a **separate language**; parsed, translated/ interpreted
good control over syntax, semantics, tooling
costlier to implement
example: Google protocol buffers

clear **order of reading**
efficient **editing**
easy to incorporate **expressions**
popular among programmers
cheaper to implement, good **tools** exist
harder to read, esp. for non-programmers
complex **dependencies hide** in indirections
examples: Google protocol buffers, Rails active record

a **library** within a **host** language
host must be **flexible** syntactically and support meta-programming features: scala, ruby, python, smalltalk, C++, haskell
cheap to implement, **reuse** from host
limited control of syntax, semantics, tooling
example: Rails active record

BUILDING DSLS

language workbenches

language workbench:

tool for creating and using (domain-specific) languages

early workbenches (textual)

- SEM, MetaPlex, Metaview, QuickSpec, MetaEdit

workbenches for graphical languages

- MetaEdit+, DOME, GME

workbenches for textual languages

- Centaur, Synthesizer, ASF+SDF Meta-Environment, Gem-Mex/Montages, LRC, Lisa, JastAdd, Rascal, Spoofax, Xtext

workbenches for projectional languages

- Jetbrains MPS, Intentional Domain Workbench, …

Erdweg, Sebastian, et al. "The state of the art in language workbenches." *International Conference on Software Language Engineering*. 2013.

in-class demo

We will implement a language (incl. abstract syntax and a textual and graphical concrete syntax) for expressing simple graphs.

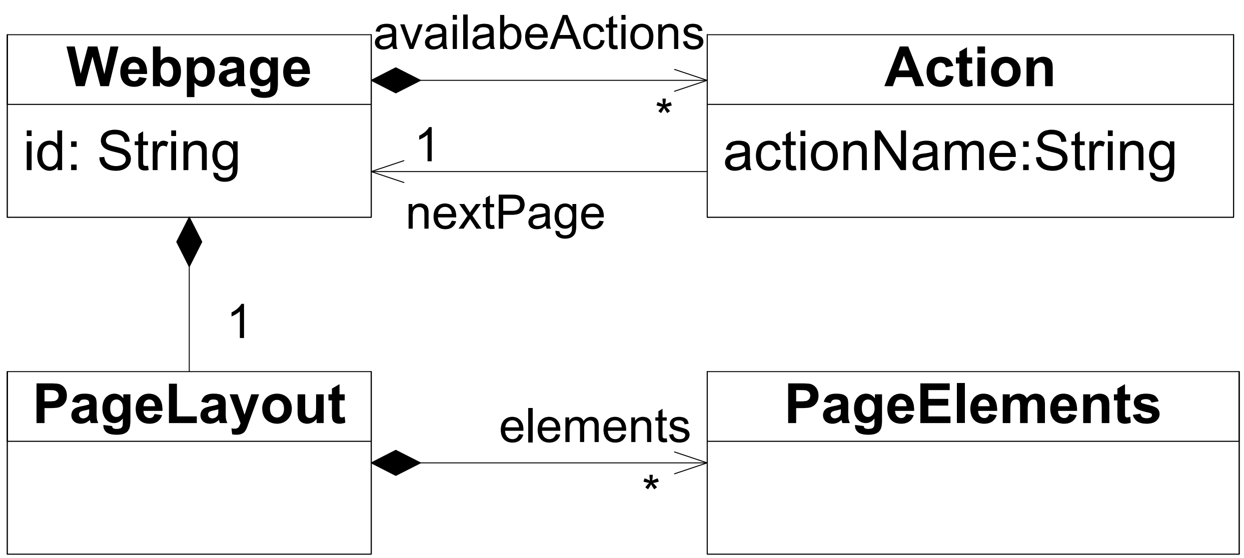
Using the following technological space:

meta-model (abstract syntax)

how to build a modelling languages?

we *model* the modelling language

result is a meta-model:



meta-model: definition

A meta-model is a model that precisely defines the parts and rules needed to create valid models.

Parts: domain concepts (model elements)

Rules: Well-formedness rules, determine validity of a model.

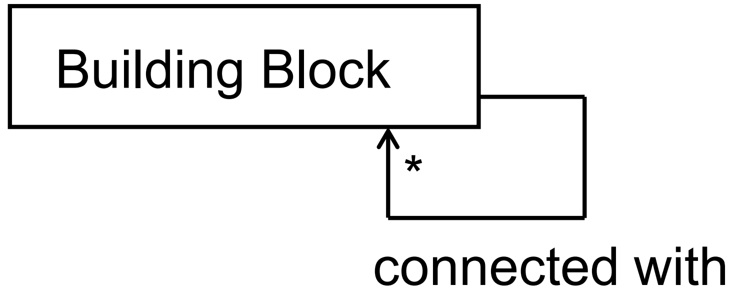
Defines a languages' abstract syntax: elements and their relations independent of the representation

Mapped to:

- Concrete syntax: representation of models (instances), e.g., within an editor
- Static semantics: semantics evaluatable without executing/interpreting the model (using constraints)
- Dynamic semantics: what the model means or expresses

meta-model: lego

Meta-Model
Building Rules



Model
Lego

Real world
House

examples from Steffen Becker, Model-Driven Software Development, Univ. of Paderborn

Images: pixelquelle.de

meta-model: languages

Meta-Model
Building Rules

Model
A natural description

"A nice brown and white coloured house in the middle of the black forest"

Real world
House

examples from Steffen Becker, Model-Driven Software Development, Univ. of Paderborn

Images: pixelquelle.de

a well-known DSL

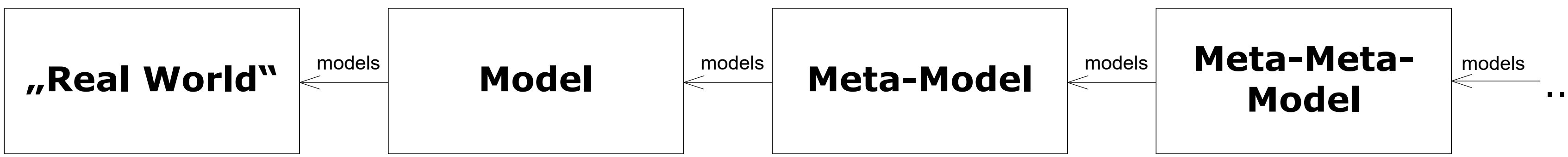
examples from Steffen Becker, Model-Driven Software Development, Univ. of Paderborn

meta-model levels

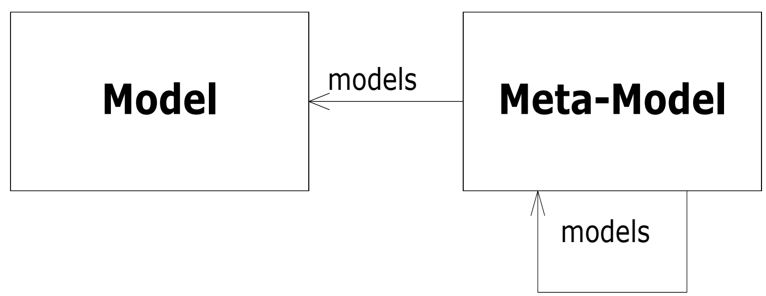
a meta-model is *also a model*

how is the meta-model then being modelled?

using a meta-meta-model! ☺



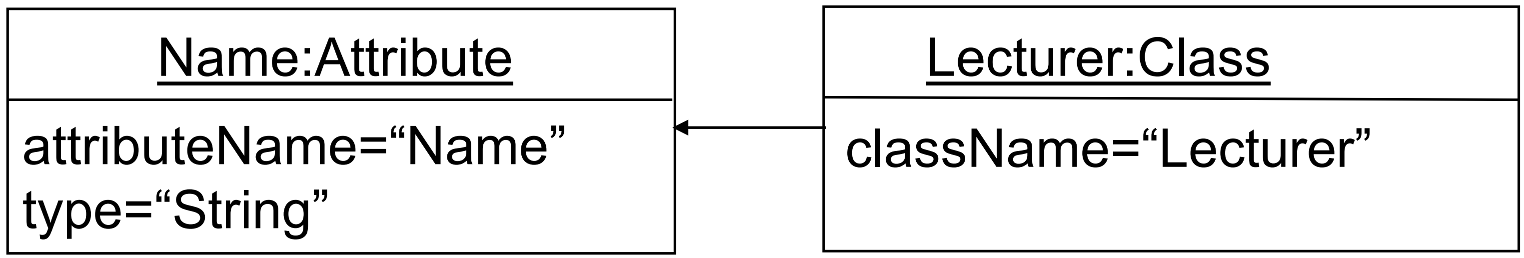
this goes (theoretically) infinite or ends if the model is *self-describing*



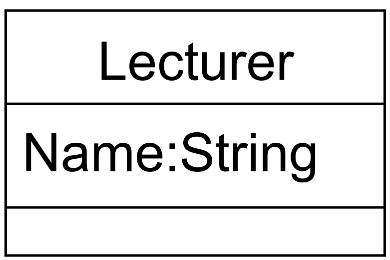
typically called *bootstrapping* (your favorite Java compiler is likely implemented in Java itself)

abstract vs. concrete syntax

abstract



Concrete




```
Class Lecturer {
   String Name;
}
```

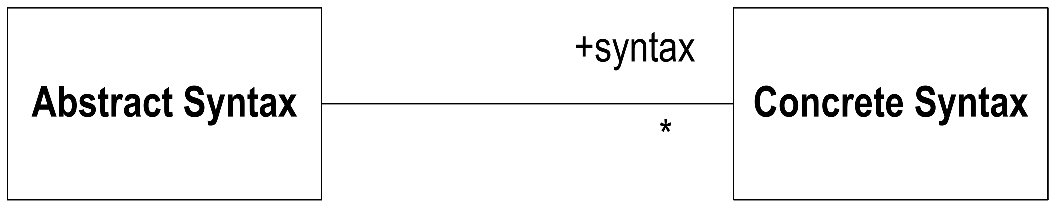
```
modelclass Lecturer
(
   attributes
   (
      Name : Type String
   )
)
```

```
<class
   className="Lecturer">
   <attribute
      attributeName="Name"
      type="String"/>
</class>
```

Compare to the AST concept in compiler construction!

examples from Steffen Becker, Model-Driven Software Development, Univ. of Paderborn

abstract vs. concrete syntax

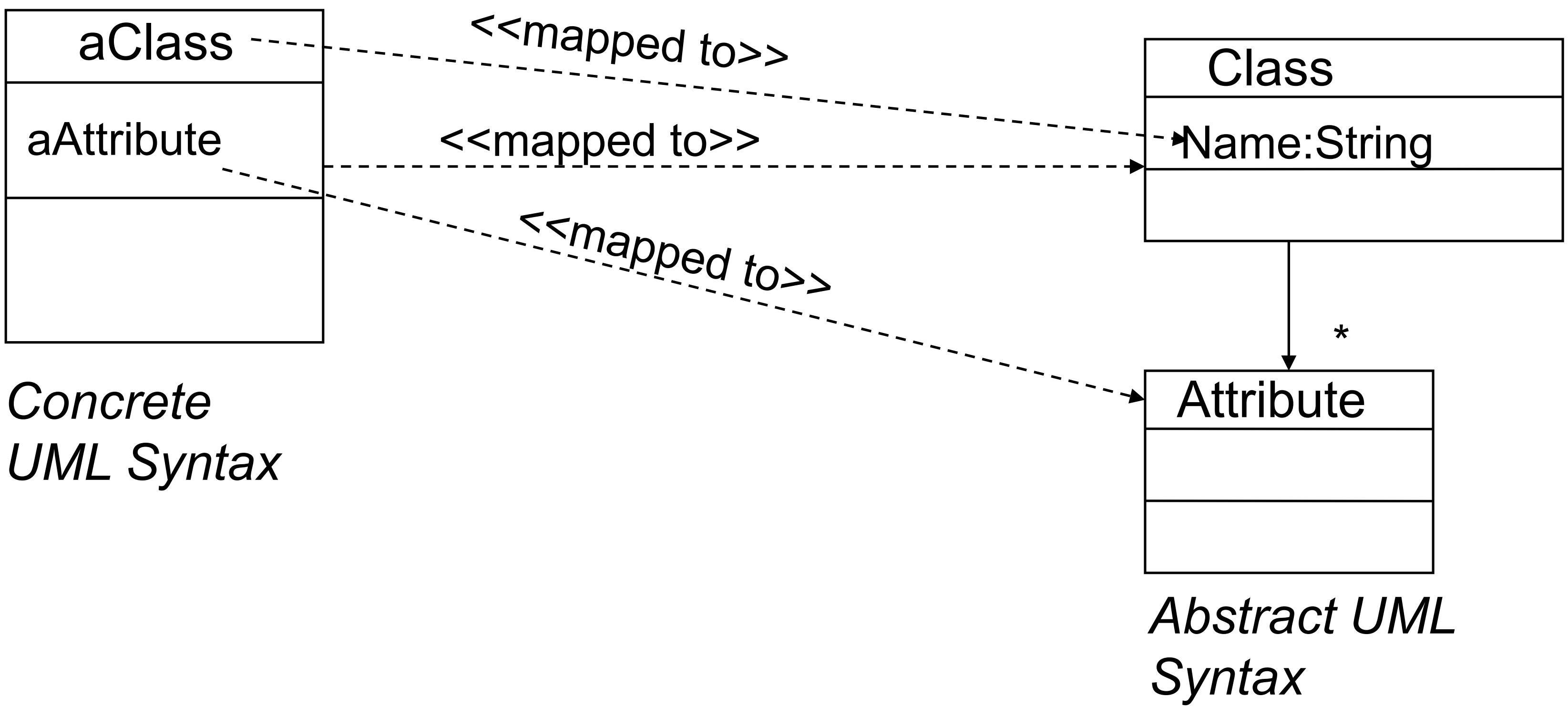


abstract syntax uses the meta-model concepts to represent the models

concrete syntaxes can choose any kind of representation plus a mapping

abstract vs. concrete syntax

a mapping maps elements of the concrete syntax to the meta-model elements

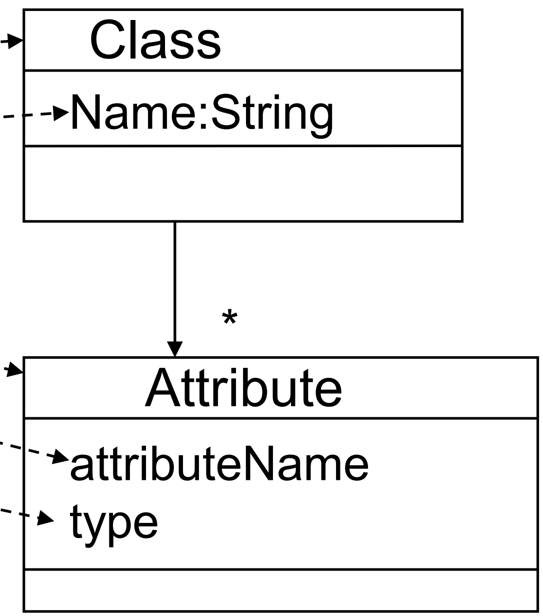


examples from Steffen Becker, Model-Driven Software Development, Univ. of Paderborn

abstract vs. concrete syntax

A mapping maps elements of the concrete syntax to the meta-model elements

```
<class
   className="Lecturer">
   <attribute
      attributeName="Name"
      type="String"/>
</class>
```



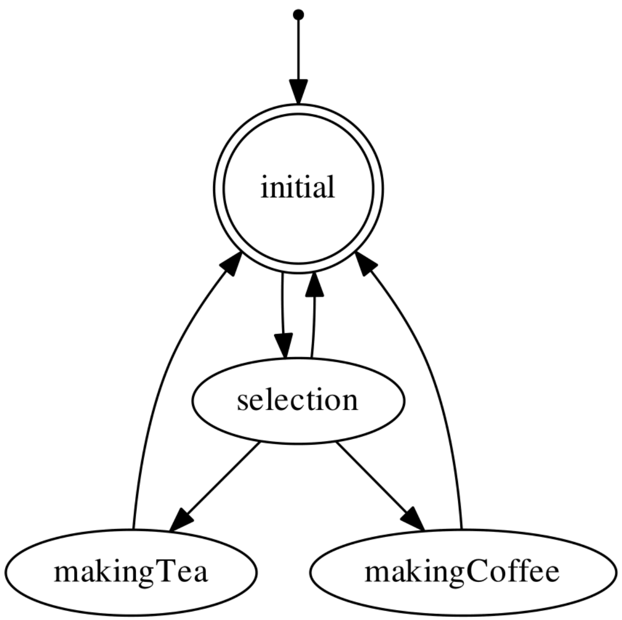
there is a standard defining the mapping from (MOF) models to XML: XMI (XML Metadata Interchange)

examples from Steffen Becker, Model-Driven Software Development, Univ. of Paderborn

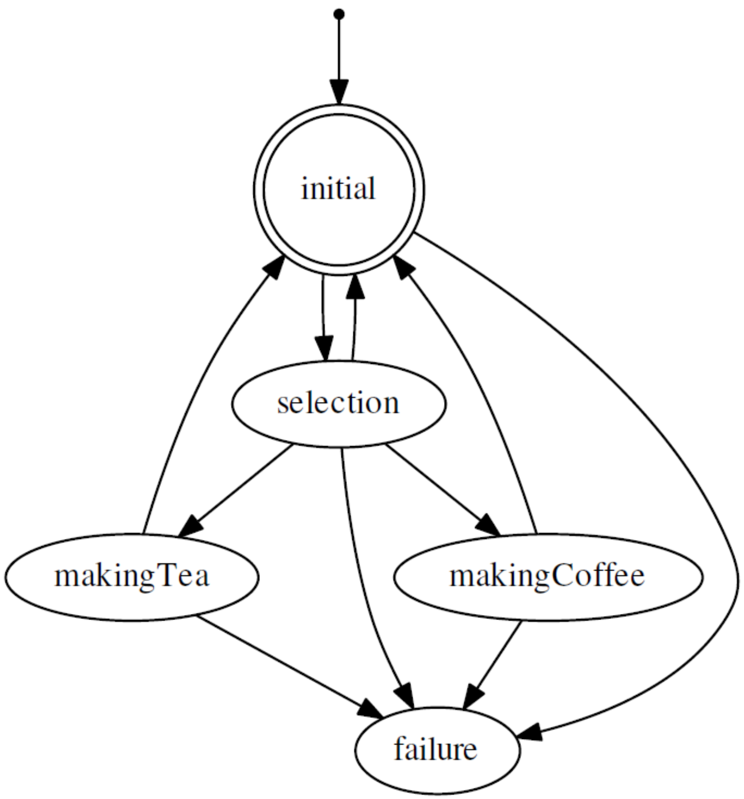
example walkthrough again

EXTERNAL DSL (TEXTUAL)

coffee machine



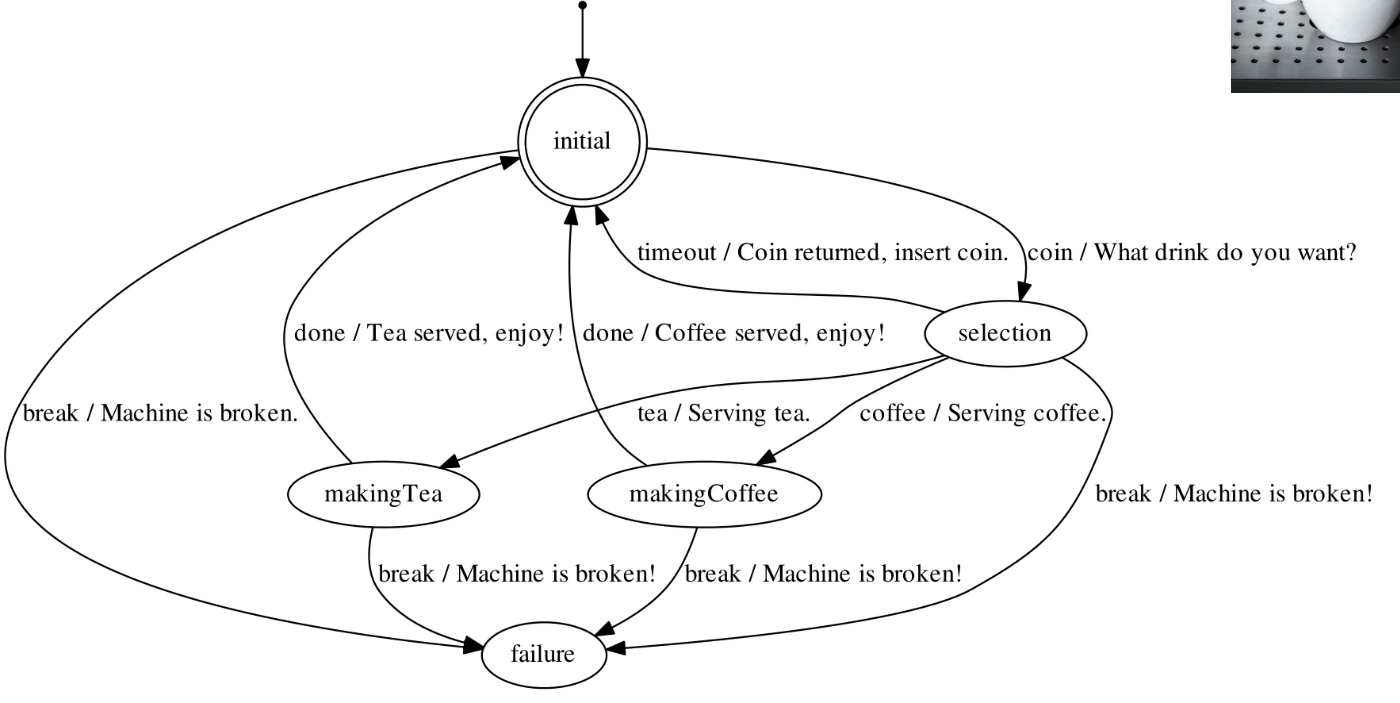
coffee machine



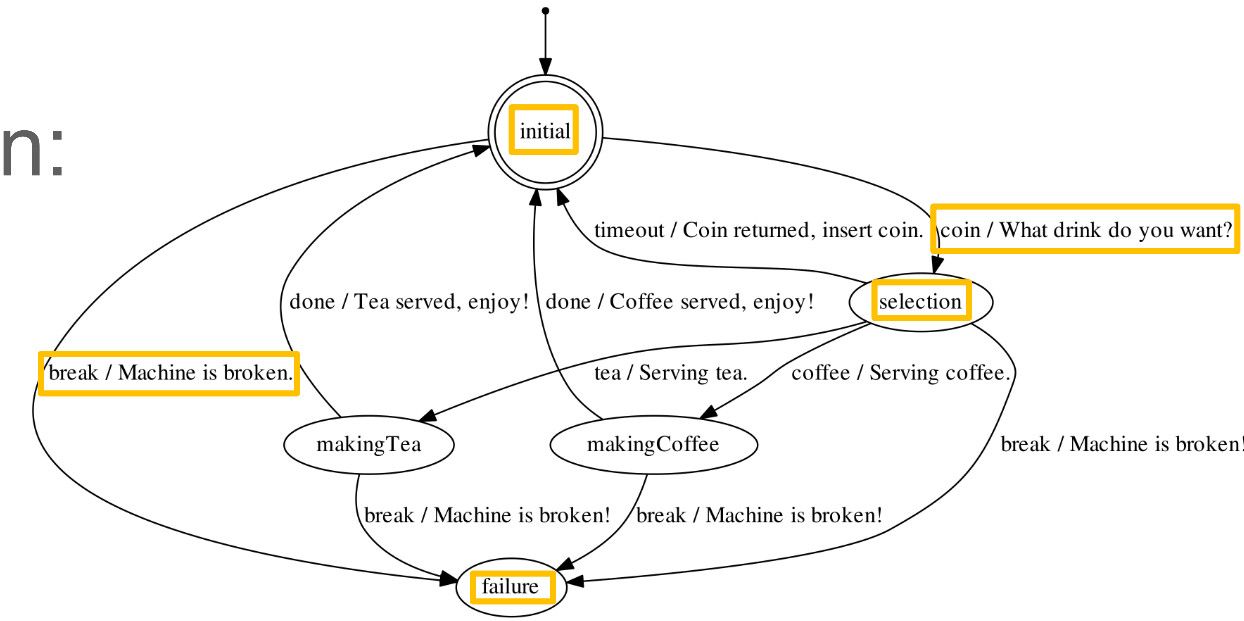
input / output

Σ = {coin, timeout, tea, coffee, break}

Γ = {*Tea served, enjoy!*, *Machine is broken.*, …}



implementation: switch pattern




```
int current = INITIAL;
while (true) {
    String input = scanner.nextLine();
    switch (current) {
        case INITIAL:
            switch (input) {
                case "coin":
                    System.out.println ("What drink do you want?");
                    current = SELECTION; break;
                case "break":
                    System.out.println ("Machine is broken");
                    current = FAILURE; break;
            } break;
        case SELECTION:
            switch (input) {
…
```

other implementation options

switch pattern

- simple, fast
- cluttered

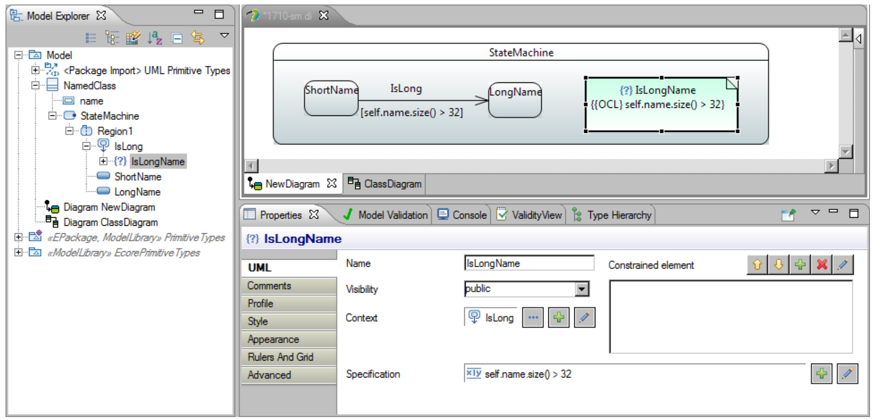
state pattern [Gamma et al. 95, Johnson and Zweig 91]

- comprehensible (esp. for hierarchical models)
- OO overhead (polymorphism)

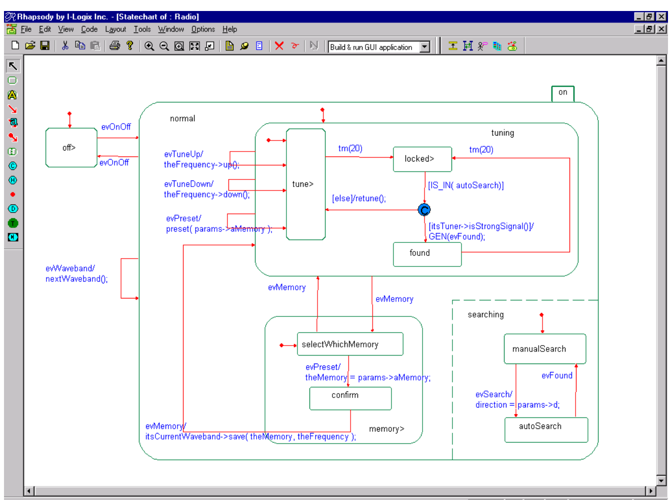
interprete a data structure at runtime [Pinter and Majzik 03, Zündorf 02]

- few code, very memory-efficient
- interpreter overhead

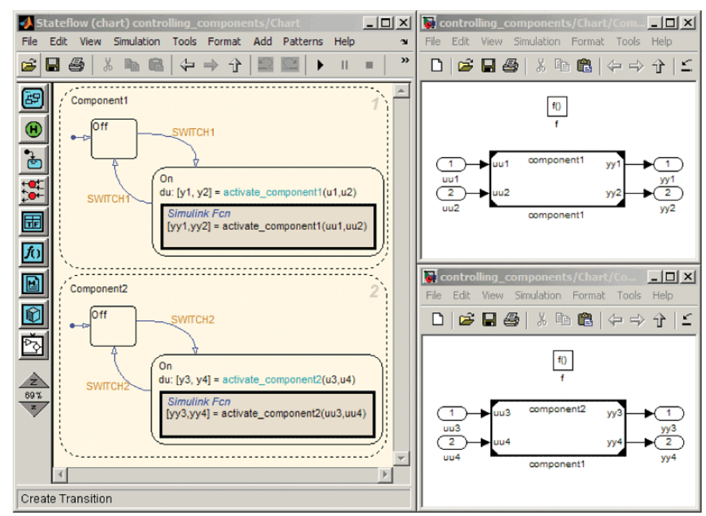
use existing tools?



Papyrus



IBM Rational Rhapsody



Mathworks Simulink

let’s build our own DSL

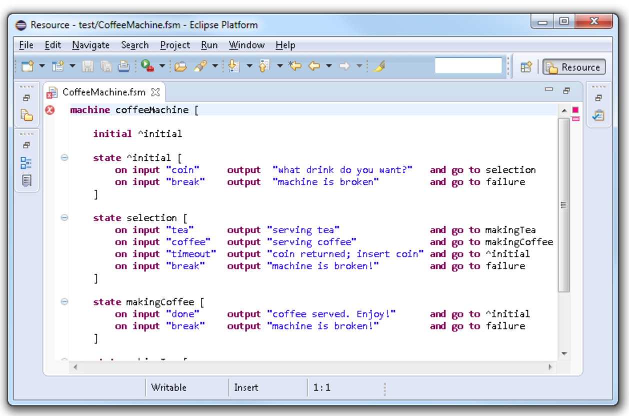
domain-specific language (DSL)

programming language (GPL)

graphical DSL

code generation

textual DSL



code generation

```
int current = INITIAL;
while (true) {
    String input = scanner.nextLine();
    switch (current) {
        case INITIAL:
            switch (input) {
                case "coin":
                        System.out.println ("What drink do you want?");
                        current = SELECTION; break;
                case "break":
                        System.out.println ("Machine is broken");
                        current = FAILURE; break;
            } break;
        case SELECTION:
            switch (input) {
…
```

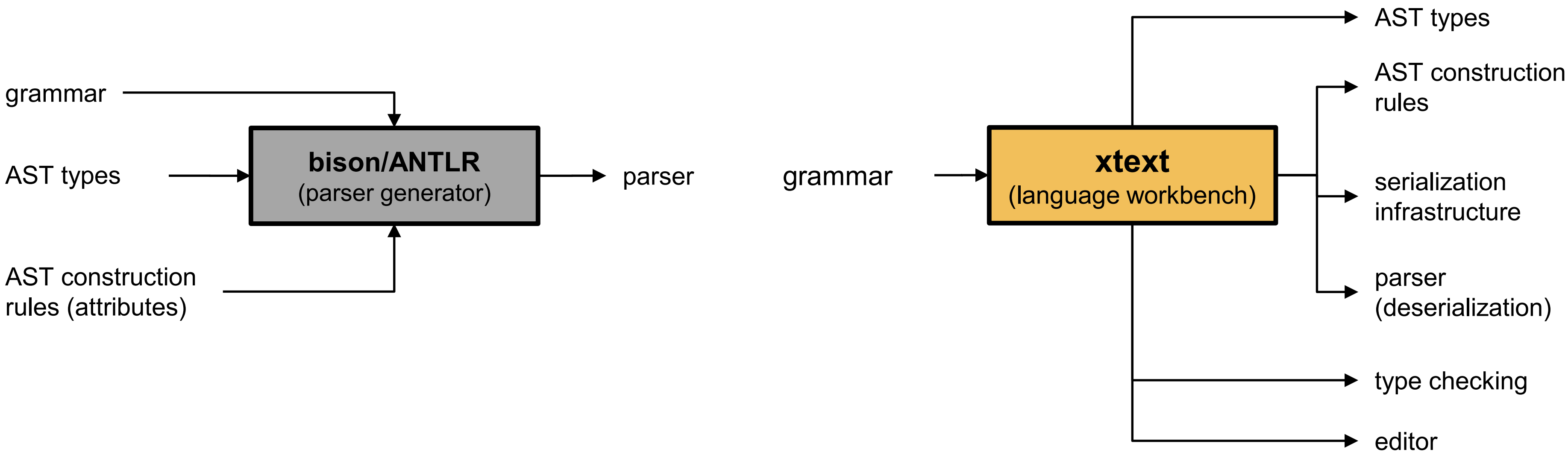
DSL … Domain-Specific Language
GPL … General-Purpose Language

automata model in concrete textual syntax

editor with:
auto-completion,
syntax- and
reference-checks

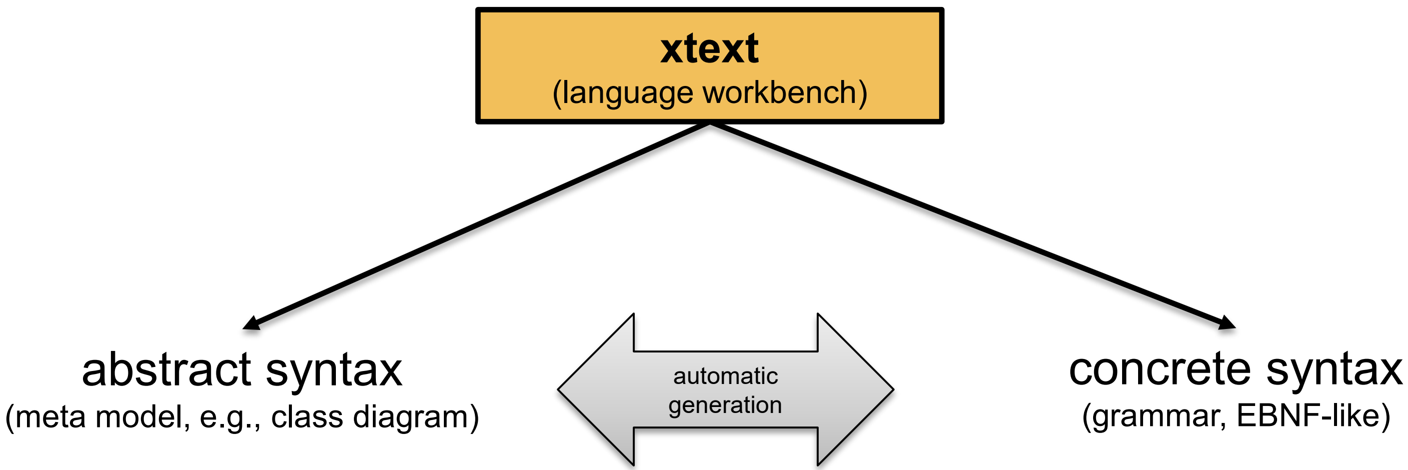
```
machine coffeeMachine [

    initial ^initial

    state ^initial [
        on input "coin“          output  "what drink do you want?"   and go to selection
        on input "break“         output  "machine is broken"         and go to failure
    ]

    state selection [
        on input "tea”           output "serving tea"                and go to makingTea
        on input "coffee"        output "serving coffee"             and go to makingCoffee
        on input "timeout"       output "coin returned; insert coin" and go to ^initial
        on input "break"         output "machine is broken!"         and go to failure
    ]

    state makingCoffee [
        on input "done"          output "coffee served. Enjoy!"      and go to ^initial
        on input "break"         output "machine is broken!"         and go to failure
    ]

    state makingTea [
        on input "done"          output "tea served. Enjoy!"         and go to ^initial
        on input "break"         output "machine is broken!"         and go to failure
    ]

    state failure
]
```

two external DSL tactics



- ANTLR supports many target platforms. There exists a tool like that for any serious language technical space
- ANTLR can do a bit more than the figure would suggest (AST classes and construction)
- there exist workbenches for visual languages (GMF, Graphiti, Microsoft DSL Tools, ...)
- implementation of graphical DSLs only differs from textual in the syntax/editor aspect
- the third tactic: do everything manually is left only to the insane ...

syntax

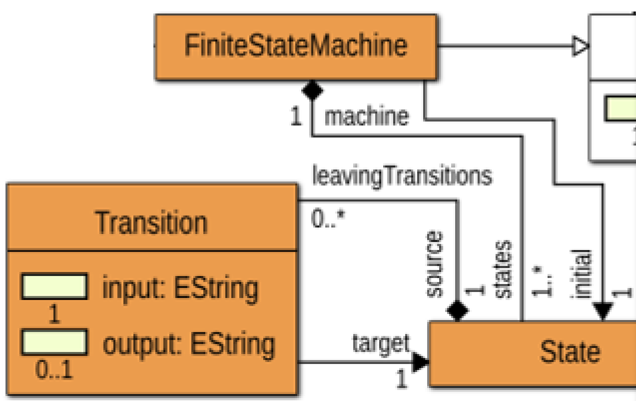


xtext
(language workbench)
abstract syntax
(meta model, e.g., class diagram)
automatic
generation
concrete syntax
(grammar, EBNF-like)

syntax

```
machine coffeeMachine [

  initial ^initial

  state ^initial [
    on input "coin“      output  "what drink do you want?“   and go to selection
    on input "break“     output  "machine is broken“         and go to failure
  ]

…

  state failure
]
```

abstract syntax　　　　concrete syntax




```
FiniteStateMachine:
  'machine' name=ID
  ('[' 'initial' initial=[State]
      (states+=State)*
   ']')?;

State:
  'state' name=ID ('[' leavingTransitions+=Transition* ']')?;

Transition:
  'on' 'input' input=STRING ('output' output=STRING)?
  'and' 'go' 'to' target=[State];
```

model-driven development

textual DSL

domain-specific language (DSL)
in concrete and abstract syntax

code generation (M2T)

programming language (GPL)

```
int current = INITIAL;
while (true) {
    String input = scanner.nextLine();
    switch (current) {
        case INITIAL:
            switch (input) {
                case "coin":
                        System.out.println ("What drink do you want?");
                        current = SELECTION; break;
                case "break":
                        System.out.println ("Machine is broken");
                        current = FAILURE; break;
            } break;
        case SELECTION:
            switch (input) {
...
```

DSL … Domain-Specific Language
GPL … General-Purpose Language

generator

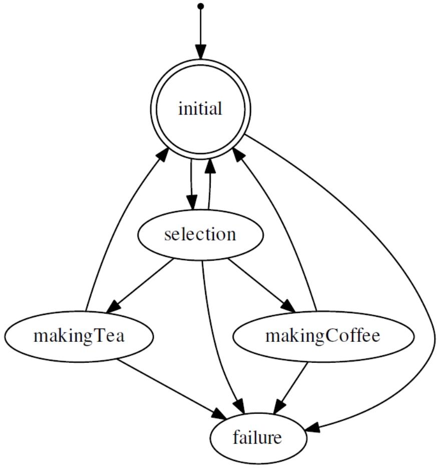
Xtend

alternatives

- XPAND, JSP, Velocity, etc.

```
def static compileToJava(FiniteStateMachine it) {
  var int i = -1
  '''
...

          Scanner scanner = new Scanner(System.in);
          current = «initial.name.toUpperCase»;

          while (true) {
            print ("[" + stateNames[current] + "] ");
            print ("What is the next event? available: " + availableInputs[current]);
            String input = scanner.nextLine();

            switch (current) {

              «FOR state : states»
                case «state.name.toUpperCase»:
                  switch (input) {
                    «FOR t : state.leavingTransitions»
                      case "«t.input»":
                        println ("machine says: «t.output»");
                        current = «t.target.name.toUpperCase»;
                        break;
                    «ENDFOR»
                  }
                  break;
              «ENDFOR»
            }}}}'''
}
```

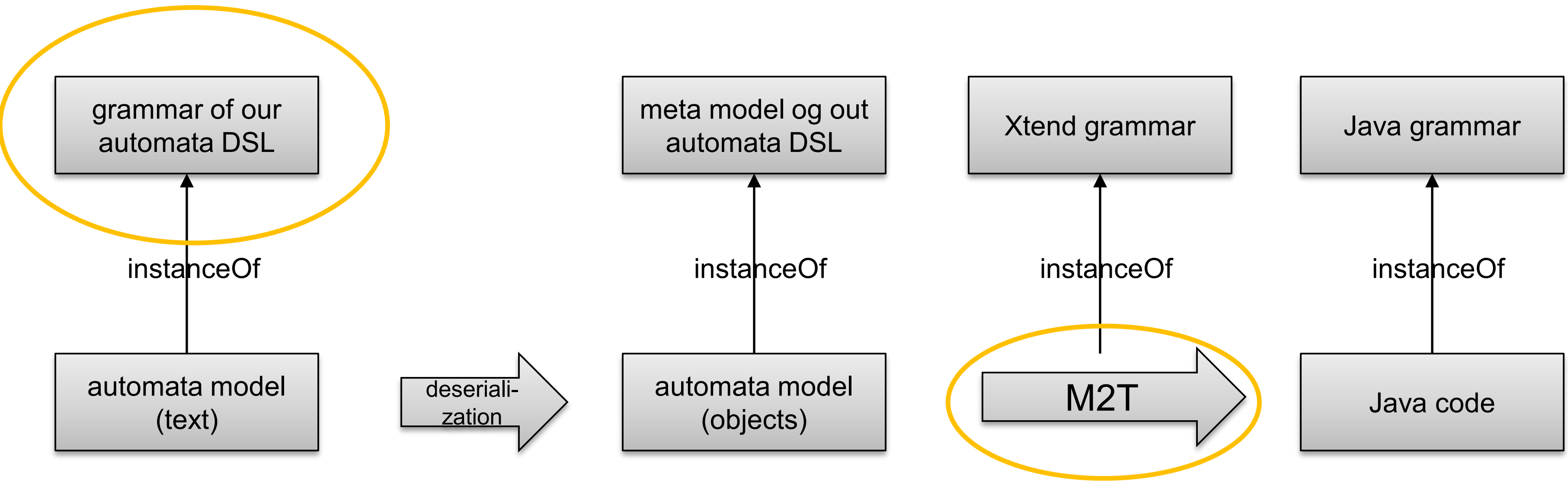
target: Java

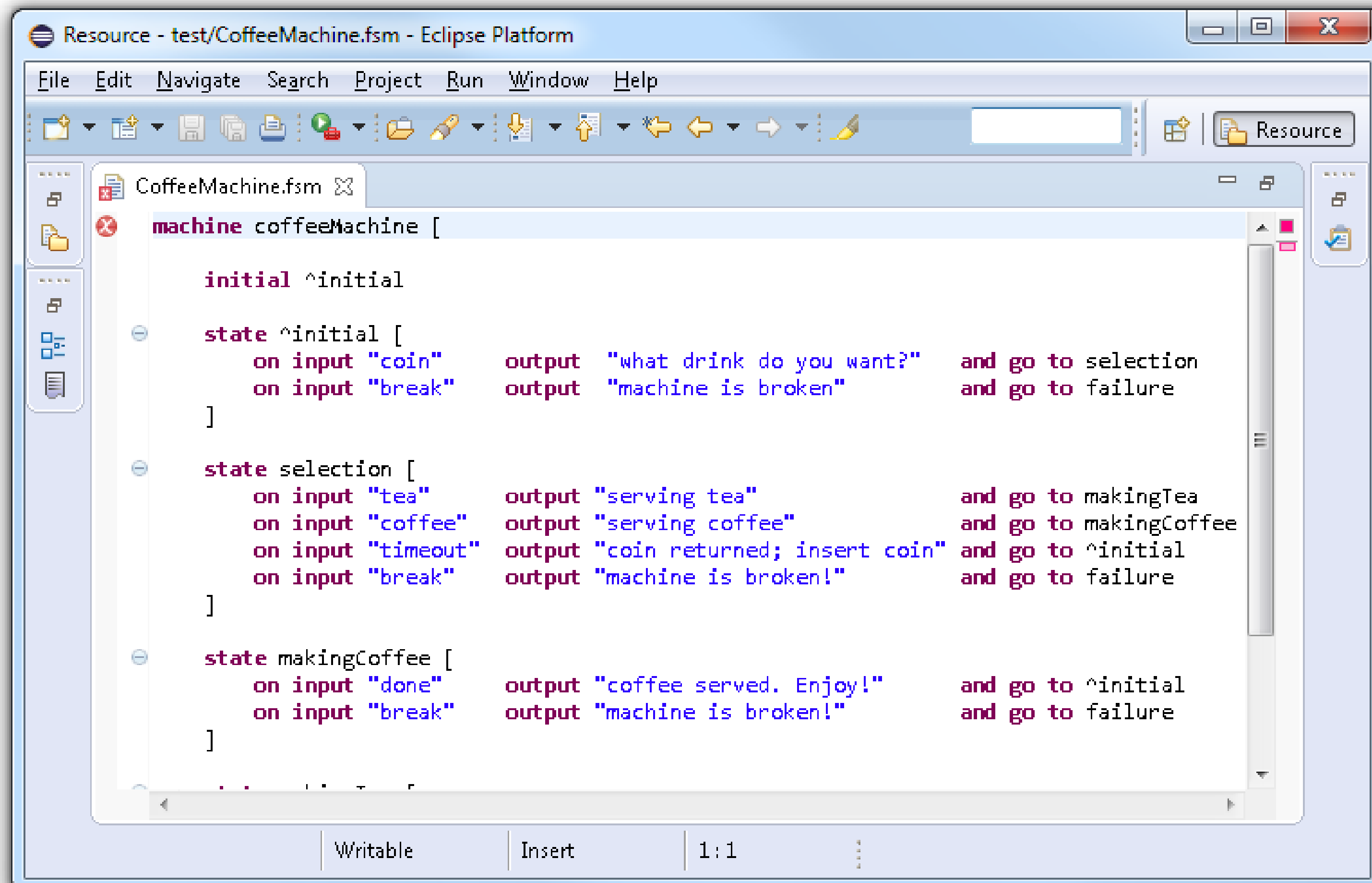



```
def static compileToDot(FiniteStateMachine it) {
'''
digraph "«it.name»" {
  _init -> «it.initial.name»;
  «FOR state : states»
    «FOR t : state.leavingTransitions»
      "«state.name»" -> "«t.target.name»" [label="«t.input» / «t.output» "];
    «ENDFOR»
  «ENDFOR»
  «it.initial.name» [shape=doublecircle];
  _init [shape=point];
}
'''}
```

target: Graphviz

overview


grammar of our automata DSL
meta model og out automata DSL
Xtend grammar
Java grammar
instanceOf
instanceOf
instanceOf
instanceOf
automata model (text)
deseriali-zation
automata model (objects)
M2T
Java code


Resource - test/CoffeeMachine.fsm - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Resource
CoffeeMachine.fsm
machine coffeeMachine [
initial ^initial
state ^initial [
on input "coin" output "what drink do you want?" and go to selection
on input "break" output "machine is broken" and go to failure
]
state selection [
on input "tea" output "serving tea" and go to makingTea
on input "coffee" output "serving coffee" and go to makingCoffee
on input "timeout" output "coin returned; insert coin" and go to ^initial
on input "break" output "machine is broken!" and go to failure
]
state makingCoffee [
on input "done" output "coffee served. Enjoy!" and go to ^initial
on input "break" output "machine is broken!" and go to failure
]
Writable
Insert
1 : 1

GRAPHICAL SYNTAX

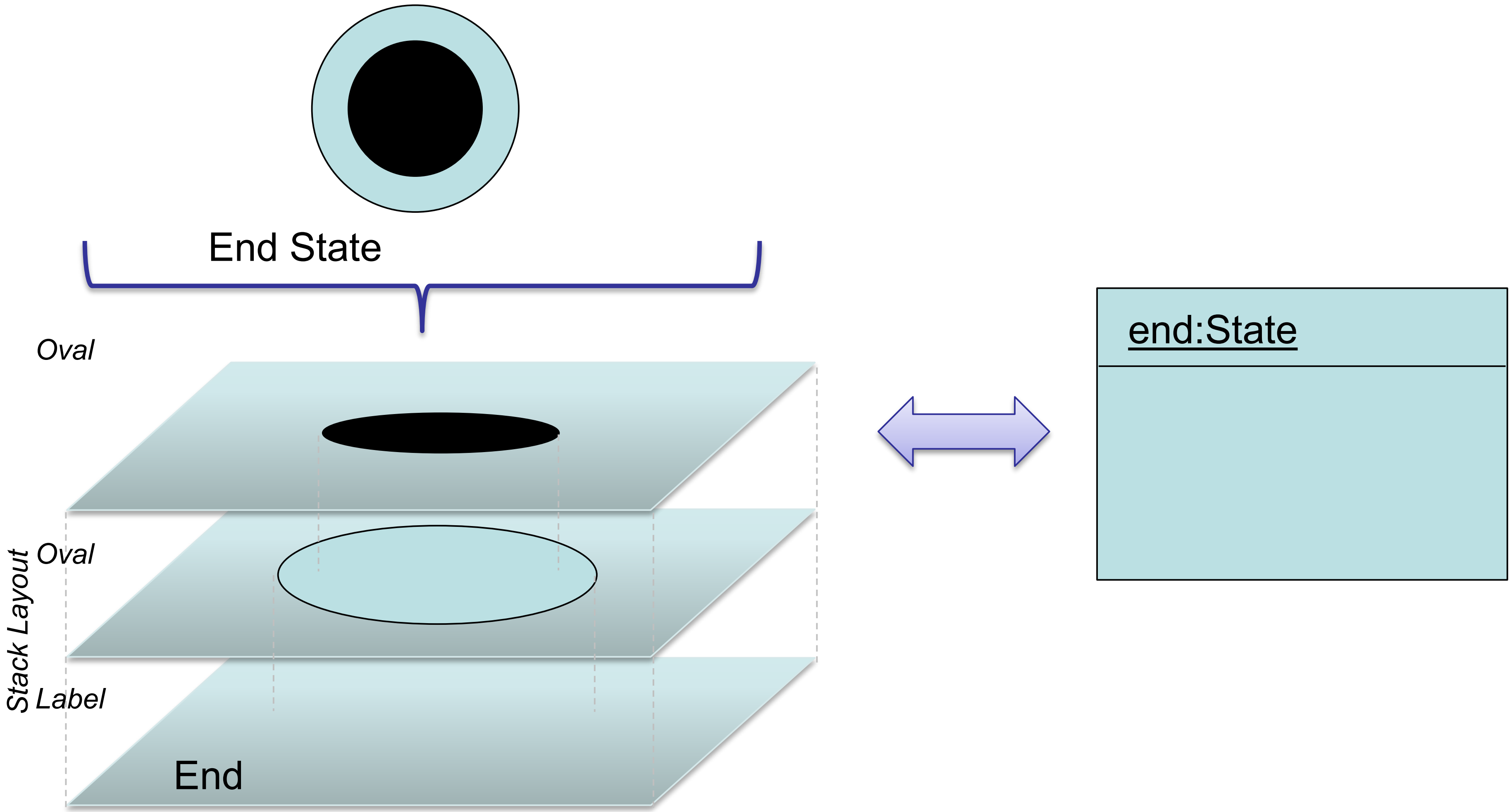
creation of a graphical syntax

Steps
- Define graphical elements
- Develop editing tool
- Implement mapping to meta-model

Typical questions
- Where do we store layout information?
- Do we support partial views?
- Auto layouts?

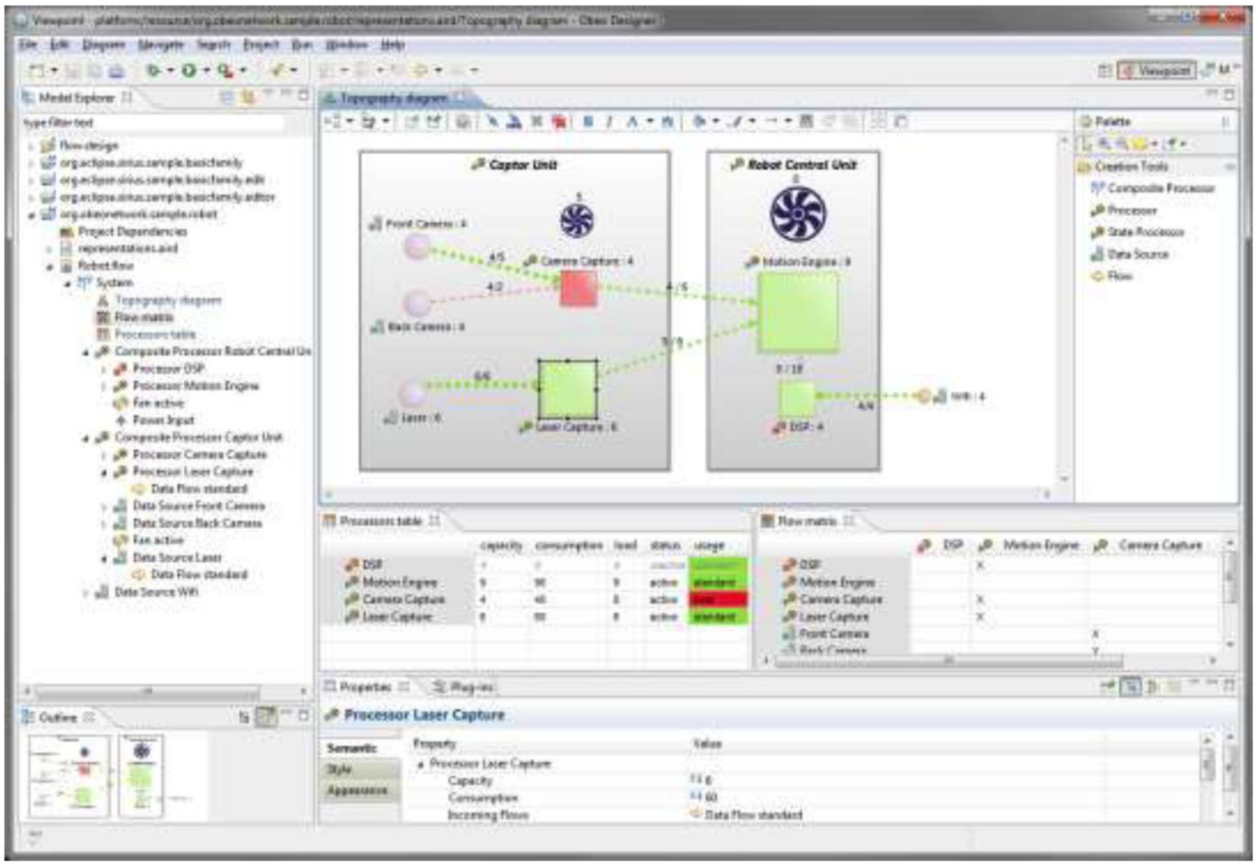
Graphical Editing Framework (GEF)
- Eclipse Plugin
- Framework for Diagram Editors
- used in IBM Rational Software Architect and Borland Together

Example: Mapping an end state…

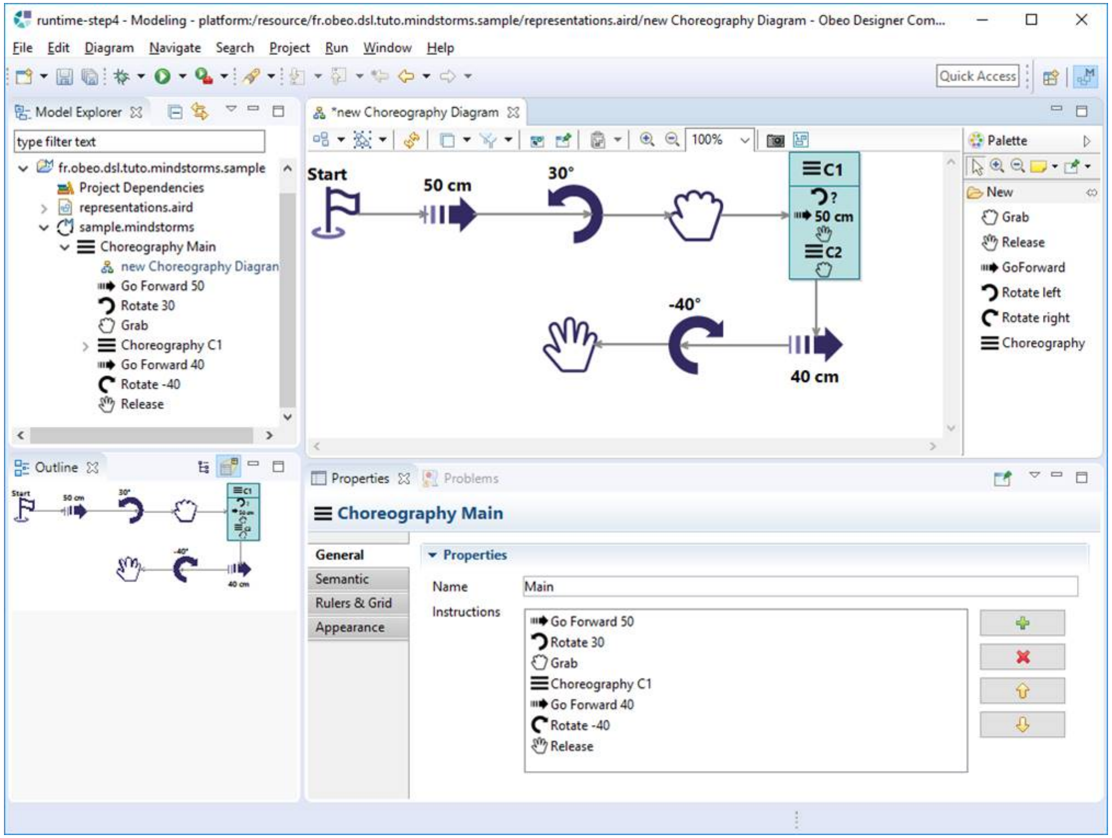


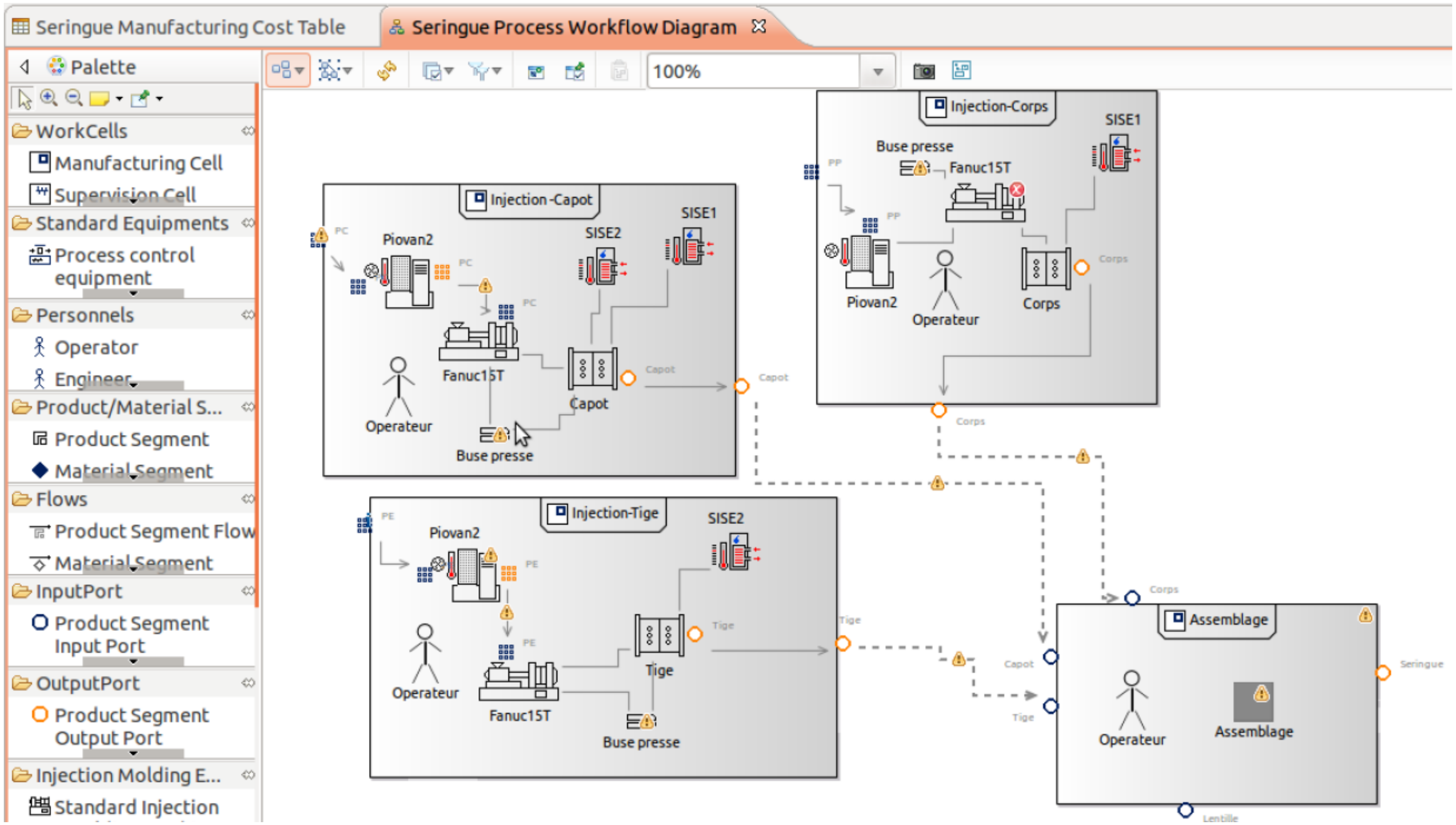
graphical syntax

Sirius



http://www.eclipse.org/sirius/gallery.html


runtime-step4 - Modeling - platform:/resource/fr.obeo.dsl.tuto.mindstorms.sample/representations.aird/new Choreography Diagram - Obeo Designer Com...
File Edit Diagram Navigate Search Project Run Window Help
Quick Access
Model Explorer
type filter text
fr.obeo.dsl.tuto.mindstorms.sample
Project Dependencies
representations.aird
sample.mindstorms
Choreography Main
new Choreography Diagram
Go Forward 50
Rotate 30
Grab
Choreography C1
Go Forward 40
Rotate -40
Release
*new Choreography Diagram
100%
Start
50 cm
30°
C1
?
50 cm
C2
-40°
40 cm
Palette
New
Grab
Release
GoForward
Rotate left
Rotate right
Choreography
Outline
Properties
Problems
Choreography Main
General
Semantic
Rulers & Grid
Appearance
Properties
Name
Main
Instructions
Go Forward 50
Rotate 30
Grab
Choreography C1
Go Forward 40
Rotate -40
Release


Seringue Manufacturing Cost Table
Seringue Process Workflow Diagram
Palette
WorkCells
Manufacturing Cell
Supervision Cell
Standard Equipments
Process control equipment
Personnels
Operator
Engineer
Product/Material S...
Product Segment
Material Segment
Flows
Product Segment Flow
Material Segment
InputPort
Product Segment Input Port
OutputPort
Product Segment Output Port
Injection Molding E...
Standard Injection
100%
Injection-Capot
Piovan2
SISE2
SISE1
Fanuc15T
Operateur
Capot
Buse presse
Injection-Corps
Corps
Injection-Tige
Tige
Assemblage
Seringue
Lentille

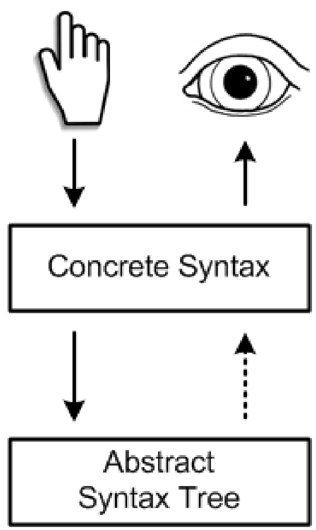
projectional language workbenchs for

LANGUAGE-ORIENTED PROGRAMMING

projectional editing

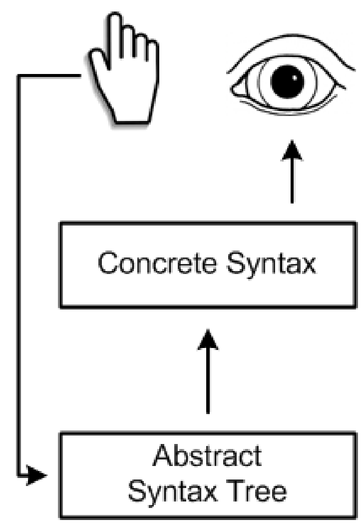
no grammars or parsers involved

parser-based editing



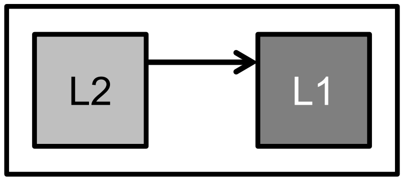
projectional editing

(a.k.a., syntax-directed editing, structured editing)



slide credits: Markus Völter

advantage: language composition



separate files

type system
transformation
constraints

in one file

type system
transformation
constraints
syntax
IDE

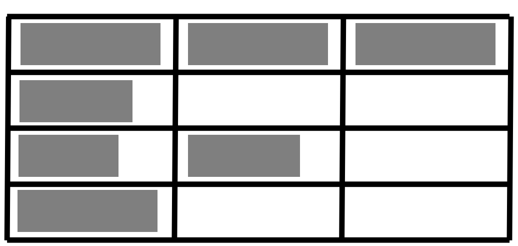
slide credits: Markus Völter

advantage: flexible notations

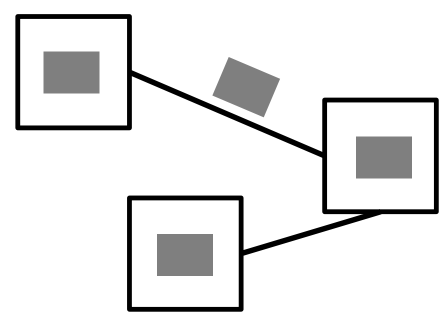
regular code/text

mathematical

tables



graphical



slide credits: Markus Völter

advantage: flexible notations

regular code/text

```
//[ A documentation comment with references ]
  [ to @arg(data) and @arg(dataLen)           ]
void aSummingFunction(int8[ ] data, int8 dataLen) {
  int16 sum;
  for (int8 i = 0; i < dataLen; i++) {
    sum += data[i];
  } for
} aSummingFunction (function)
```

mathematical

```
double midnight2(int32 a, int32 b, int32 c) {
```

$$\texttt{return}\ \frac{-b + \sqrt{b^2 - \sum_{i=1}^{4} a * c}}{2 * a};$$

```
} midnight2 (function)
```

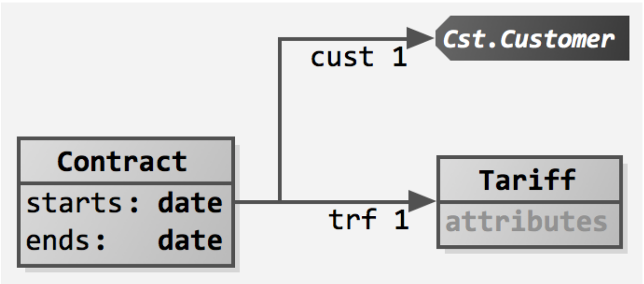
tables

```
int16 decide(int8 spd, int8 alt) {
  return
```

	spd > 0	spd > 100
alt < 0	1	1
alt == 0	10	20
alt > 0	30	40
alt > 100	50	60

```
  otherwise 0;
} decide (function)
```

graphical

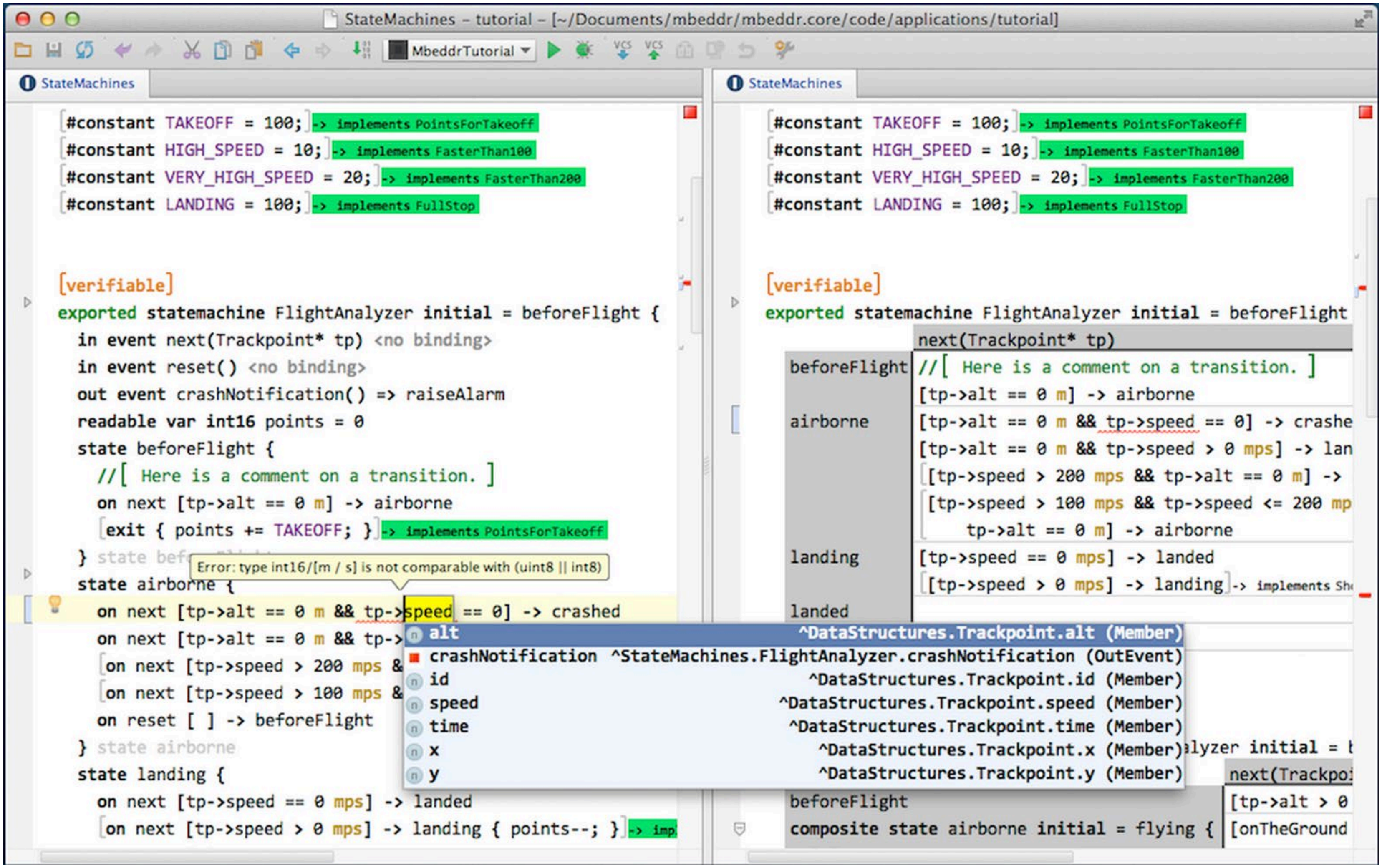


slide credits: Markus Völter

Jetbrains Meta Programming System (MPS)

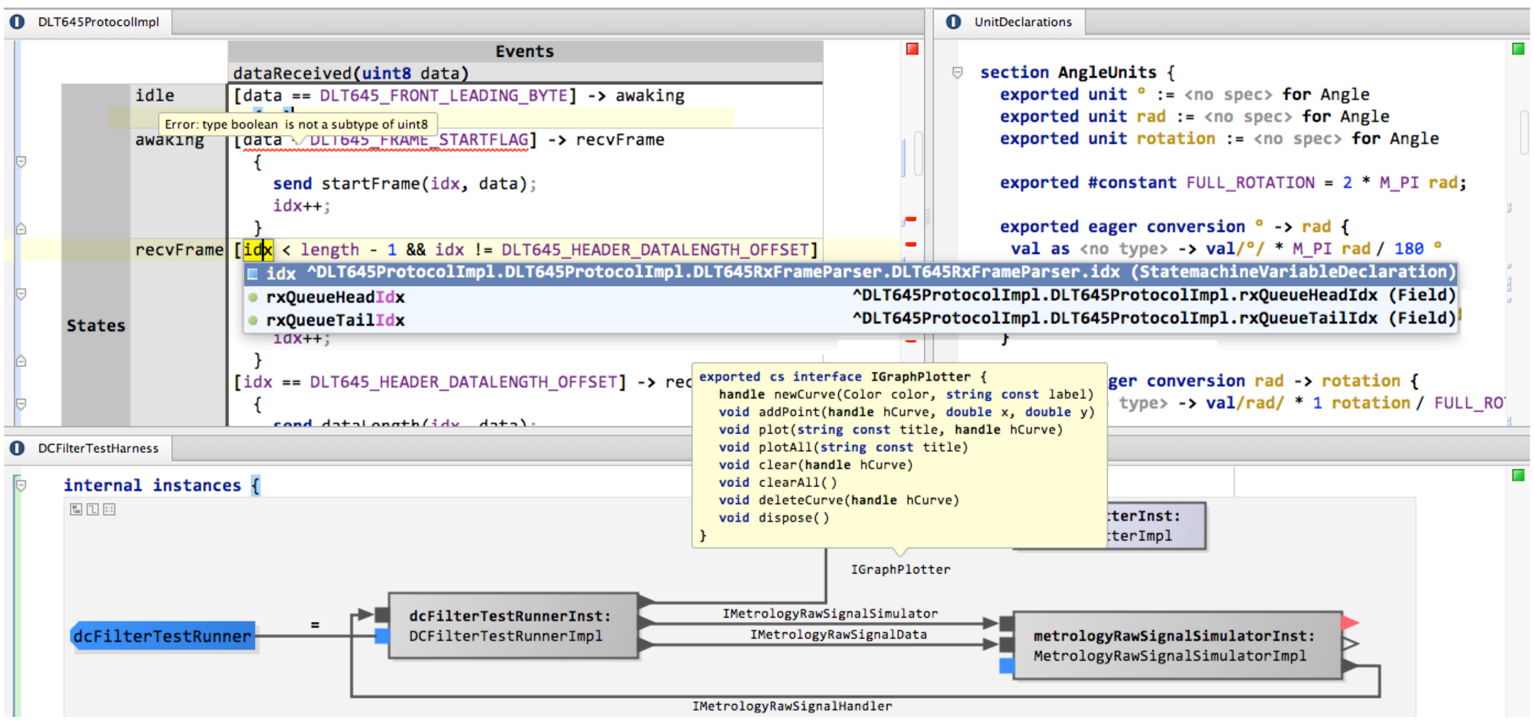
MPS is a language workbench
(a tool for defining, composing, and using languages)

MPS language workbench



slide credits: Markus Völter

MPS language ecosystems

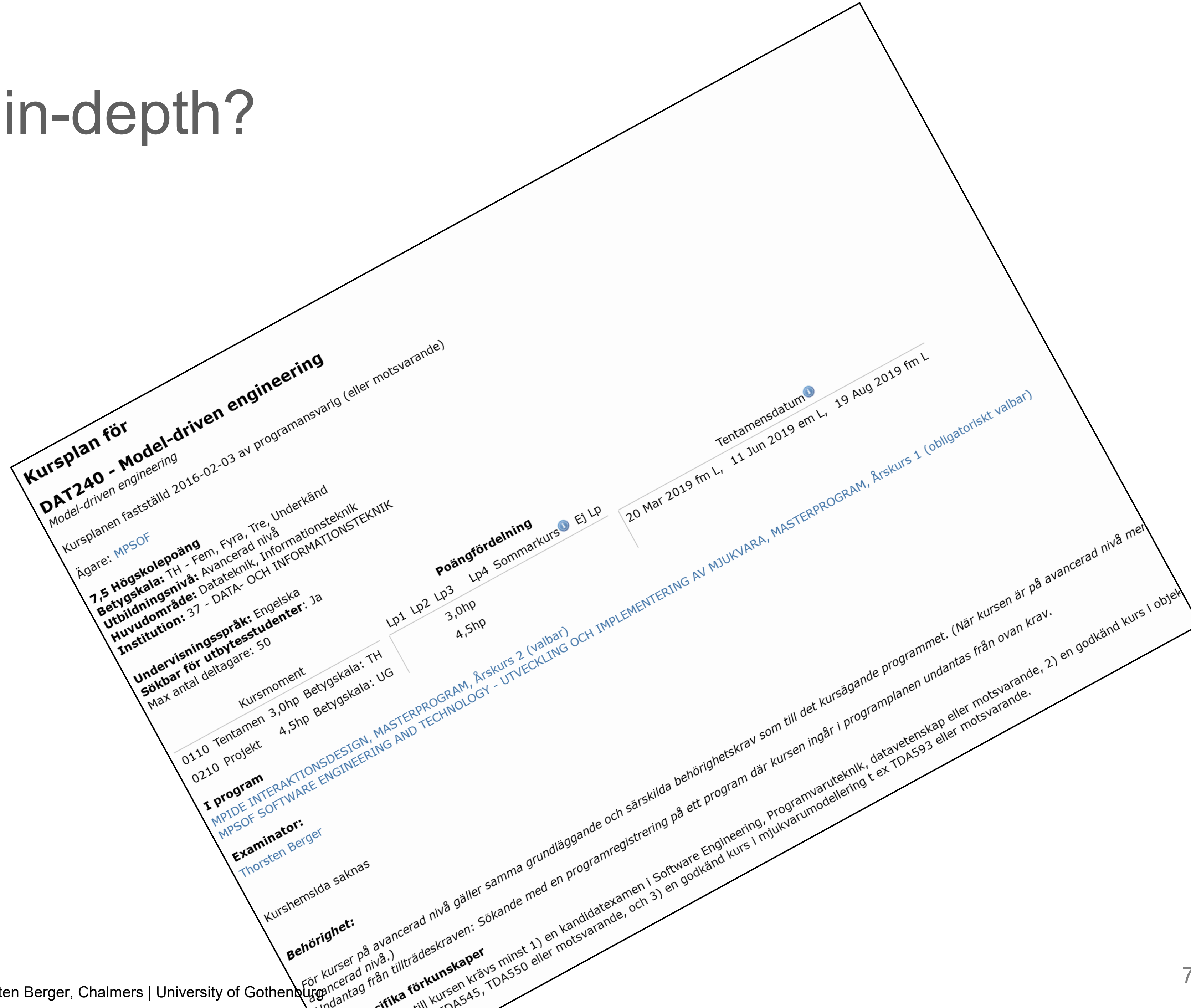


mbeddr

5+ base languages
50+ extensions to C
10+ extensions to requirements lang.

slide credits: Markus Völter

in-depth?



Kursplan för

DAT240 - Model-driven engineering

Model-driven engineering

Kursplanen fastställd 2016-02-03 av programansvarig (eller motsvarande)

Ägare: MPSOF

7,5 Högskolepoäng
Betygskala: TH - Fem, Fyra, Tre, Underkänd
Utbildningsnivå: Avancerad nivå
Huvudområde: Datateknik, Informationsteknik
Institution: 37 - DATA- OCH INFORMATIONSTEKNIK

Undervisningsspråk: Engelska
Sökbar för utbytesstudenter: Ja
Max antal deltagare: 50

Kursmoment	Lp1	Lp2	Lp3	Lp4	Sommarkurs	Ej Lp	Tentamensdatum
0110 Tentamen 3,0hp Betygskala: TH			3,0hp				20 Mar 2019 fm L, 11 Jun 2019 em L, 19 Aug 2019 fm L
0210 Projekt 4,5hp Betygskala: UG			4,5hp				

I program
MPIDE INTERAKTIONSDESIGN, MASTERPROGRAM, Årskurs 2 (valbar)
MPSOF SOFTWARE ENGINEERING AND TECHNOLOGY - UTVECKLING OCH IMPLEMENTERING AV MJUKVARA, MASTERPROGRAM, Årskurs 1 (obligatoriskt valbar)

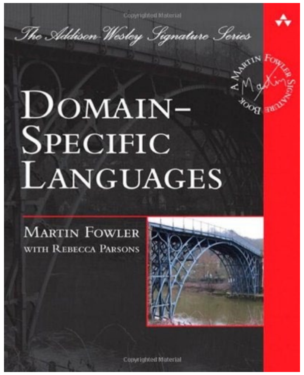
Examinator:
Thorsten Berger

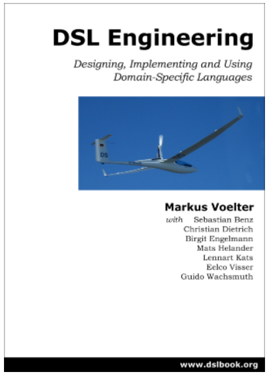
Kurshemsida saknas

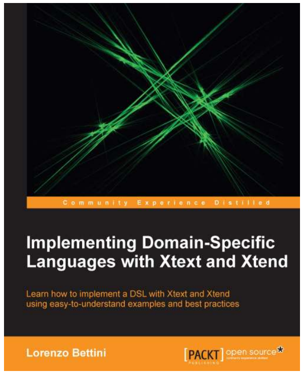
Behörighet:
För kurser på avancerad nivå gäller samma grundläggande och särskilda behörighetskrav som till det kursägande programmet. (När kursen är på avancerad nivå men ... avancerad nivå.)
Undantag från tillträdeskraven: Sökande med en programregistrering på ett program där kursen ingår i programplanen undantas från ovan krav.

...cifika förkunskaper
... till kursen krävs minst 1) en kandidatexamen i Software Engineering, Programvaruteknik, datavetenskap eller motsvarande, 2) en godkänd kurs i objek...
...TDA545, TDA550 eller motsvarande, och 3) en godkänd kurs i mjukvarumodellering t ex TDA593 eller motsvarande.

questions?







Domain-Specific Languages

Thorsten Berger, Assoc. Prof.
<thorsten.berger@chalmers.se>