


Teaching Program Generation to the Masses*

Ulrik Pagh Schultz, SDU UAS, University of Southern Denmark

(*) all the non-CS people

Overview

1. Context: the non-CS (the masses), who are they and what do they know?
2. Problem: what is hard for them to learn about program generation?
3. Solution: obvious motivation, modelling as abstraction, internal DSL stepping stone, …
4. Result: a few concrete examples & *input from the audience*

Audience

Dynamically added as we speak. Input from the audience added here…

Context: To be [CS] or not to be [CS]

Computer Scientist

- Knows programming, language theory, and GPL compilers.
- More likely to be making DSLs than full-blown compilers.
- *Statemachine: regular language, automata, computation, ….*

Roboticist

- Knows development, hardware, control theory, AI, embodiment.
- Needs program generation for artifacts, abstractions for complexity.
- *Statemachine: model, embedded controller.*

Software Engineer (this talk)

- Knows (popular) OO, large-scale SW development, reuse, projects, and enough to work with the CSs?
- Program generation (DSLs) as a way to maintain productivity (but at a cost)
- *Statemachine: DSL?*

Audience

Scientific computing. Not a comprehensive list. …

Problem [1/2]: Abstraction Adversity

Students:

- Modelling reduced to a simple "code illustration activity"
- Here's the new syntax – same as the old syntax
- Layered execution, interpretation, and code generation

Not just students:

- Abstraction as a way of solving problems at a new level
- Designing good languages

Audience

…

Problem [2/2]: Constraining Complexity

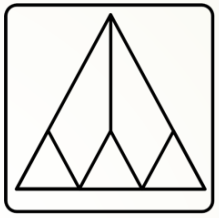
- There: going from text to model
- ..and back again: from model to (high-level) code
- Scoping and type checking: intuitive understanding versus operational knowledge
- Validation of DSL properties: brave new world

Audience

…

Non-Problems: Productive DSL Development

- IDE support (completion, hovering)
- Programming with a language workbench (once the concepts are in place)
  - concrete example: xtext grammar language, xtend as swiss army knife language



Xtext

...

Audience

...

Solutions [1/3]: Setting the stage

Obvious Motivation:

- Easy: systems are difficult to program; repetitive code; DSLs are everywhere
- Hard: dated examples do not motivate, Java is already old and boring

	A310	A320	A340
digital units	77	102	115
size in M	4	10	20
errors per 100K	hun-dreds	dozens	<10

A340, auto-generated code
- fly-by-wire: 70%
- automatic flight control: 70%
- display computer: 50%
- warning and maintenance: 40%

Audience

…

Solutions [1/3]: Setting the stage

Modelling as Abstraction:

- Easy: metamodeling is new and exotic and yet known territory; push in the right direction
- Hard: what does it really mean, where is the code?

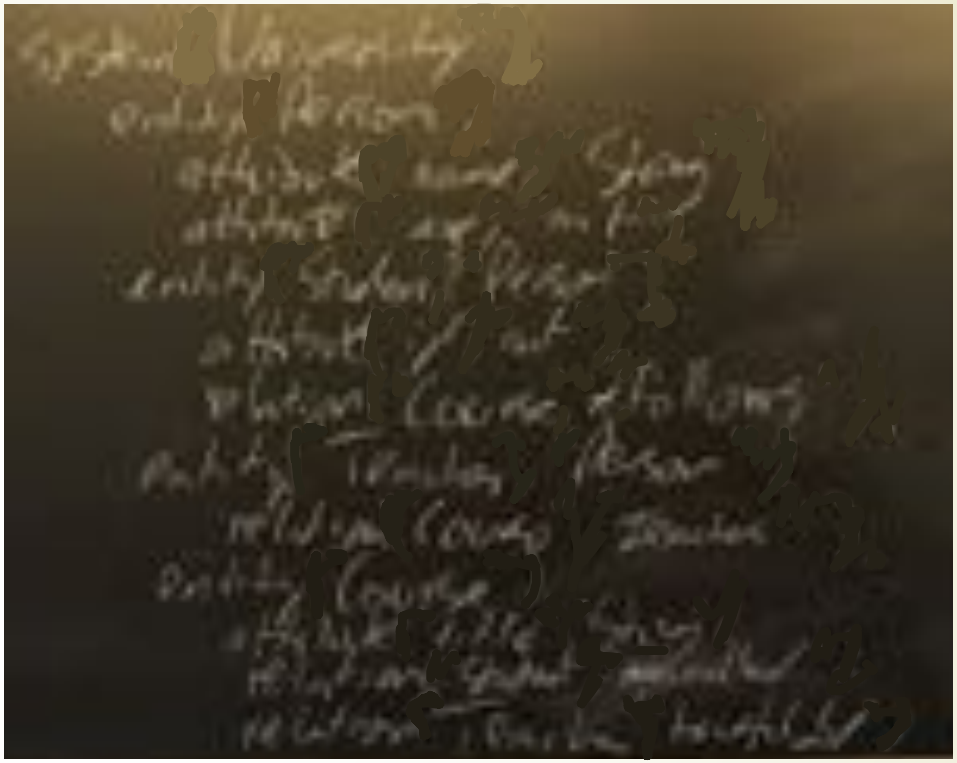
Audience

…

Solutions [2/3]: Bridging the gap

Internal DSL Stepping Stone:

- Easy: everything's included, easy to open the hood (metamodel abstraction, syntax and semantics, interpretation and code generation)
- Hard: DSL vs fluent interface vs bunch of classes



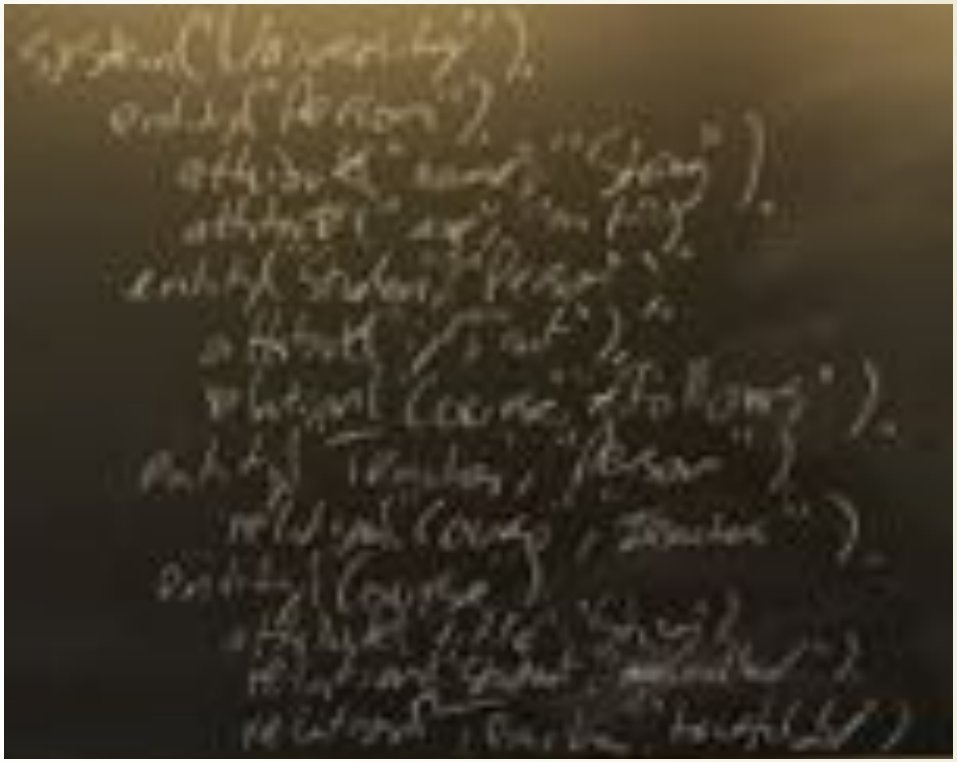
Audience

…

Solutions [2/3]: Bridging the gap

Internal DSL Stepping Stone:

- Easy: everything's included, easy to open the hood (metamodel abstraction, syntax and semantics, interpretation and code generation)
- Hard: DSL vs fluent interface vs bunch of classes



Audience

…

Solutions [2/3]: Bridging the gap

Internal DSL Stepping Stone:

- Easy: everything's included, easy to open the hood (metamodel abstraction, syntax and semantics, interpretation and code generation)
- Hard: DSL vs fluent interface vs bunch of classes

```
public class DataFormatter {
  private List<FormatElement> model;
  public String format(Object...inputs) { /* ??? */ }
  public static class Builder {
    private List<FormatElement> model = new ArrayList<>();
    public Builder t(String text) { /* ??? */ }
    public Builder object(int index) { /* ??? */ }
    public Builder day(int index) { /* ??? */ }
    public Builder month(int index) { /* ??? */ }
    public Builder year(int index) { /* ??? */ }
    public DataFormatter end() { /* ??? */ }
  }
}
```

```
DataFormatter f1 = DataFormatter.build().t("Course ").object(0).
                   t(" had start date: ").day(1).t("/").month(1).t("-").year(1).
                   end();
Calendar mdsd = new GregorianCalendar(2019, 6, 2);
System.out.println(f1.format("MDSD",mdsd));
```

Audience

Formatting is not difficult enough, do something more, graphics, web page. Formatting is contrived. External DSL that generates program as introductory example even before the internal DSL (conceptualization is key). Abstract data types as the abstraction. Deep embedded versus shallow embedding. Builder pattern obscures things. …

Solutions [2/3 → 3/3]:

```
protected void build() {
    entity("Person").
      attribute(String.class,"name").
      attribute(Integer.class,"age").
    entity("Course").
      attribute(String.class,"title").
    entity("Student").sub("Person").
      attribute(Integer.class,"id").
      relation_n_n("follows","Course","enrolled").
    entity("Teacher").sub("Person").
      relation_n_1("teaches","Course","taught_by")
    ;
}
```

```
protected void build() {
    integerState("power");
    state("POWER_OFF").
      transition("PLUS").to("POWER_ON").setState("power",MIN_POWER).
    state("POWER_ON").
      transition("PLUS").to("MAX_POWER").whenEq("power",MAX_POWER).
                         incState("power",1).otherwise().
      transition("MINUS").to("POWER_OFF").whenEq("power",MIN_POWER).
                         incState("power",-1).otherwise().
    state("MAX_POWER").
      transition("MINUS").to("POWER_ON").setState("power",MAX_POWER)
    ;
}
```

Audience

…

Solutions [2/3 → 3/3]:

```
entity Person {
    attribute name String
    attribute age Integer
    require age >= 0 && age<120
}
entity Student: Person {
    attribute id Integer
    relation follows *Course inverse *enrolled
    require age>17
}
entity Course {
    attribute title String
}
entity Teacher: Person {
    relation teaches Course inverse taught_by
}
```

```
machine CookingHood
var power = 0
var limit = 6
events PLUS, MINUS
state OFF
  PLUS to ON set power = 1
state ON
  MINUS [ power = 1 ] to OFF set power = 0
    else set power -= 1
  PLUS [ limit = power ] to MAX
    else to MAX set power += 1
state MAX
  MINUS to ON
```

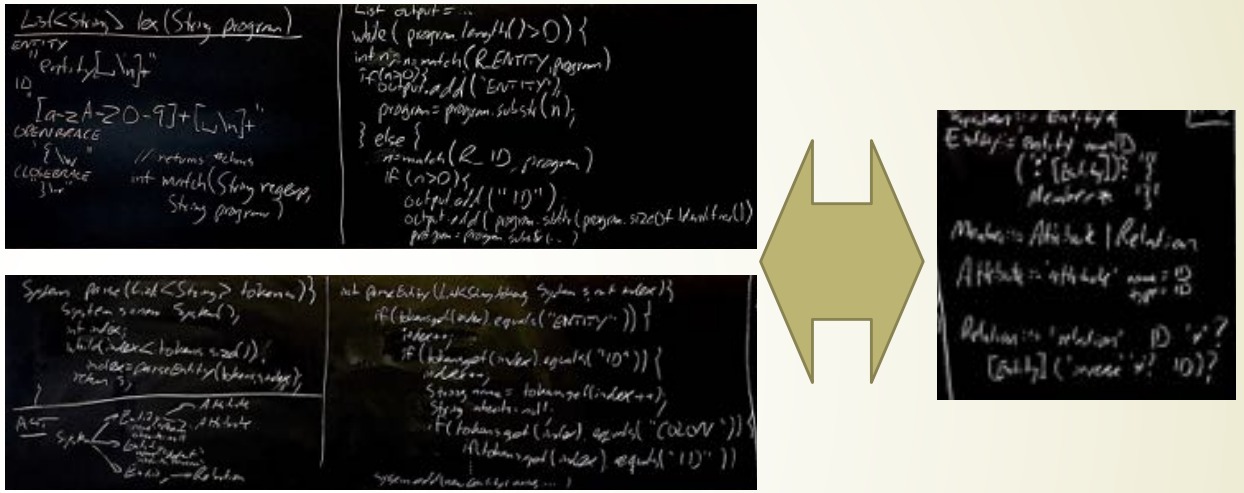
Audience

…

Solutions [3/3]: Taking the leap

External DSL Liberator

- Easy: the joy of automatic parser generation, the simplicity of templates
- Hard: to LL-parse or not to LL-parse, connecting the code generation dots
- Hors categorie: language design



Audience

…

Solutions [3/3]: Taking the leap

Analysis Baby Steps:

- Easy: model validation, scope for DSLs, IDE suggestions
- Hard: the rest - and understanding your limits

```
override getScope(EObject context, EReference reference) {
  // Use of attribute or relation names in expressions
  if(context instanceof Variable && reference==Literals.VARIABLE__VAR) {
    val seen = new HashSet<Entity>
    var entity = EcoreUtil2.getContainerOfType(context,Entity)
    val candidates = new ArrayList<NamedMember>
    while(entity!==null) {
      if(seen.contains(entity)) return super.getScope(context, reference)
      seen.add(entity)
      candidates.addAll(entity.members.filter(NamedMember))
      entity = entity.inherits
    }
    return Scopes.scopeFor(candidates)
  }
  return super.getScope(context, reference)
}
```

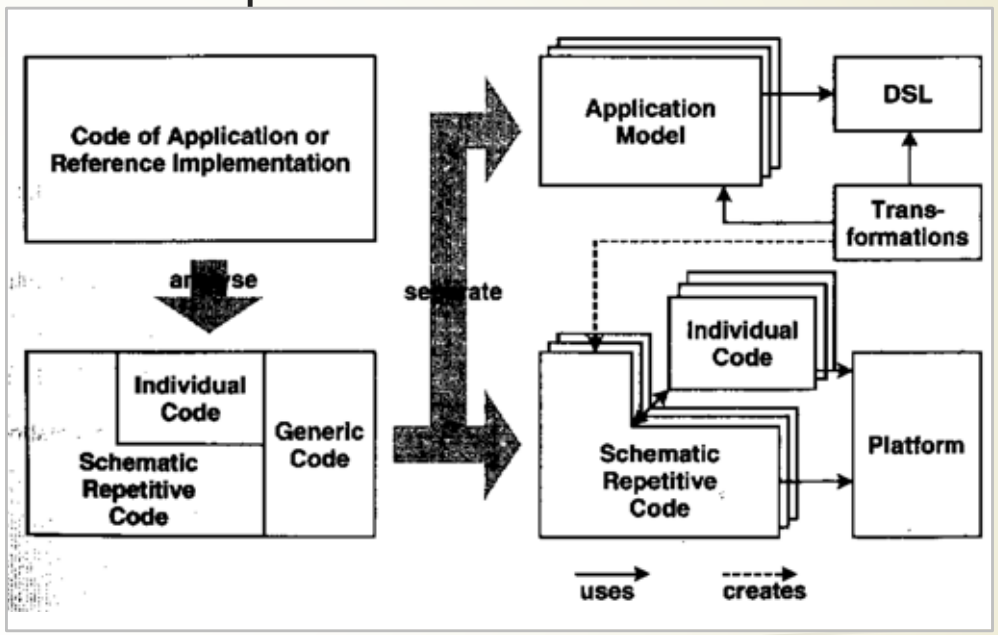
Audience

…

Solutions [3/3]: Taking the leap

Release the Engineer:

- Easy: basic model-driven software development
- Hard: making DSLs that the non-CS will actually use
- Hors de categorie: systematic approach to DSL engineering



Audience

…

Results: What the non-CS can do

Niclas Mølby Niels Heltner

- The software engineer: mixed range of results, some nice design and functionality
  - IOT DSLs: mostly limited functionality but seeding a new way of thinking
  - Microservice modeling
- (The robotics student: ugly DSL that makes the robot do something useful.)

```
template GENERIC_GET(path,
    type)
   /{path}
      GET
         return {type}

template GET_BY(path, type,
    name, returnType)
   implements
       GENERIC_GET(path,
       returnType)
   /{path}/{{type} {name}}
      GET
         return {returnType}
```

```
microservice
     PET_STORE_SERVICE @
     localhost:5000
   implements GET_BY(pet,
       int, id, string[])
```

```
microservice
    PET_STORE_SERVICE @
    localhost:5000
   /pet
      GET
         return string[]
   /pet/{int id}
      GET
         return string[]
```

Audience

…

Results: anything to do for WG 2.11?

- Remember to talk about teaching every once in a while
- Start humble: index (good) resources on teaching program generation
- Realistic longer-term goal?

Audience

…