

CS4200-A: Summary & Further Study

Eelco Visser



CS4200 | Compiler Construction | October 22, 2020

Compiler Components

- What did we study?

Meta-Linguistic Abstraction

- Another perspective

Domain-Specific Languages

- Applying compiler construction in software engineering

Further Study & Research

- Courses and conferences

Research Challenges

- Including topics for master thesis projects

Exam Dates

Compiler Components

What is a Compiler?

A bunch of components for translating programs



Compiler Components

Parser

- Reads in program text, checks that it complies with the syntactic rules of the language, and produces an abstract syntax tree, which represents the underlying (syntactic) structure of the program.

Type checker

- Consumes an abstract syntax tree and checks that the program complies with the static semantic rules of the language. To do that it needs to perform name analysis, relating uses of names to declarations of names, and checks that the types of arguments of operations are consistent with their specification.

Optimizer

- Consumes a (typed) abstract syntax tree and applies transformations that improve the program in various dimensions such as execution time, memory consumption, and energy consumption.

Code generator

- Transforms the (typed, optimized) abstract syntax tree to instructions for a particular computer architecture. (aka instruction selection)

ChocoPy Compiler

Syntax definition

- Parser through generation, design of abstract syntax

Static semantic analysis

- Name analysis
 - ▶ Lexical scoping, type-dependent name resolution
- Type checking
 - ▶ Class-based object-oriented language with sub-typing

Desugaring

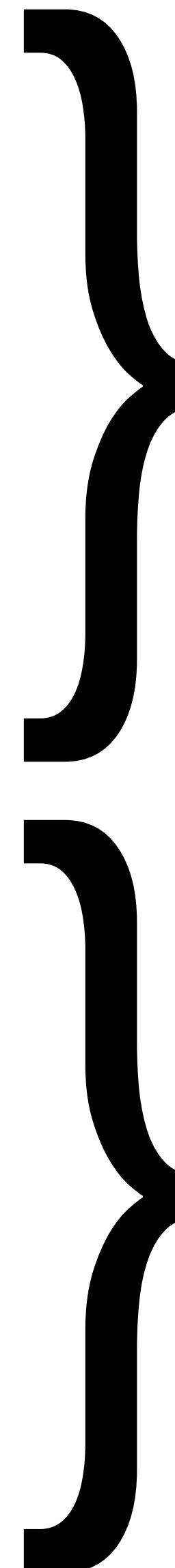
- Simple rewrite rules and strategies

Code generation

- Generation of Risc V instructions
- AST-to-AST transformation

Data-flow analysis

- Optimization

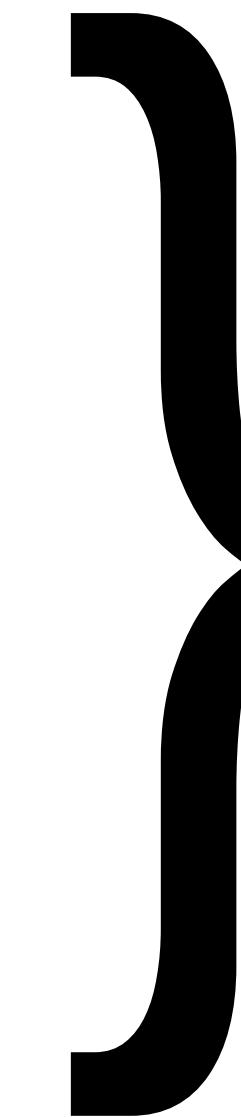


CS4200-A

CS4200-B

More Compiler Components

- Static analyses
- Optimization
- Register allocation
- Code generation for register machines
- Garbage collection



CS4200-B

Other Object Languages

- Functional programming: first-class functions, laziness
- Domain-specific languages: less direct execution models
- Data (description) languages
- Query languages
- ...

Meta-Linguistic Abstraction

Separation of Concerns

Language design

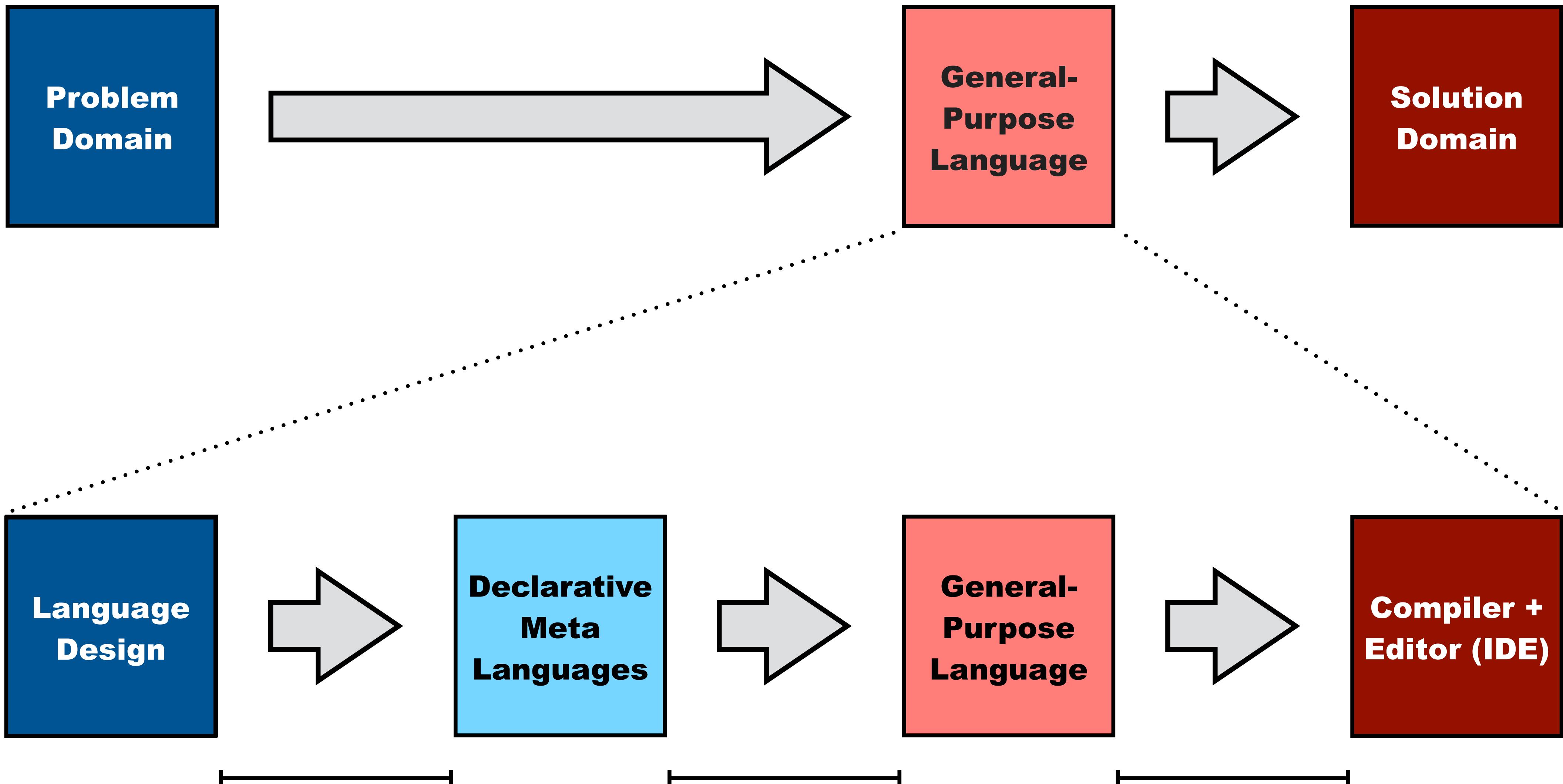
- Define the properties of a language
- Done by a language designer

Language implementation

- Implement tools that satisfy properties of the language
- Done by a language implementer

Can we automate the language implementer?

- That is what language workbenches attempt to do



That also applies to the definition of (compilers for) general purpose languages

Declarative Language Definition

Objective

- A workbench supporting design and implementation of programming languages

Approach

- Declarative multi-purpose domain-specific meta-languages

Meta-Languages

- Languages for defining languages

Domain-Specific

- Linguistic abstractions for domain of language definition (syntax, names, types, ...)

Multi-Purpose

- Derivation of interpreters, compilers, rich editors, documentation, and verification from single source

Declarative

- Focus on what not how; avoid bias to particular purpose in language definition

Spoofax Meta-Languages

SDF3: Syntax definition

- context-free grammars + disambiguation + constructors + templates
- derivation of parser, formatter, syntax highlighting, ...

Statix: Names & Types

- name resolution with scope graphs
- type checking/inference with constraints
- derivation of name & type resolution algorithm

Stratego: Program Transformation

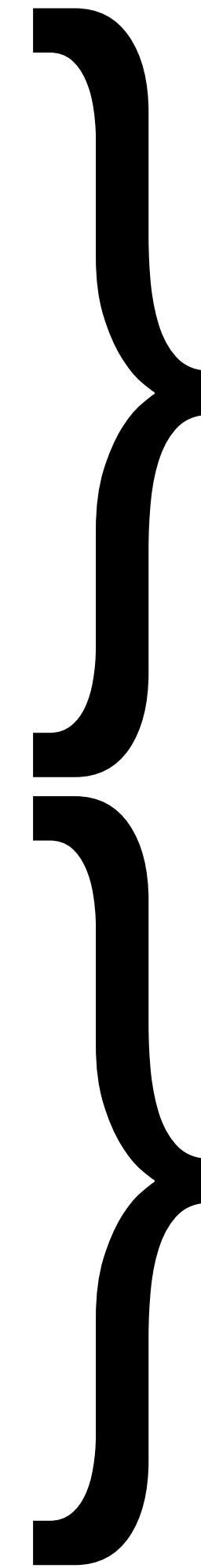
- term rewrite rules with programmable rewriting strategies
- derivation of program transformation system

FlowSpec: Data-Flow Analysis

- extraction of control-flow graph and specification of data-flow rules
- derivation of data-flow analysis engine

DynSem: Dynamic Semantics

- specification of operational (natural) semantics
- derivation of interpreter



CS4200-A

CS4200-B

Compiler construction is a lot of fun ...

... but when would I ever implement a programming language?

Domain-Specific Languages

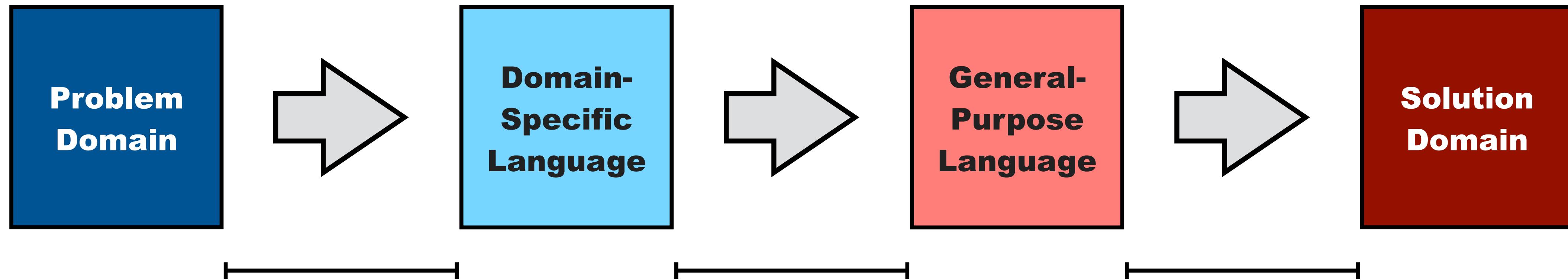
Traditional Compiler

Source: high-level machine language



Target: low-level machine language

Domain-Specific Language



Domain-specific language (DSL)

noun

1. a programming language that provides notation, analysis, verification, and optimization specialized to an application domain
2. result of linguistic abstraction beyond general-purpose computation

DSL Compiler

Source: domain-specific language



Target: high-level machine language

Same architecture, techniques as traditional compiler

Domain

- Graph analytics

Design

- Domain-specific graph traversal, aggregation

Implementation

- Compiler introduces parallel implementation
- Back-ends with different characteristics (parallel, distributed, ...)

Applications

- Many graph analytics algorithms such as page rank, ...

Domain

- Web programming

Design

- Sub-languages for sub-domains
 - ▶ Entities, Queries, UI (Pages, Templates, Actions), Search, Access Control
- Type checker checks cross-domain consistency

Implementation

- Generate Java code with web libraries
 - ▶ Hibernate (ORM), Lucene (search), ...

WebDSL Applications

The screenshot shows the SPLASH 2020 conference website. At the top, there's a banner with the Chicago skyline at night. Below it, a large image of the city skyline is labeled "Downtown". A navigation bar includes links for "Attending", "Tracks", "Organization", "Search", "Series", "Sign in", and "Sign up". A "Write a Blog" button is also present. The main content area has a section titled "SPLASH 2020" with a message about the website being under construction. It features a "SPLASH 2020 Tracks" sidebar with links to "OOPSLA | Onward! Essays", "Papers", "Rebase", and "Workshops". Another sidebar lists "Upcoming Important Dates" from April to July 2020, including "Wed 15 Apr 2020 OOPSLA Paper Submission" and "Thu 11 - Tue 16 Jun 2020 OOPSLA Author Response".

The screenshot shows a code editor interface for a Java assignment. The title bar indicates the URL is <http://weblab.tudelft.nl/tl2606/2017-2018/assignment/14943/submission/9215/view>. The page header includes "Actions", "0.010.0", "1 2 3", "Submission info", and "Edit Assignment". The main content area shows a Java code snippet:

```
1 import Library._;
2 import Untyped._;
3
4 case class NotImplementedException() extends ParseException(")");
5
6 object Parser {
7     def parse(str: String): ExprExt = parse(Reader.read(str))
8
9     def parse(sexpr: SExpr): ExprExt = {
10         throw NotImplementedException()
11     }
12 }
13
14 object Desugar {
15     def desugar(e: ExprExt): ExprC = {
16         throw NotImplementedException()
17     }
18 }
19
20 object Interp {
21     def interp(e: ExprC): Value = interp(e, Nil)
22
23     def interp(e: ExprC, nv: Environment): Value = {
24         throw NotImplementedException()
25     }
26 }
27
28 }
```

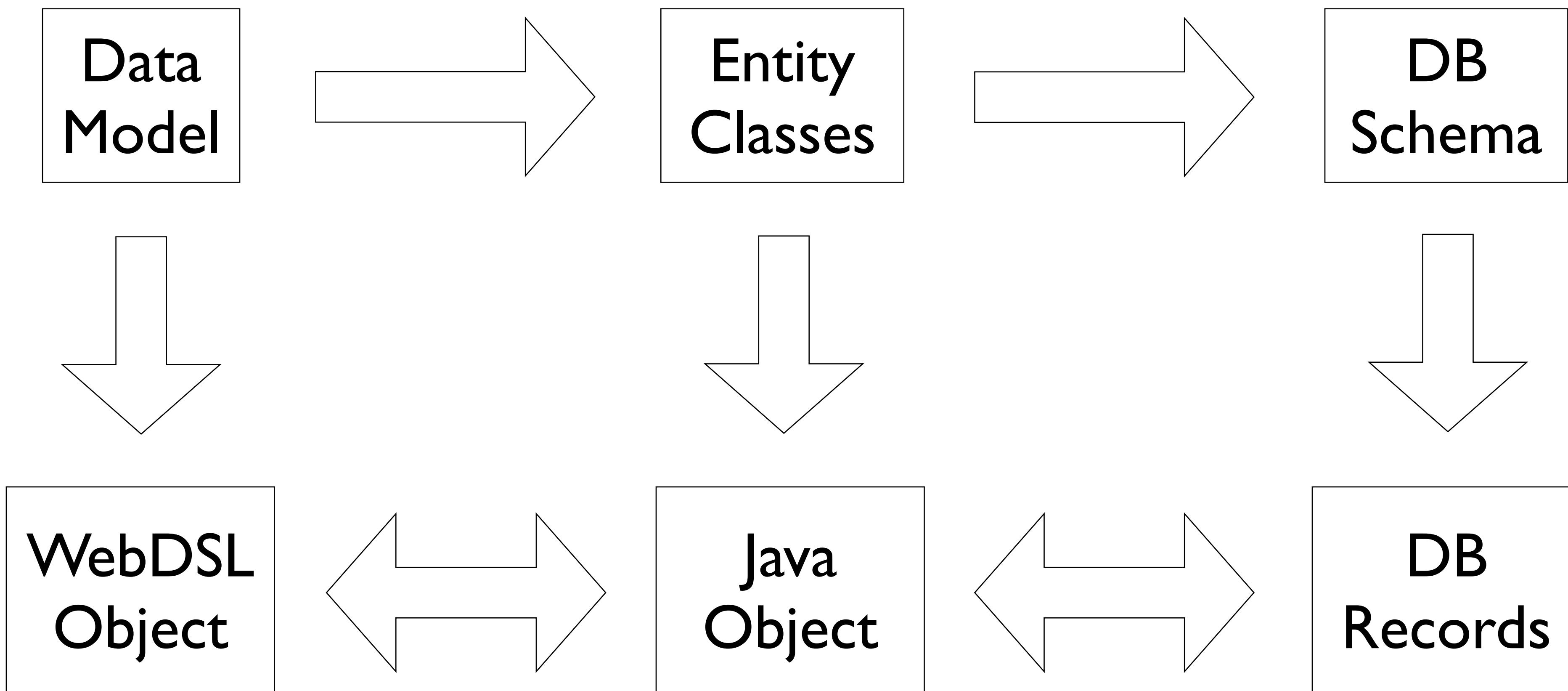
Below the code, there are tabs for "Console", "Grading", "Discussion", and "Revision History". A "Compile" button is at the bottom left, and a "Your Test" button is highlighted in blue. The status bar at the bottom says "Spec-tests ran: 0".

The screenshot shows a researcher profile page for Elco Visser. The URL is <http://researchr.org/profile/eelcovisser>. The page includes sections for "All Publications", "External Links", "Profile Name", "Statistics", and "Aliases". A large photo of Elco Visser is on the right. The "About" section provides a brief biography: "Elco Visser is Antoni van Leeuwenhoek Professor of Computer Science at Delft University of Technology. He received a master's and doctorate in computer science from the University of Amsterdam in 1993 and 1997, respectively. Previously he served as postdoc at the Oregon Graduate Institute, as Assistant Professor at Utrecht University, and as Associate Professor at TU Delft. His research interests include software language engineering, domain-specific programming languages, model-driven engineering, program transformation, software deployment, and interaction design. With his students he has designed and implemented the *Spoofax language workbench*, as well as many domain-specific languages, including DSLs for syntax definition (SDF), program transformation (Stratego), software deployment (Nix), web application development (WebDSL), and mobile phone applications (mobi). He is the lead developer of the researcher bibliography management system." The "AFFILIATIONS" section lists his academic history: "2006 - : Delft University of Technology", "1998 - 2006: Utrecht University", "1997 - 1998: Oregon Graduate Institute", and "1993 - 1997: University of Amsterdam". The "RECENT PUBLICATIONS" section lists several papers, including "A Research Agenda for Formal Methods in the Netherlands" (with Marieke Huisman, Wouter Swierstra, Elco Visser), "Fast and Safe Linguistic Abstraction for the Masses" (with Elco Visser), "From Definitional Interpreter to Symbolic Executor" (with Mensing, Adrian D., Hendrik van Antwerpen, Casper Bach Poulsen, Elco Visser), "Precise, Efficient, and Expressive Incremental Build Scripts with PIE" (with Gabriel Konat, Roelof Sol, Sebastian Erdweg, Elco Visser), and "In Second Workshop on Incremental Computing (IC 2019), 2019: [X] Toward language-parametric ergonomic editor concepts based on derivation type system specifications".

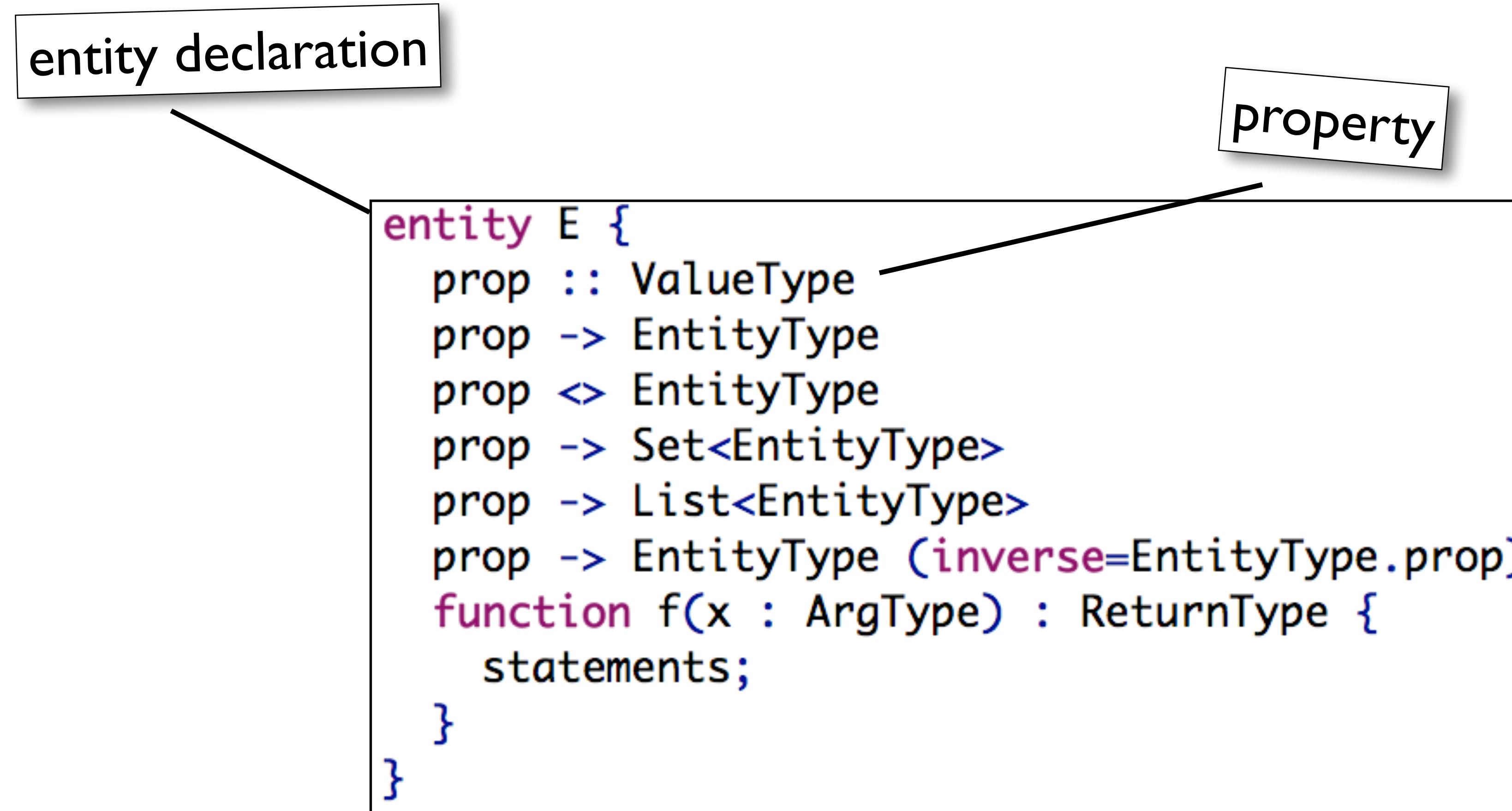
The screenshot shows a "My Study Planning" application interface. The title bar says "mystudyplanning.tudelft.nl/scheduler#year?_k=2ywzaz". The top navigation bar includes links for "Overview Supervisor", "Student Overview", "Course Statistics", "Demo Track Select", "Demo Planning", "Demo Track Req", "Demo Add Additional Courses", "Demo Submit", "Demo Submissions", and "History". The user "Eelco Visser" is signed in. The main content area displays a grid of course offerings for the year 2016. Courses include "IN4301 Advanced Algorithms" (EC 34), "IN4152 3D Computer Graphics and Animation" (EC 5), "CS4065 Multimedia Search and Recommendation" (EC 5), "IN4306 Literature Survey" (EC 10), "IN4150 Distributed Algorithms" (EC 6), "IN4130 Seminar Programming Languages" (EC 5), "CS4090 Quantum Communication and Cryptography" (EC 5), "IN4252 Web Science & Engineering" (EC 5), "IN4191 Security and Cryptography" (EC 5), "CS4106 Language-Based Software Security" (EC 5), "IN4085 Pattern Recognition" (EC 6), "IN4073TU Embedded Real-Time Systems" (EC 6), and "CS4015 Behaviour Change Support Systems" (EC 5). A sidebar on the left lists various academic categories like "Common Core ST 2016", "Specialisation courses start first period 2016", etc.

The screenshot shows the "EvaTool - Education Evaluation" dashboard. The title bar says "evatool.tudelft.nl". The top navigation bar includes "EvaTool", "Administration", "People", "Committees", "Faculty", and "Prof. Dr. E. Visser". The main content area has a section titled "Your Dashboard" with tabs for "Educations" and "Your Courses". It features a search bar for "Filter by course name, code or period" and three dropdown filters for "Education period", "Education level", and "Education discipline", all set to "Nothing to filter". Below this is a "Results" section with a table showing course data, though the table content is mostly obscured by a large watermark.

WebDSL: Automatic Persistence



WebDSL: Entity Declarations



WebDSL: Page Definition & Navigation

page navigation (page call)

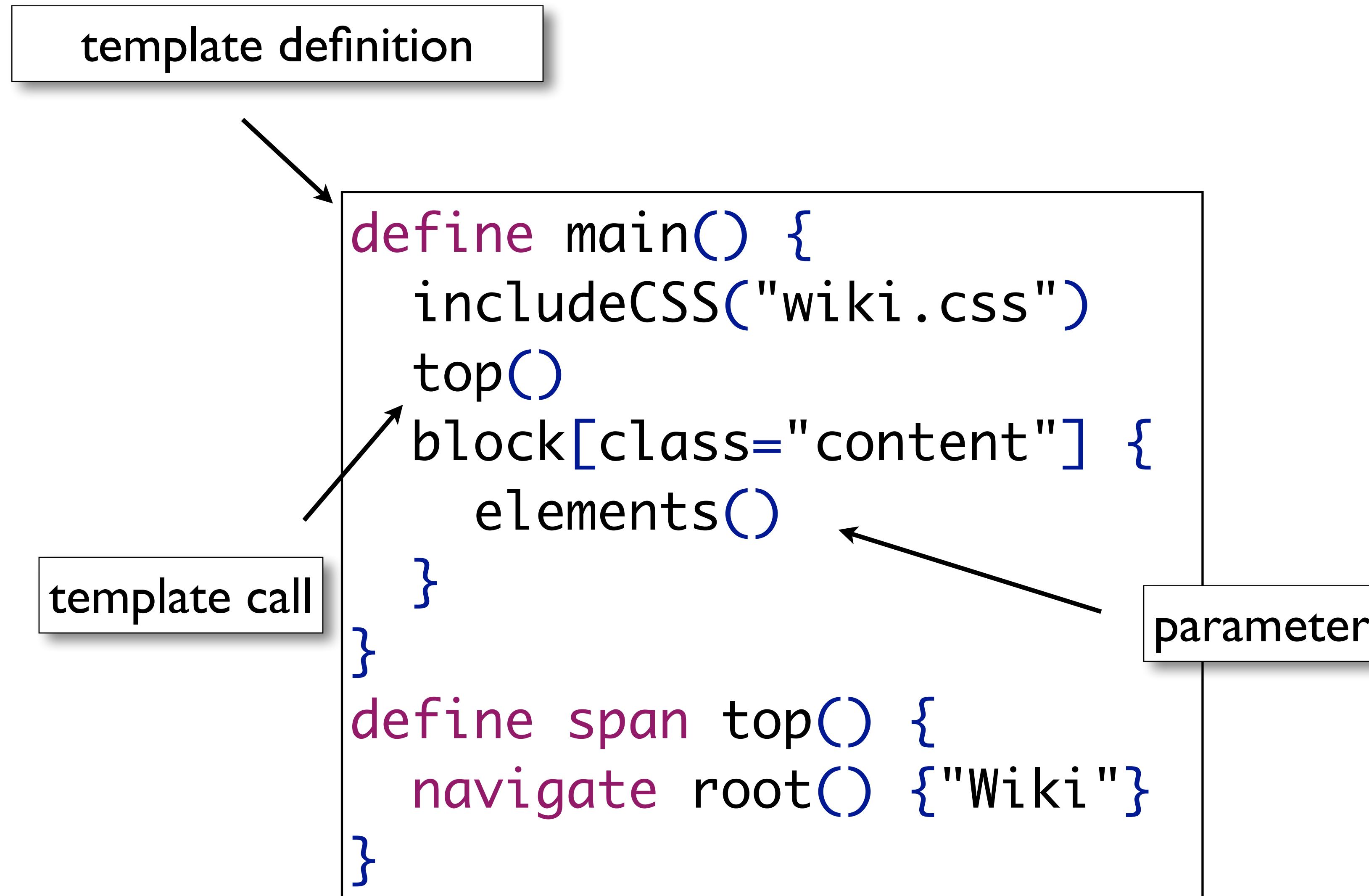
```
entity A { b -> B }
entity B { name :: String }

define page a(x : A) {
    navigate b(x.b){ output(x.b.name) }
}

define page b(y : B) {
    output(y.name)
}
```

Page definition

WebDSL: Templates (Page Fragments)



WebDSL: Forms

```
define page editpage(p : Page) {  
    main{  
        header{output(p.name) " (Edit)"}  
        form{  
            input(p.content)  
            submit action{ return page(p); } { "Save" }  
        }  
    }  
}
```

data
binding

page
flow

no separate controller: page renders form *and* handles form submission

WebDSL: Search

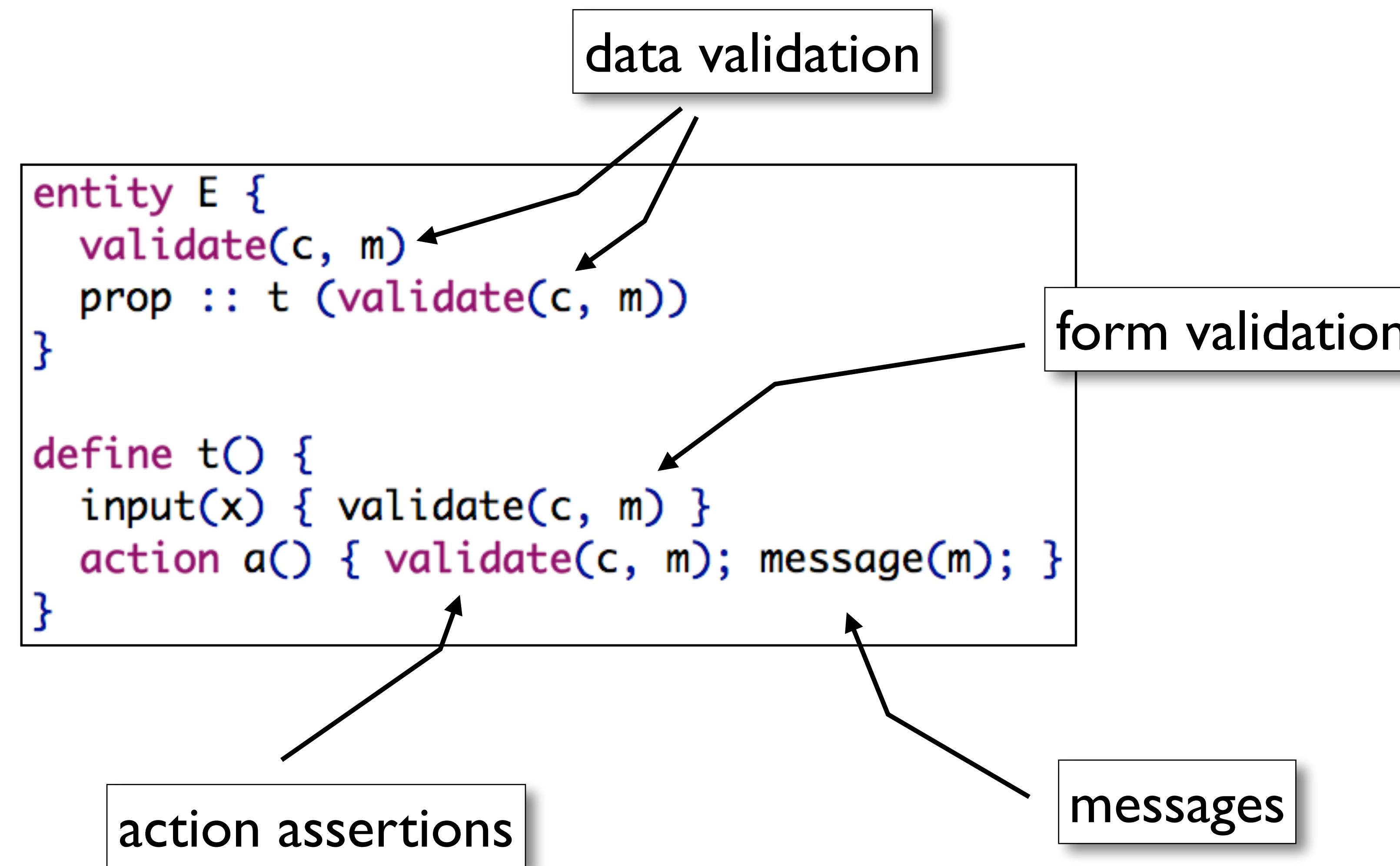
```
entity Page {  
    name      :: String (id,searchable)  
    content   :: WikiText (searchable)  
    modified   :: DateTime  
    authorSearch :: String (searchable) := authorNames()  
}  
  
define page search(query : String) {  
    var newQuery : String := query;  
    form {  
        input(newQuery)  
        submit action{ return search(newQuery); } {"Search"}  
    }  
    for(m : Message in searchPage(query, 50)) {  
        output(m)  
    }  
}
```

search annotations

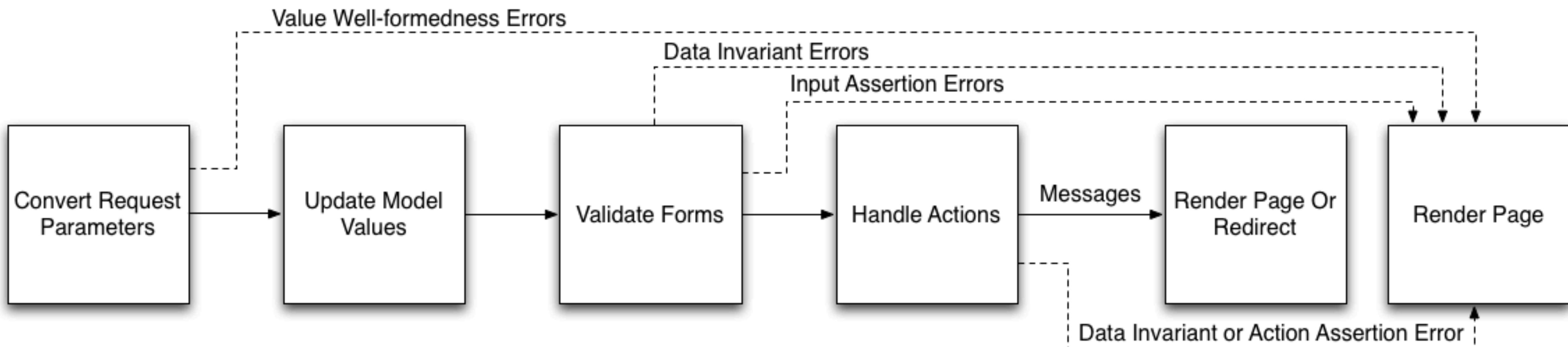
search queries

The diagram illustrates the flow of data and annotations in a WebDSL search application. It consists of two main components: 'search annotations' at the top and 'search queries' at the bottom. A vertical line connects them. Two arrows point from the 'search annotations' box to the 'authorSearch' field in the entity definition and to the 'search' function call in the search logic. Another arrow points from the 'search queries' box up to the 'output(m)' statement in the search logic.

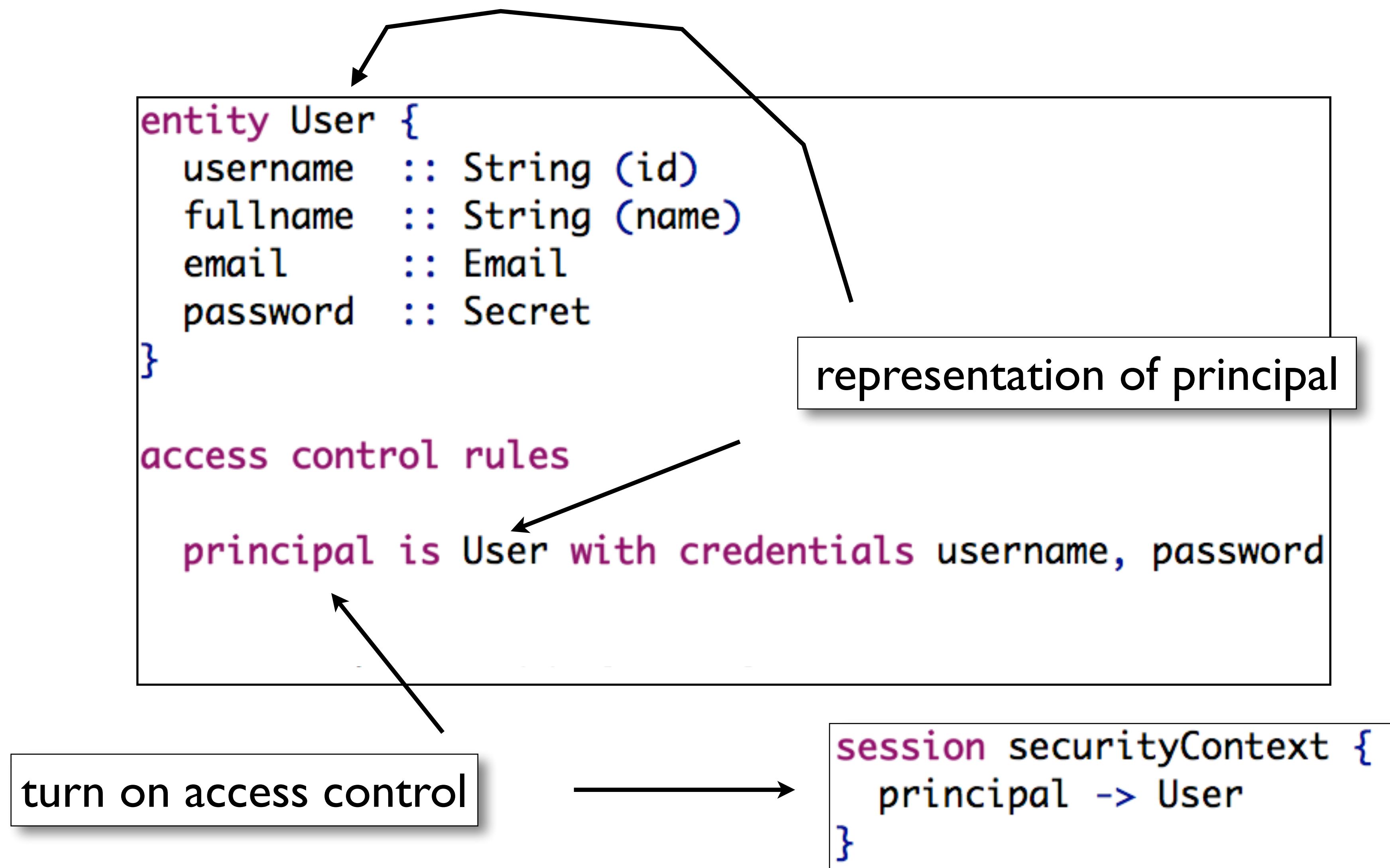
WebDSL: Validation Rules



WebDSL: Data Validation Lifecycle



WebDSL: securityContext



WebDSL: Authentication

```
define page signin() {
    var username : String
    var password : Secret
    action doit(){ signin(username, password); }
    main{
        header{"Sign In"}
        form{
            par{ label("Username: "){ input(username) } }
            par{ label("Password: "){ input(password) } }
            par{ action("Sign in", doit()) }
        }
        section{
            header{"Register"}
            par{ "No account? " navigate(register()){ "Register now" } }
        }
    }
}
```

WebDSL Wiki Signin

Sign In

Username:

Password:

Register

No account? [Register now](#)

WebDSL: Access Control Rules

access control rules

```
rule template {*} { true }

rule page page(n : String) {
    loggedIn() || findPage(n) != null
}

rule page editpage(p : Page) {
    loggedIn()
}
```

‘anyone can view existing pages, only logged in users can create pages’

‘only logged in users may edit pages’

WebDSL: Linguistic Integration

Data models

- automatic persistence

User interface templates

- parameterized definition of page fragments
- request and response handling

Data validation

- form validation & data integrity

Access control rules and policies

- through constraints over objects

IceDust: Computing with Derived Values

Domain

- Information systems
- Data modelling with derived values

Design

- Native multiplicities and relations
- Different strategies for (re-)computing derived values
 - ▶ On demand (on read), incremental (on write), eventual (eventually consistent)

Implementation

- Generate WebDSL code
- Strategy implementation based on static dependency analysis

Applications

- WebLab grading logic

IceDust: Grading Logic

```
entity Submission {  
    pass      : Boolean = grade >= 5.5 <+ false  
    grade     : Float?   = if(conj(children.pass))  
                           avg(children.grade)  
}  
  
entity Assignment {  
    avgGrade : Float? = avg(submissions.grade)  
}  
  
relation Assignment.parent ? <-> * Assignment.children  
relation Submission.assignment 1 <-> * Assignment.submissions  
relation Submission.parent ? <-> * Submission.children
```

IceDust: Grading Logic

```
gradeWeighted: Float = if(weightCustom > 0.0) totalGrade / weightCustom <+ 0.0 else totalGrade (inline)
gradeRounded : Float = max(gradeWeighted - (sub.penalty <+ 0.0) ++ 1.0).round1() (inline)

gradeOnTime  : Float = if(sub.onTime <+ false) gradeRounded else 0.0 (inline)

maxNotPassed : Float = max(0.0 ++ assignment.minimumToPass - 0.5).round1() (inline)
passSub      : Boolean = sub.filter(:AssignmentCollectionSubmission).passSub <+ true (inline)
maxNotPass   : Float = if(passSub) gradeOnTime else min(gradeOnTime ++ maxNotPassed) (inline)

grade        : Float = min(maxNotPass ++ scheme.maxGrade) (eventual)
```

Domain

- Client-side web programming

Design

- Web views as IceDust-style derived values
- Incremental update of view based on changes in model

Implementation

- Generate JavaScript code
- Strategy implementation based on static dependency analysis

Applications

- Small toy application(s)

PixieDust: Model & View

```
model
entity TodoList {
    todos : Todo* (inverse = Todo.list)
}
entity Todo {
    description : String
    finished     : Boolean
}

view
TodoList.view = div { ul { todos.itemView } }

Todo.itemView = li {
    input[type="checkbox", value=finished]
    span { description }
}
```

```
view
TodoList {
    input : String = (init = "")
    show  : String = (init = "All")

    finishedTodos : Todo* =
        todos.filter(todo => todo.finished)
        (inverse = Todo.inverseFinishedTodos?)

    visibleTodos : Todo* =
        switch {
            case show == "All"      => todos
            case show == "Finished" => finishedTodos
            default => todos \ finishedTodos
        }
        (inverse = Todo.inverseVisibleTodos?)
}
```

```
view
TodoList {
    view : View = div {
        header
        ul { visibleTodos.itemView }
        footer
    }

    header : View = div {
        h1 { "Todos" }
        input[type="checkbox", value = allFinished,
              onClick = toggleAll]
        StringInput[onClick = addTodo](input)
    }

    footer : View = div {
        todosLeft "items left"
        ul{
            visibilityButton(this, "All")
            visibilityButton(this, "Finished")
            visibilityButton(this, "Not finished")
        }
        if(count(finishedTodos) > 0)
            button[onClick = clearFinished]
    }
}

Todo {
    itemView : View = li { div {
        BooleanInput(finished)
        span { task }
        button[onClick=deleteTodo] { "X" }
    } }
}
```

PIE: Interactive Software Pipelines

Domain

- Build systems, software pipelines

Design

- Define tasks as functions
- Dynamic dependencies
- Incrementally recompute only tasks affected by a change

Implementation

- Generate Kotlin code
- Run-time dependency analysis

Applications

- Spoofax build, benchmarking pipeline

PIE: Parsing Pipeline

```
typealias In = Serializable; typealias Out = Serializable
interface Func<in I:In, out O:Out> {
    fun ExecContext.exec(input: I): O
}
interface ExecContext {
    fun <I:In, O:Out, F:Func<I, O>> requireCall(clazz: KClass<F>, input: I,
        stamper: OutputStamper = OutputStampers.equals): O
    fun require(path: PPath, stamper: PathStamper = PathStampers.modified)
    fun generate(path: PPath, stamper: PathStamper = PathStampers.hash)
}

class GenerateTable: Func<PPath, PPath> {
    override fun ExecContext.exec(syntaxFile: PPath): PPath {
        require(syntaxFile); val tableFile = generateTable(syntaxFile);
        generate(tableFile); return tableFile
    }
}
class Parse: Func<Parse.Input, ParseResult> {
    data class Input(val tableFile: PPath, val text: String): Serializable
    override fun ExecContext.exec(input: Input): ParseResult {
        require(input.tableFile); return parse(input.tableFile, input.text)
    }
}
class UpdateEditor: Func<String, ParseResult> {
    override fun ExecContext.exec(text: String): ParseResult {
        val tableFile = requireCall(GenerateTable::class, path("syntax.sdf3"))
        return requireCall(Parse::class, Parse.Input(tableFile, text))
    }
}
```

Research Challenges in Compiler Construction

Vision: Language Designer's Workbench

High-Level Declarative Language Definition

- Human readable / understandable definition
- Serves as reference documentation

Verification

- Automatically verify properties of language definition
- Type soundness of interpretation
- Type preservation of transformations
- Semantics preservation of transformation

Implementation

- Generate production quality tools from language definition
- Interpreter, compiler, IDE with refactoring, completion, ...
- Correct-by-construction, high performance

High-Performance Parsing

- JSGLR2: 2x to 10x speed-up compared to JSGLR
- More speed-up possible?
- Explore effects of different parse table formats (LR, SLR, LALR)

Error Recovery & Error Messages

- Apply error recovery approach of [TOPLAS12] to JSGLR2
- Generate high quality error messages

Incremental Parsing

- Re-parse effort proportional to change of program text
- Approach: adapt Graham/Wagner algorithm to SGLR

Extensible Syntax

- Extend syntax during parsing to support extensible languages

Workbench / Editor Services

Code Completion

- Semantic code completion based on static semantics

Refactoring

- Sound refactoring scripts
- Refactoring based on scope graph program model
- New NWO MasCot project: programming and validating software restructurings

Live Language Development

- Immediate response after edit of language definition
- Requires: incremental evaluation of all compiler components
- Ongoing work: PIE DSL for interactive software development pipelines

Language Deployment

- Generate stand-alone language implementation: PIE partial evaluation

Workbench / Editor Services

Portable Editors

- Portable editor bindings based on AESI model (Pelsmaeker)
- Case study: bindings for Visual Studio, IntelliJ, LSP

Web Editors

- Generate language-specific editors for use in web browser
- Architectural questions
 - ▶ All processing client-side? Stateful back-end on server? Scalability?
 - ▶ Performance of Web Assembly (WASM) better than JS?
- Collaborative editing (operational transform)

Interactive Notebooks

- Combine documents with code in several languages and results of execution

Specification of type systems with Statix

- Subset of CHR (Constraint Handling Rules) + domain-specific constraints for scope graphs and relations
- Support more advanced type systems
- Structural types, polymorphism (generics), sub-typing [OOPSLA'18]
 - ▶ Better encoding?
 - ▶ Generalization (for parametric polymorphism)?

Solver

- Matrix-based name resolution algorithm?
- Correctness wrt resolution calculus?
- Scalability: modular and incremental analysis?

Exploring Type System Design Landscape

Substructural Type Systems

- Linear types
- Rust

Gradual Type Systems

- Gradual type theory: encode calculi and experiment
- Implement existing gradual type checkers
 - ▶ Python, TypeScript, Dart, Hack
- Design gradual type system for Stratego

Dependent Types

- Agda, Idris

Program Model

- Extend term data model to incorporate scopes and types
- Persistent storage
- Query: retrieve information based on scope graph model
 - ▶ All methods in class A
- Construction
 - ▶ well-formed wrt static semantics

Random Program Generation

- Generation of well-formed and well-typed programs
- based on syntax + static semantics
- for testing compilers and other language processing tools

Theme: Incremental Compilation

Make all (meta) language processing incremental

- Effort proportional to size of change

Modular analysis out of the box

- Static analysis incremental based on (scope graph) dependencies

Compiler = build system

- Use PIE to glue together language processing pipelines

In progress

- Incremental parsing
- Incremental compilation for Stratego (in Beta)
- Incremental compilation for WebDSL

Theme: Error Localization and Diagnosis

Error Localization

- What program element is responsible for the failure?
- Minimal unsatisfiable core
 - What is the smallest set of constraints that correspond to failure?

Error Diagnosis

- Generate good (understandable) explanation of error
- Based on unsat core

Studying Programming Languages

Courses

Compiler Construction B (Q2)

- Study back-end components of compiler

Software Verification (Q3)

- Learn the basics of mechanised verification with Agda dependently typed programming language

Web Programming Languages (Q3)

Language-Based Software Security (Q4)

Language Engineering Project (Q4)

- Develop a Spooftax language definition for an interesting language

Seminar Programming Languages (Q1)

- Read and discuss papers from the PL literature

System Validation (Q1)

- Check properties of (concurrent) software with model checking

Master Thesis Project in PL group

Industrial Internships

Oracle Labs (Zürich)

- Applications of Spoofax: GreenMarl, PGQL
- Other projects (Truffle/Graal)

Canon (Venlo)

- Designs and manufactures digital printers
- New project to investigate design of DSLs in digital printing domain

Philips (Best)

- Software restructuring

Other

- Opportunities for language design and implementation projects at other companies

Conferences

ACM Special Interest Group on Programming Languages

- <http://sigplan.org/>

Key SIGPLAN Conferences

- POPL: Principles of Programming Languages
- PLDI: Programming Language Design and Implementation
- ICFP: International Conference on Functional Programming
- OOPSLA/SPLASH: Systems, Programming Languages, and Applications
- SLE: Software Language Engineering
- GPCE: Generative Programming

Other Conferences

- ECOOP: European PL conference
- ESOP: European Symposium on Programming

Summer Schools

PLMW: Programming Languages Mentoring Workshop

- technical sessions on cutting-edge research in programming languages, and mentoring sessions on how to prepare for a research career
- At ICFP, POPL, PLDI, SPLASH

OPLSS: Oregon Programming Languages Summer School

- Foundational work on semantics and type theory
- Advanced program verification techniques
- Experience with applying the theory

DSSS: DeepSpec Summer School

- Formal verification

PLISS: Programming Language Implementation Summer School

- Programming language systems, implementation, analysis

PhD

- Dive into PL research for four years
- Develop new PL theory, designs, and implementations
- Write research papers and a dissertation
- ~~Present your work at conferences around the world~~

PL in industry

- Develop compilers, analyses, run-time systems
- Contribute to development of industrial programming languages
 - Oracle Labs (PGX), Google (Dart), Amazon (Cloud9), Canon (OIL)

Wanted: PhD Students in PL

Software Restructuring

- A principled approach to programming refactorings/restructurings
- Application: Transforming C++ code

Language Engineering

- Static semantics and type checking
- Deriving interpreters, compilers from dynamic semantics

Dependently Typed Programming

- Contributing to the semantics and implementation of Agda

Wanted: Grammar Engineer

Goal

- A collection of high quality syntax definitions for key languages
- Spoofax with `batteries included'
- Speeding up research case studies

Developing Syntax Definitions

- High quality
- High coverage

Research Assistant

- 4 - 8 hours per week (flexible)
- Appointment per project (language)

Wanted: Web Programmer

Academic Workflow Engineering

- Make university work better with web apps that automate workflows
- Education
 - ▶ WebLab, mystudyplanning, EvaTool
- Research
 - ▶ [conf.researchr.org](#), [researchr.org](#)
- Administration

Combine with PL research

- Use high-level web PLs (WebDSL, IceDust)
- Contribute to better abstractions for web programming

Exam

Exam and Resit

November 29: Exam

- 13:30-16:30

January 22: Resit

- 13:30-16:30

Topics

- Everything we studied in the lectures
- Example exam questions: homework assignments

Except where otherwise noted, this work is licensed under

