

CS4200-A: Summary & Further Study

Eelco Visser



CS4200 | Compiler Construction | October 21, 2021

Outline

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

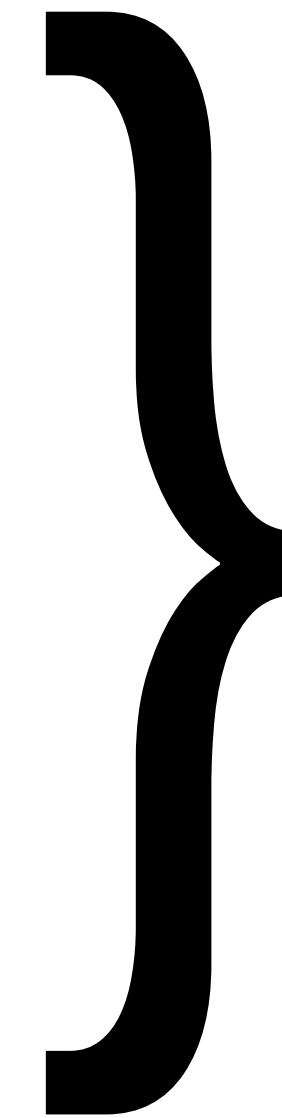
- Simple rewrite rules and strategies

Code generation

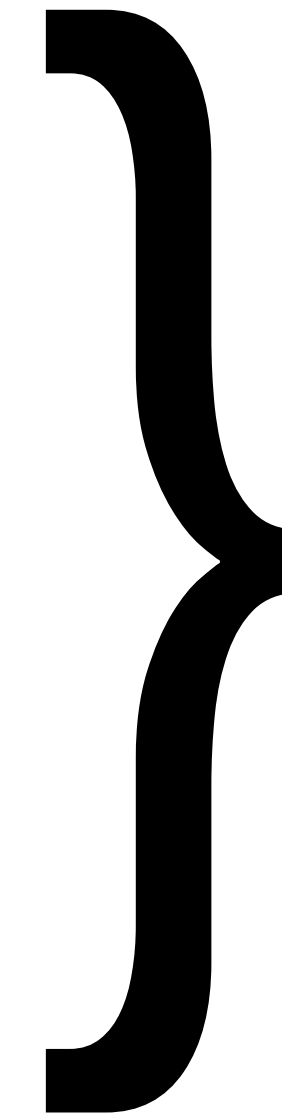
- Generation of Risc V instructions
- AST-to-AST transformation

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

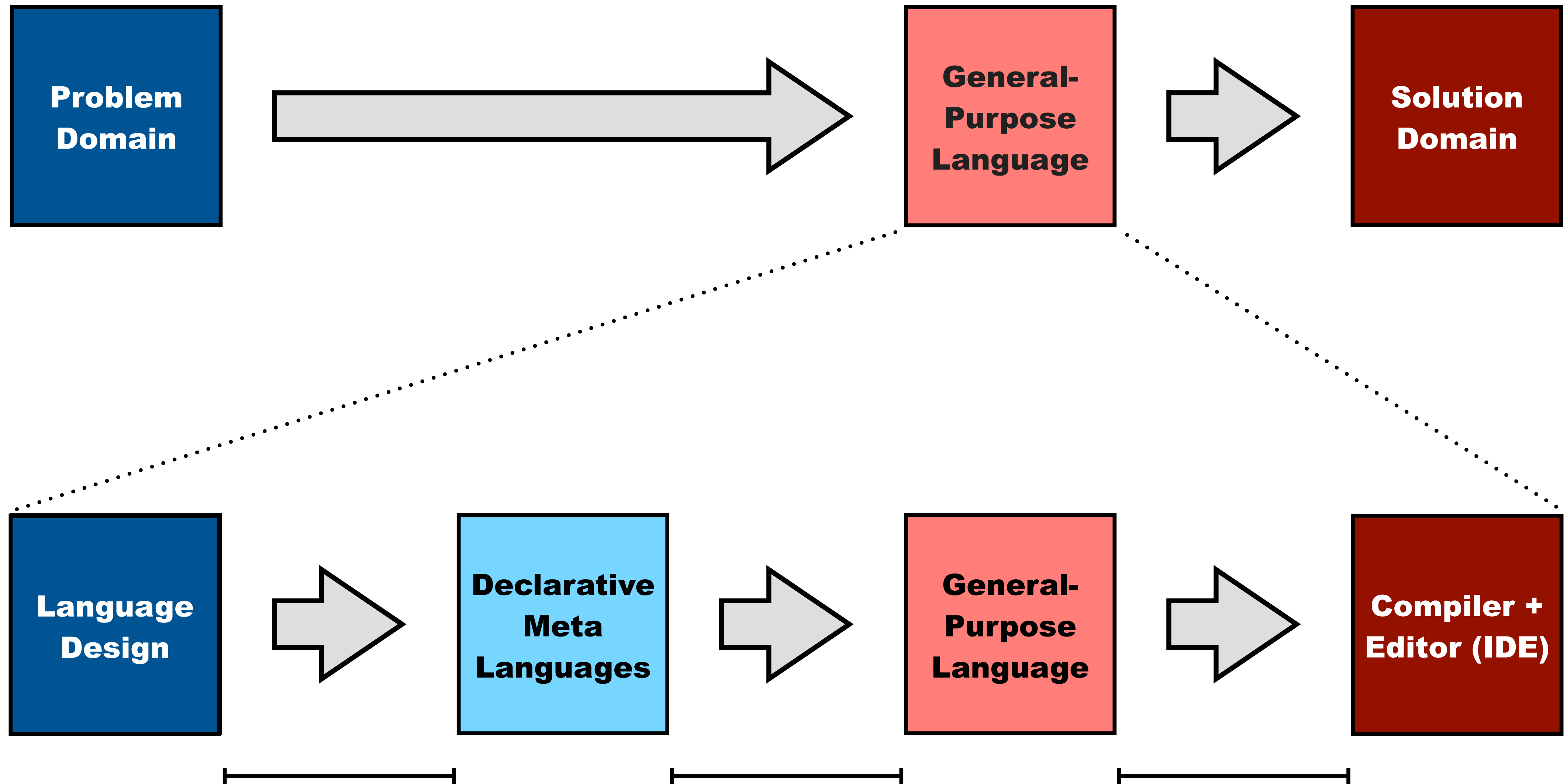
- Define the properties of a language
- Done by a language designer

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

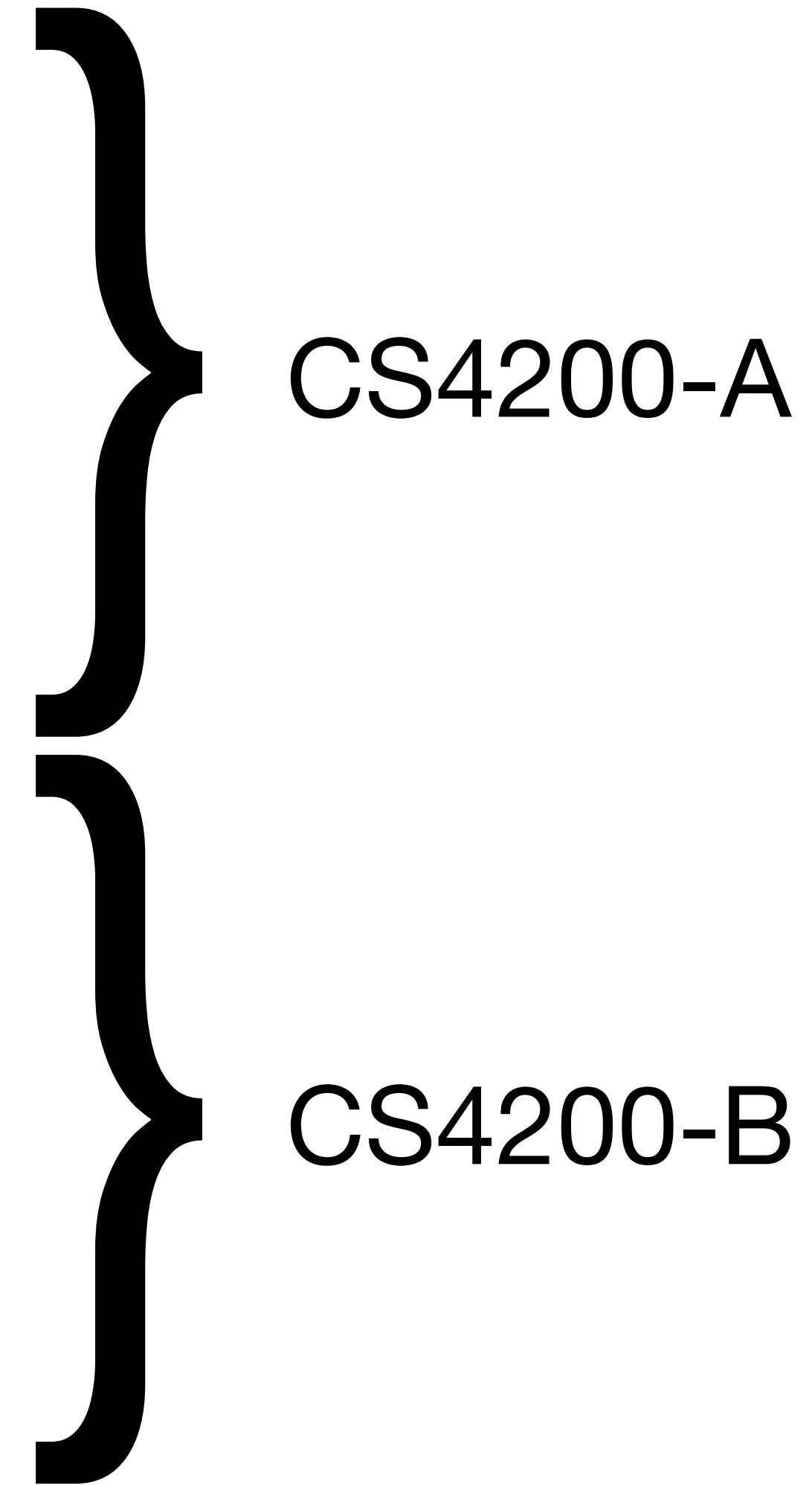
- term rewrite rules with programmable rewriting strategies
- derivation of program transformation system

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



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

- Spoofox 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 = OutputStamper.equals): O
    fun require(path: PPath, stamper: PathStamper = PathStamper.modified)
    fun generate(path: PPath, stamper: PathStamper = PathStamper.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))
    } }
```

Spoofax 3 = Spoofax/PIE

- An implementation of Spoofax with PIE as glue
- Get more live responses
- Under development => bugs! (apologies)

Compiler construction is a lot of fun ...

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

Domain-Specific Languages

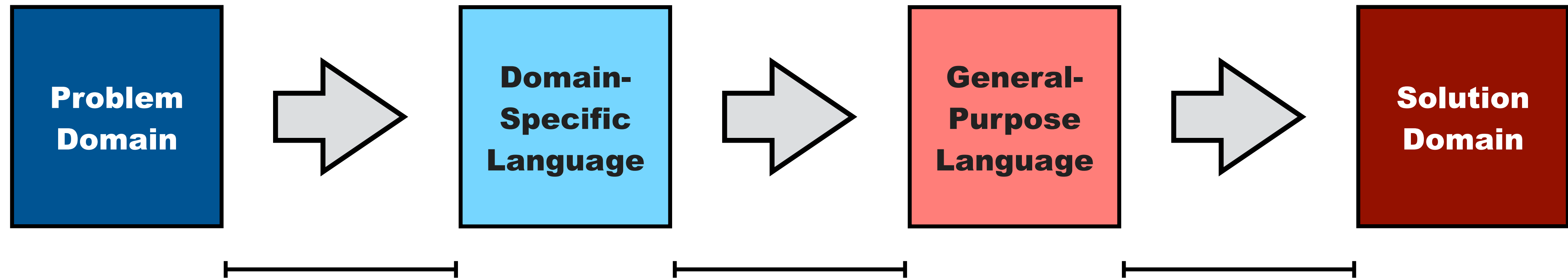
Traditional Compiler

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

CHICAGO
SPLASH2020

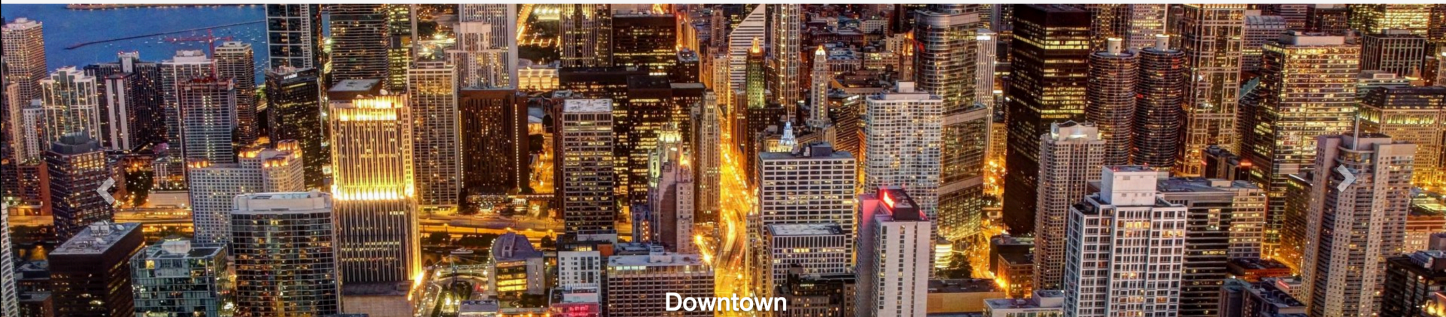
Sun 18 - Fri 20 November 2020 Chicago, Illinois, United States

CHICAGO
SPLASH2020

Write a Blog >>

Attending ▾ Tracks ▾ Organization ▾ Search Series ▾

Sign in Sign up



Downtown

Photo by Allen McGregor

SPLASH 2020

Welcome to the website of the SPLASH 2020 conference. We are working hard to fill the website with all related information. Please check back soon!

SPLASH 2020 - Chicago, USA

SPLASH is the ACM SIGPLAN conference on Systems, Programming, Languages, and Applications: Software for Humanity. SPLASH embraces all aspects of software construction and delivery, to make it the premier conference on the applications of programming languages – at the intersection of programming languages and software engineering. SPLASH 2020 will take place in Chicago from Sunday 15th to Friday 20th of November 2020. Early registration deadline for the conference will be Thursday 15th October AOE.

SPLASH includes the following co-located conferences: OOPSLA, Onward!, GPCE, SLE, DLS, and MPLR; as well as a large array of workshops and events.

The SPLASH-I will feature a number of speakers of interest to software practitioners and researchers alike.

SPLASH 2020 Tracks

OOPSLA | Onward! Essays | Onward! Papers | Rebase | Workshops

Show all tracks

Upcoming Important Dates

Wed 15 Apr 2020 ☉
OOPSLA Paper Submission

Wed 22 Apr 2020 ☉
SAS Abstract Submission

Fri 24 Apr 2020 ☉
SAS Paper Submission

Sat 13 - Tue 16 Jun 2020 ☉
SAS Author Response Period

Thu 11 - Tue 16 Jun 2020 ☉
OOPSLA Author Response

Wed 1 Jul 2020 ☉
OOPSLA Author Notification

EELCO VISSER / CSE2120 / 2017-2018 / CPL / LAB / ASSIGNMENT 4

Actions ▾ 0.0/10.0 1 2 3 + - ↩ ↪ Submission info

Func + Env ✖ Not completed yet

Submission Assignment Answer Statistics/Dates Submissions Edit Checklist Edit Assignment Manage Assignment Tree Discussions

Description Assignment Info

visible from: Fri, Jul 6, 2018 13:00

Summary:

- Implement the interpreter as specified by the [notes](#). You should consult [chapter 6](#) and [chapter 7](#) from the book to understand the concepts.
- Develop more tests to support your understanding of the expected behaviour of the interpreter.
- Do not change the signature of the functions given, or you'll run into compilation problems with specification tests.

Solution Test

You are viewing this submission in read-only mode. Deadline passed.

```
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   def parse(sexpr: SExpr): ExprExt = {
9     throw NotImplementedException()
10   }
11 }
12
13
14 object Desugar {
15   def desugar(c: ExprExt): ExprC = {
16     throw NotImplementedException()
17   }
18 }
19
20 object Interp {
21   def interp(c: ExprC): Value = interp(c, Nil)
22   def interp(c: ExprC, nv: Environment): Value = {
23     throw NotImplementedException()
24   }
25 }
26
27
28
```

Console Grading Discussion Revision History

Compile ▶ Your Test ▶ Spec-test

Status: None

Spec-tests ran: 0

researchr profile library explore calendar eg: dijkstra 1959 search

Inbox (1551) | Eelco Visser | Sign Off

All Publications

Unidentified publications

Copy publications

External Links

Homepage

DBLP

ACM

LinkedIn

Twitter

Google Scholar

Blog

ORCID

Profile Name

eelcovisser

User

Statistics

Publications: 201

Edited: 10

Credit: 33776

Aliases

Visser

E. Visser

Visser, Eelco

Visser, E.

Eelco Visser

Your Profile

About Publications Co-authors Theses Contributions Reviews Bibliographies Groups Edit

ABOUT EELCO VISSER

Eelco 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 [Spoofox 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 ([mob](#)). He is the lead developer of the [researchr](#) bibliography management system.

AFFILIATIONS

2006 – : Delft University of Technology

1998 – 2006 : Utrecht University

1997 – 1998 : Oregon Graduate Institute

1993 – 1997 : University of Amsterdam

RECENT PUBLICATIONS

A Research Agenda for Formal Methods in the Netherlands

Marieke Huisman, Wouter Swierstra, Eelco Visser.

Technical Report UU-CS-2019-004, 2019. [doi] [X]

Fast and Safe Linguistic Abstraction for the Masses

Eelco Visser.

In A Research Agenda for Formal Methods in the Netherlands. pages 10-11, July 2019. [X]

From Definitional Interpreter to Symbolic Executor

Mensing, Adrian D., Hendrik van Antwerpen, Casper Bach Poulsen, Eelco Visser.

In Proceedings of the 4th ACM SIGPLAN International Workshop on Meta-Programming Techniques and Reflection. 2019: 11-20 [doi] [X]

Precise, Efficient, and Expressive Incremental Build Scripts with PIE

Gabriel Konat, Roelof Sol, Sebastian Erdweg, Eelco Visser.

In Second Workshop on Incremental Computing (IC 2019). 2019: [X]

Towards language-parametric semantic editor services based on declarative type system specifications

mystudyplanning.tudelft.nl/scheduler/#/year?_k=2ywczz

My Study Planning

Overview Supervisor Student Overview Course Statistics Demo Track Select Demo Planning Demo Track Reqs Demo Add Additional Courses Demo Submit Demo Submissions History

Eelco Visser Sign out

EEMCS \ Computer Science

CS - Software Technology 2016

Search, ex algorithm ec:4 period:3

Common Core ST 2016

EC 48/53

Specialisation courses start first period 2016

EC 10/100

Specialisation courses start second period 2016

EC 23/27

Specialisation courses start third period 2016

EC 21/102

Specialisation courses start fourth period 2016

EC 10/60

Research Groups 2016

EC 63/327

Seminarvakken CS 2016

EC 5/88

Special Programmes 2016

EC 47/282

Common Core DST 2016

EC 27/48

Other Specialisations 2016

EC 5/20

EIT Master's Programme ICT Innovation 2016

EC 22/212

Total ec's: 73

Q1/Q5 EC 34

Q2 EC 28

Q3 EC 5

Q4 EC 5

IN4301 Advanced Algorithms

EC 5

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

CS4130 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 5

IN4073TU Embedded Real-Time Systems

EC 5

CS4015 Behaviour Change Support Systems

EC 5

evatool.tudelft.nl

EvaTool Administration People Committees Faculty ▾

Prof. Dr. E. Visser ▾

EvaTool - Education Evaluation

Organize the workflow of course and programme evaluation. Record course statistics. Collect feedback from students and instructors. Produce a printable report summarizing the evaluation of a quarter or semester.

[read more](#)

Your Dashboard

Educatons Your Courses

Filter by course name, code or period

Search by course code, name or period Search

Filter by category

Education period

Nothing to filter

Education level

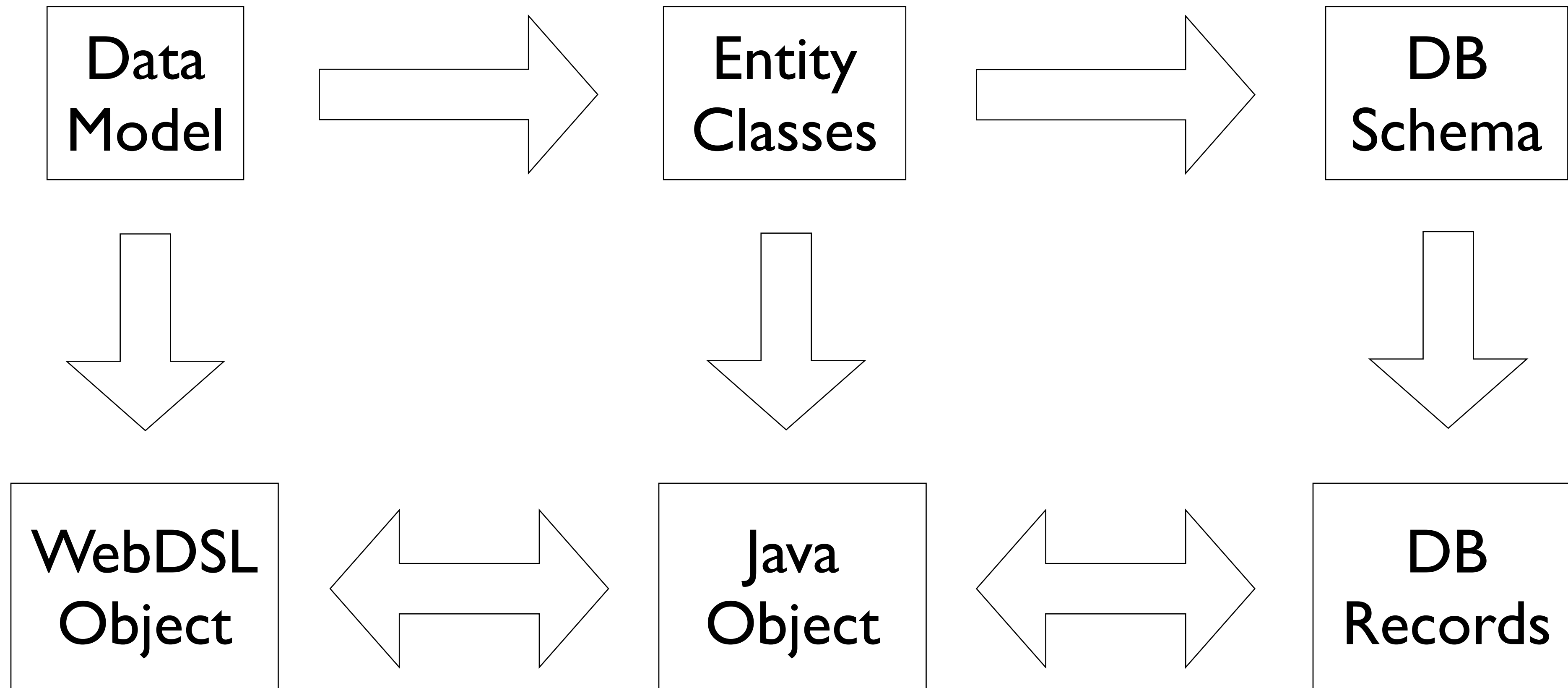
Nothing to filter

Education discipline

Nothing to filter

Results

WebDSL: Automatic Persistence



WebDSL: Entity Declarations

entity declaration

property

```
entity E {  
  prop :: ValueType  
  prop -> EntityType  
  prop <> EntityType  
  prop -> Set<EntityType>  
  prop -> List<EntityType>  
  prop -> EntityType (inverse=EntityType.prop)  
  function f(x : ArgType) : ReturnType {  
    statements;  
  }  
}
```

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)

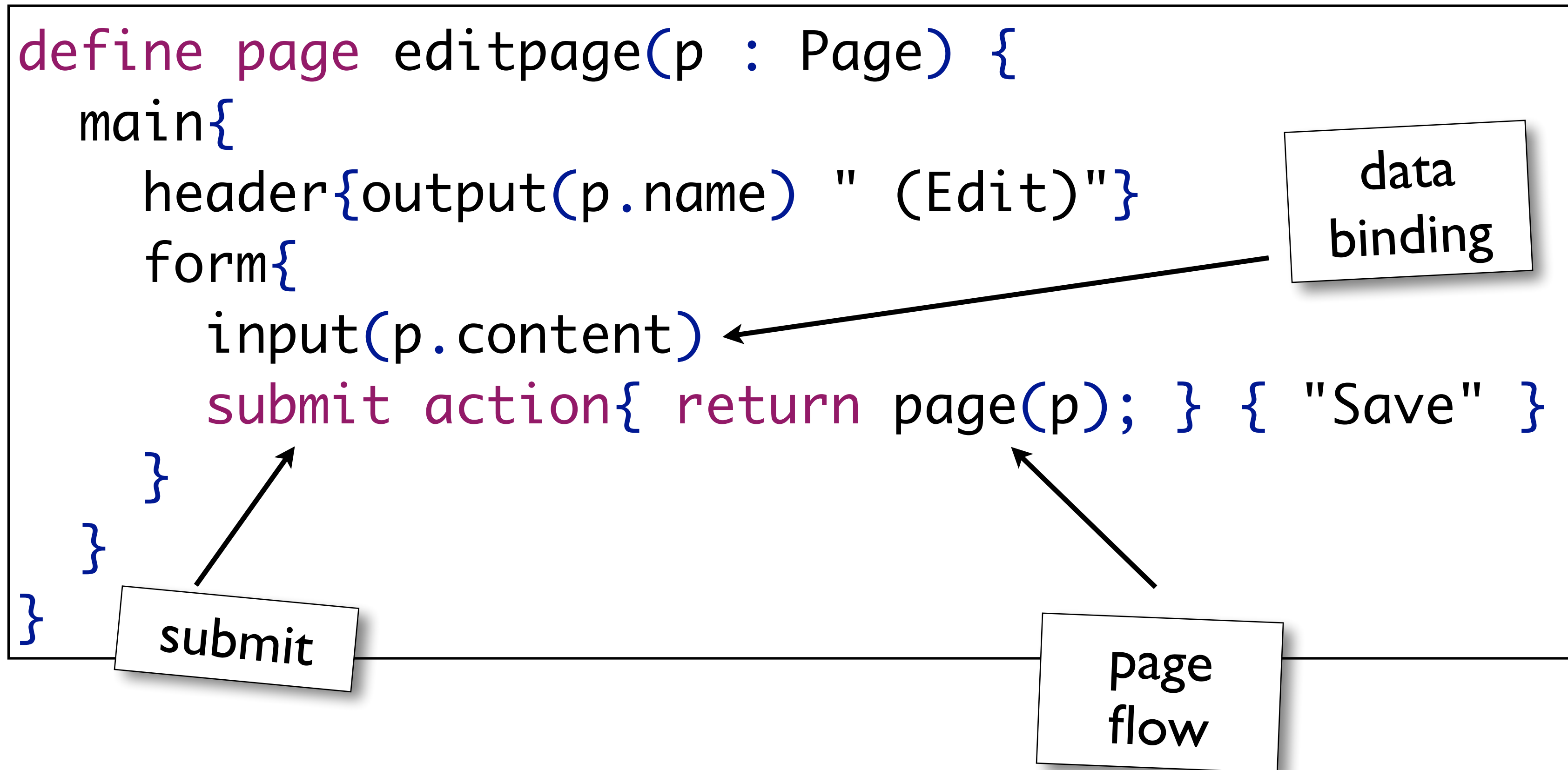
template definition

```
define main() {  
  includeCSS("wiki.css")  
  top()  
  block[class="content"] {  
    elements()  
  }  
}  
  
define span top() {  
  navigate root() {"Wiki"}  
}
```

template call

parameter

WebDSL: Forms



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

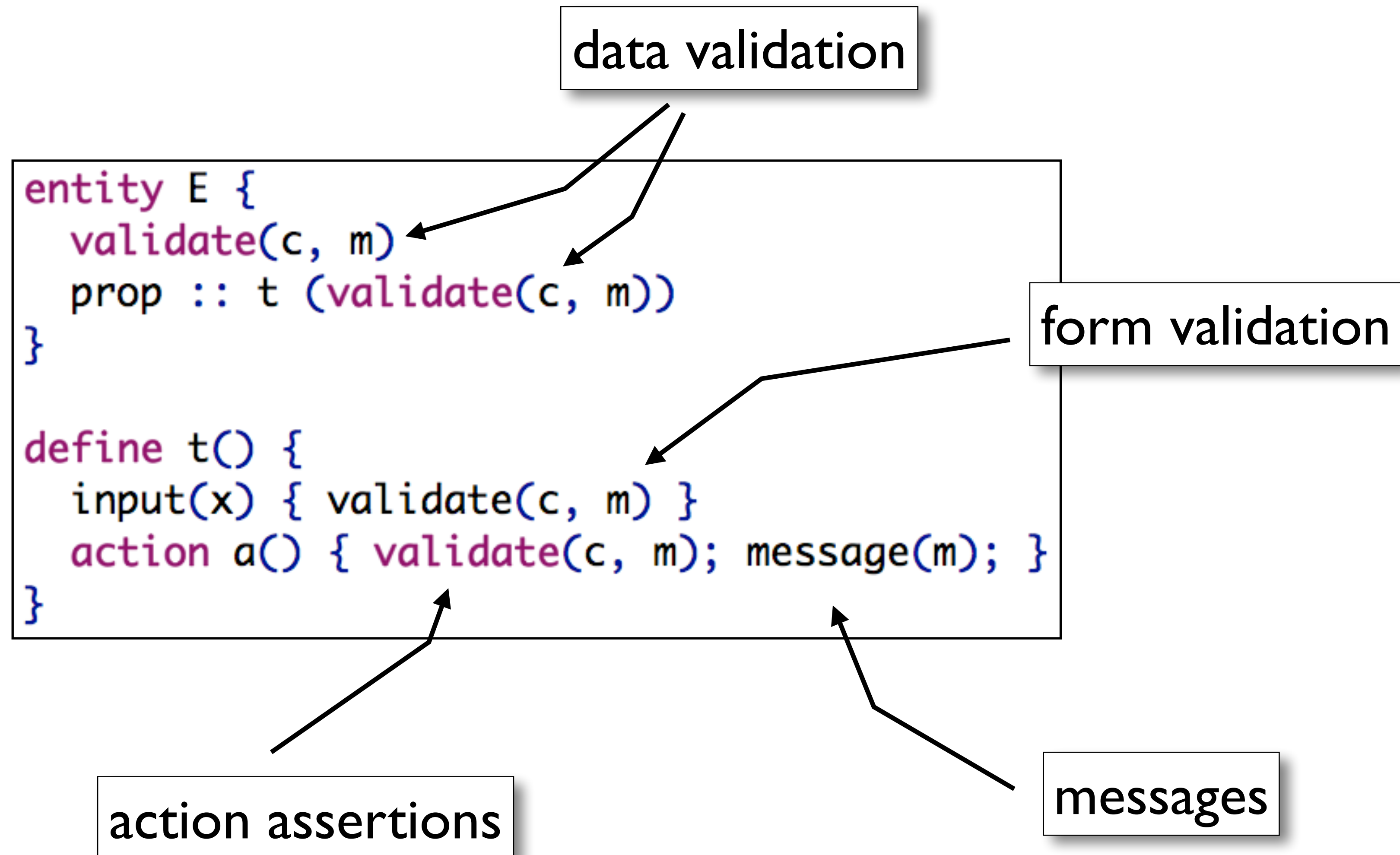
WebDSL: Search

search annotations

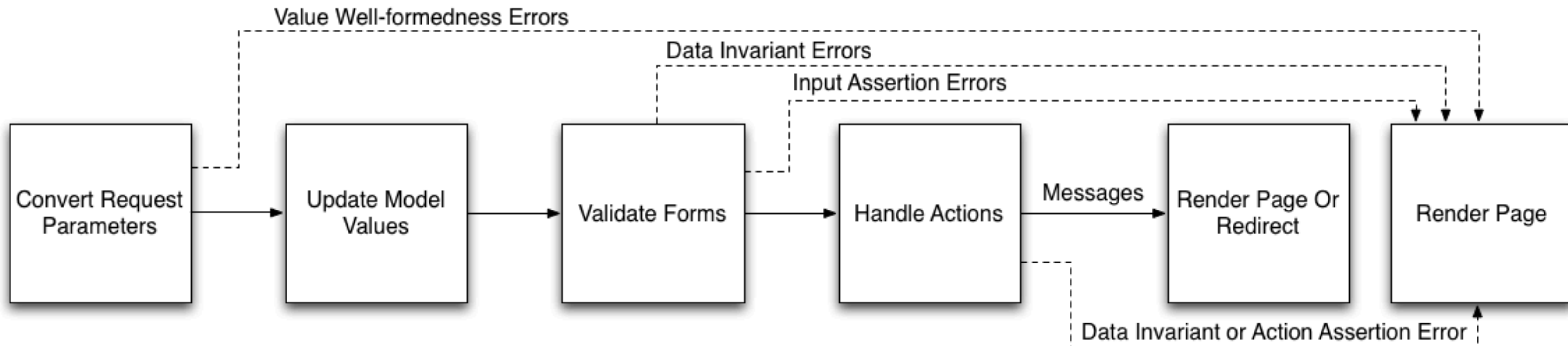
```
entity Page {  
  name      :: String (id,searchable)  
  content   :: WikiText (searchable)  
  modified  :: DateTime  
  authorSearch :: String (searchable) := authorNames()  
}  
  
define page search(query : String) {  
  var newQuery : String := query;  
  form {  
    input(newQuery)  
   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

search queries

WebDSL: Validation Rules



WebDSL: Data Validation Lifecycle



WebDSL: securityContext

```
entity User {  
  username :: String (id)  
  fullname :: String (name)  
  email    :: Email  
  password :: Secret  
}
```

representation of principal

access control rules

principal is User with credentials username, password

turn on access control

```
session securityContext {  
  principal -> User  
}
```


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’

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" }
    }}
  }
```

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

Syntax

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

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

Portable Editors

- Portable editor bindings based on AESI model (Pelsmaecker)
- 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?

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 Spoofax 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

After the Master

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

Concurrency

Category Theory

Wanted: Grammar Engineer

Goal

- A collection of high quality syntax definitions for key languages
- Spoofox 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, mentoring
- Administration

Combine with PL research

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

Exam

October 28: Exam

- 13:30-16:30

January 21: Resit

- 13:30-16:30

Topics

- Everything we studied in the lectures
- Example exam questions
 - ▶ homework assignments
 - ▶ exam from last year

Except where otherwise noted, this work is licensed under

