Domain Specific Language Design (2IMP20)

Loek Cleophas

Ivan Kurtev

Department of Mathematics and Computer Science — Software Engineering and Technology cluster

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Teachers and Teaching Assistant

Dr.ir. Loek Cleophas
(l.g.w.a.cleophas@tue.nl)
Assistant professor,
Software Engineering
and Technology
+ Research Fellow,
Stellenbosch University

Dr. Ivan Kurtev
(i.kurtev@tue.nl)
Parttime associate professor,
Software Engineering
and Technology

Catalin Pop BSc
(*contact via MS Teams*)

What about you?

- Which MSc?
- BSc here or elsewhere?
- Automata and Formal Languages course?
- Regexes?

Resources

- Syllabus / Home page on Canvas
- Books (see Canvas for links):
  - *Software Languages* by R. Lämmel
  - *Introduction to Compiler Design* by T. Mogensen
  - *The Epsilon Book* by D. Kolovos et.al.
  - *DSL Engineering* by M. Voelter
  - Papers
- Slides + recordings
- Tools + manuals: Eclipse (Rascal, Xtext, Epsilon, …)
- Assignments
  - MS Teams team for questions; see Canvas for link

Scheduling

- 2 x 2 hours lectures per week
  - All in Auditorium 9
  - All recorded, *not live streamed*

Legend
No lecture
Loek
Ivan
Guest lecture ?
Tentative

Wednesday (7th + 8th)	Friday (3rd + 4th)
April 26th	Bridge Day
May 3rd	Liberation Day
May 10th	May 12th
May 17th	May 19th
May 24th	Bridge Day
May 31st	June 2nd
June 7th	June 9th
June 14th ?	June 16th
June 21st ?	June 23rd ?

Learning objectives

1. define *regular grammars*;
2. use basic *scanner (generator)* technologies;

3. define *context-free grammars*;
4. use basic deterministic *parser (generator)* technologies;

5. compose *abstract syntax definitions* using *EMF technology*;
6. define a *concrete textual syntax* for a language based on an abstract syntax definition;

7. understand formalisms to define *static and dynamic semantics* of a language;

Learning objectives

8. define *language definitions* using *Rascal*;

9. understand and apply *model transformations* in order to develop model-to-model transformations;
10. understand and apply *code generators* in order to generate executable code;

11. use *formalisms to define (domain specific) languages*, the abstract and concrete syntax and the static and dynamic semantics;

12. develop *textual editors* for the languages using *language workbenches [LWBs]*;

13. design a domain specific language and create an *environment for editing and executing*.

Assignments

- 3 assignments in total
  - First assignment handed out on May 12th, hand in by May 29th
    - Working in pairs
  - Second assignment handed out on June 2nd, hand in by June 16th
    - Working in pairs
  - Final assignment handed out on June 16th, hand in by July 10th ***noon***
    - Partly working in pairs, see next slide

➔ Use Canvas to sign up to pair

Grading

Final grade

- First two assignments (60% of final grade—30%-points each)
- Final assignment (40% of final grade):
  - In pairs: Design and implement your own favorite DSL on your own favorite LWB
  - Individual: Write reflection report on the DSL design and LWB
- Both grades need to be at least 5.5

Course Outline

Grammarware part: basic generic language technologies — Loek

- Introduction to languages and DSLs
  - Rascal
- Design of domain specific languages
- Lexical and context-free grammars by means of grammarware approach
  - Xtext
- Semantics

Modelware part: applied generic language technologies — Ivan

- Model-ware approach to define DSLs
  - Meta-modeling
- Principle of model transformations
- Code generation

(Likely) guest lectures

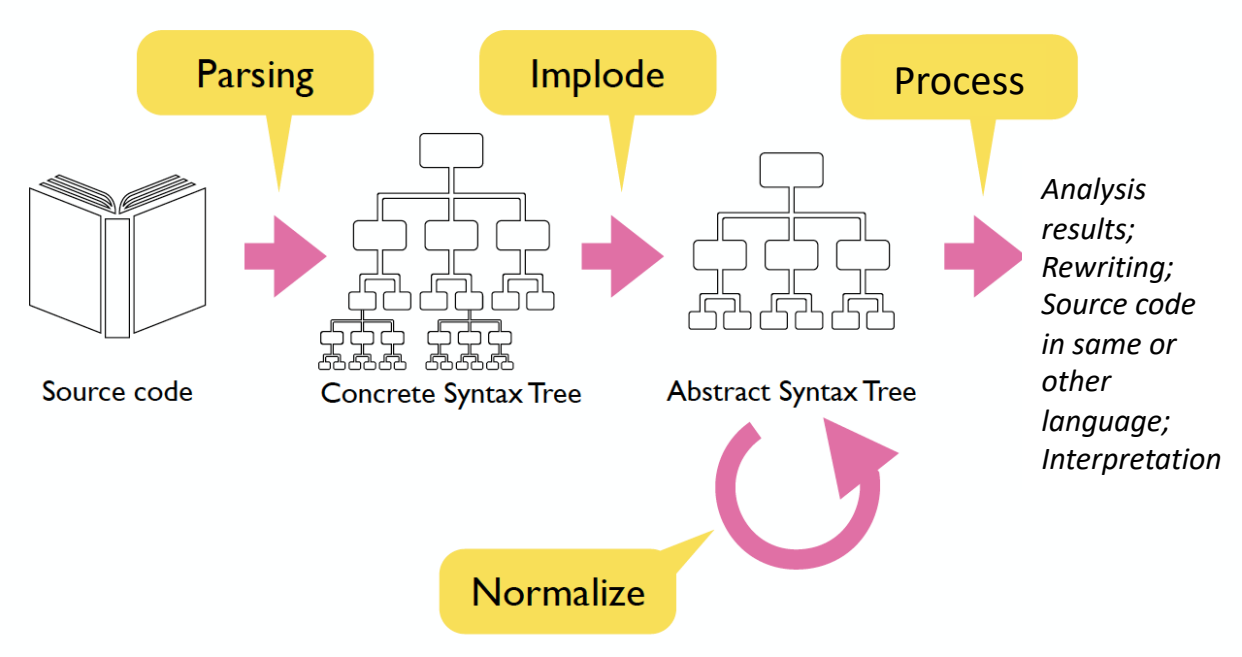
- DSL Design
- State of the art research in DSL Design and Implementation
- MPS tool
- DSLs at Canon Production Printing

Relation with other MSc courses

- *System Validation* by J.F. Groote
- *Program Verification Techniques* by A. Wijs
- *Software Evolution* by A. Serebrenik

Questions?

High-level: source code processing



Historical perspective on programming languages

First generation

- machine code languages

Second generation

- assembly languages

Third generation

- high-level languages: Fortran, Cobol, Algol, Pascal, C, etc.
  - control flow statements: conditionals, loops, etc.
  - recursion
  - increase of abstraction by means of *data abstraction*, *procedural abstraction*
- first generation of compilers and interpreters
  - formal language theory applied to scanners and parsers

Historical perspective on programming languages

Fourth generation

- data base management, report generation, GUI, web, etc.
- first domain specific languages (in some sense)

Fifth generation

- constraint-based languages, logic languages, declarative, functional: Prolog, Haskell, etc.
- More recently: scripting languages such as Python, Ruby, Rust, …

Sixth generation

- domain specific languages?

Characteristics of Modern Programming Languages

Long lifetime: C 50+ years, Java 25+ years

Often standardized: ANSI-C, C++ (ISO)

Languages inspire each other:

- Scala combines OO with functional features,
- Java incorporated lambda abstraction, etc.
- Python incorporating static types

Development and maintenance of PL is often centralized:

- Bell Labs (C), Microsoft (C#), Oracle (Java), Martin Odersky (Scala), etc.

Languages evolve slowly:

- development/maintenance of tooling (compilers/IDE) is immense effort and investment
- compiler components, such as scanners, parsers, etc., hand written
- modifications are disruptive because of large user communities, so evolution is gradual and well planned

TU/e

Characteristics of Domain Specific Languages

Observations:

- short(er) lifetime
- domain specific often even company specific, so small user communities
- rapid development and fast evolution → challenge
- investments in development and maintenance still substantial

Lead to different view on tooling to develop languages

- designed by CS experts for use by non-CS experts
- not (necessarily) universal ("Turing complete") or even executable
- generative approaches to develop tooling
  - components, such as scanners, parsers, etc., generated
- language workbenches for rapid prototyping

Domain Specific Languages

When do we consider a language domain specific?

- A DSL is a *formal, processable* language targeting a *specific domain or aspect* of a system
  - not a GPL, not necessarily Turing complete or executable, e.g. HTML

- Its semantics, flexibility and notation is designed in order to *support working with that domain or aspect as well as possible*

- *"A language that offers,* ***through appropriate notations and abstractions****,* ***expressive power focused on****, and usually restricted to,* ***a particular problem domain****"* — *Arie van Deursen, Paul Klint, Joost Visser: Domain-Specific Languages: An Annotated Bibliography. ACM SIGPLAN Notices 35(6): 26-36 (2000)*

Generic Language Technology for DSLs

- Goal: enable the easy creation of DSLs + language-specific tools
- Separate domain/language-specific aspects from generic aspects:
  1. Find good, reusable, solutions for generic aspects
  2. Find ways to define language-specific aspects
  3. Find ways to generate tools from language-specific definitions
- Integration in language workbenches

TU/e

Eating our own dog food

E.g. regexes, CFGs

- Problem description is domain specific and usually written in a DSL
- Abstracts from implementation
  - easier to switch to other implementation methods

- Program generators
  - contain generic algorithms and information about application domain
  - improvements/optimizations are good for all generated programs

Generic Language Technology for DSLs

- Program generators enable definition of (small) (domain specific) languages
- Language workbenches facilitate the development of DSLs, however *with a strong syntax perspective*
  - Rascal: http://www.rascal-mpl.org/ (*Used in this course*)
  - Eclipse with EMF + Xtext + model transformations (*Used in this course*)
  - MPS (*Guest lecture 2nd half of June?*)
  - Spoofax/IMP: http://strategoxt.org/Spoofax

Reading Material

- Chapter 1 of *Software Languages*
- Chapter 2 of *DSL Engineering*
- https://en.wikipedia.org/wiki/Programming_language_generations
- https://en.wikipedia.org/wiki/Timeline_of_programming_languages

Questions?