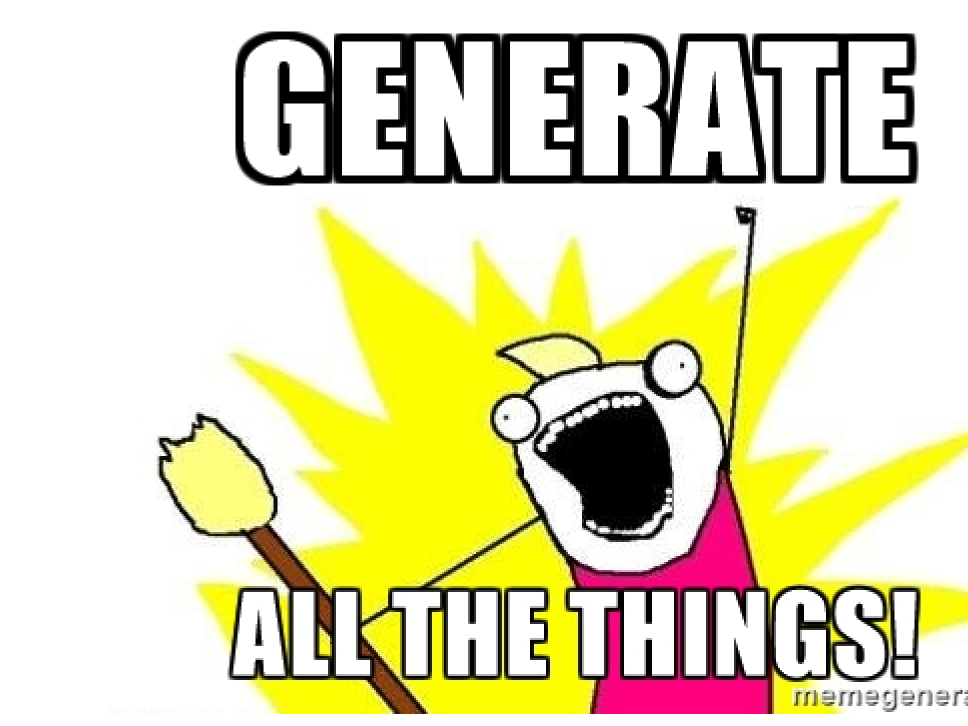
Multi-lingual code generation in Drasil

Jacques Carette, Spencer Smith, Dan Szymczak and Steven Palmer

McMaster University

WG 2.11, July 2017 Meeting


GENERATE
ALL THE THINGS!
memegenera

NO

Context

software certification

Context

software (re)certification

Context

software (re)certification

- All software artifacts as evidence:
  - requirements, software specification, software design, code, tests, “theory manual”, user manual, build mechanism, . . .

Context

software (re)certification

- All software artifacts as evidence:
  - requirements, software specification, software design, code, tests, “theory manual”, user manual, build mechanism, . . .
- Massive amounts of knowledge duplication
  - Implies that either
    - non-code artifacts do not get maintained well enough, OR
    - are felt to be an expensive nuisance
  - duplication harms traceability

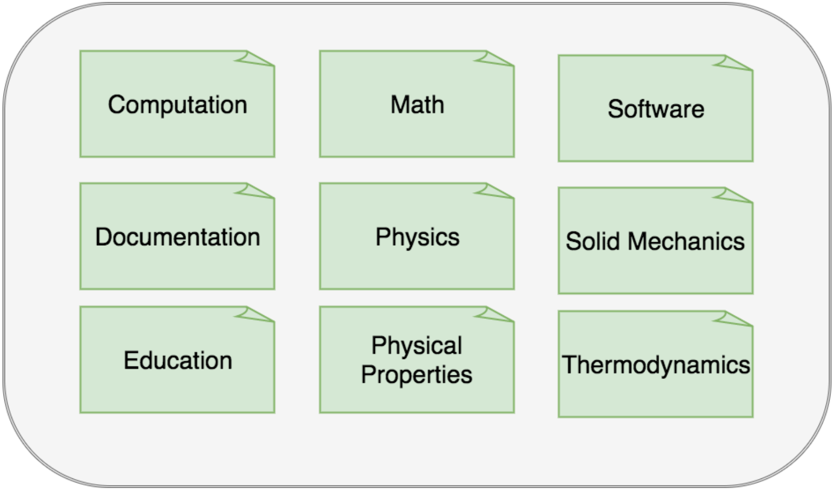
Engineering Software

Or, software that engineers write.

- GlassBR: Computer whether a given plate of glass will resist a blast force.
- GamePhysics: “Chipmunk” game physics engine.
- SSP: Computation of mixed-soil slope stability.
- SWHS: Solar Water Heating System (w/ phase change material).
- NoPCM: Solar Water Heating System without PCM.
- Tiny: convective and effective heat transfer coefficients.

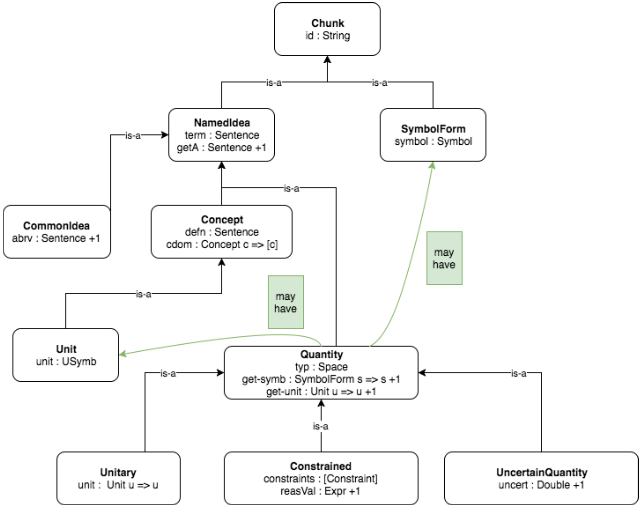
Ontologies

Data.Drasil



Ontologies

Data.Drasil - Language Concepts



Ontologies

Data.Drasil - Language Concepts - Document Language

Show the details already!

GOOL

Generic Object-Oriented Language.

History:

- Lucas Beyak, JC. SAGA: A DSL for Story Management. DSL 2011.
- Jason Costabile. GOOL: A Generic OO Language. M.Eng. 2012.
- Yuriy Toporovskyy. Used GOOL for assign. gen. for 2nd year CS (Java) course.

GOOL

Generic Object-Oriented Language.

History:

- Lucas Beyak, JC. SAGA: A DSL for Story Management. DSL 2011.
- Jason Costabile. GOOL: A Generic OO Language. M.Eng. 2012.
- Yuriy Toporovskyy. Used GOOL for assign. gen. for 2nd year CS (Java) course.

Currently covers

- Java, C#
- C++
- Python, Lua
- Objective-C
- GOOL

GOOL Design

Requirements

1. Capture the essence of programming in mainstream, pseudo-OO languages.
2. Make the embedded DSL palatable.
3. Eschew language-specific “idiomatic” patterns.
4. Target for generation.

GOOL Design

Requirements

1. Capture the essence of programming in mainstream, pseudo-OO languages.
2. Make the embedded DSL palatable.
3. Eschew language-specific “idiomatic” patterns.
4. Target for generation.

- multiple languages implies GOOL must capture

GOOL Design

Requirements

1. Capture the essence of programming in mainstream, pseudo-OO languages.
2. Make the embedded DSL palatable.
3. Eschew language-specific “idiomatic” patterns.
4. Target for generation.

- multiple languages implies GOOL must capture
    - features common to all targeted languages, and

GOOL Design

Requirements

1. Capture the essence of programming in mainstream, pseudo-OO languages.
2. Make the embedded DSL palatable.
3. Eschew language-specific “idiomatic” patterns.
4. Target for generation.

- multiple languages implies GOOL must capture
  - features common to all targeted languages, and
  - *language-required* features not common to all languages:
    - public/private scoping
    - dynamic/static binding
    - explicit destructors
    - explicit types in IO statements

GOOL Design

Requirements

1. Capture the essence of programming in mainstream, pseudo-OO languages.
2. Make the embedded DSL palatable.
3. Eschew language-specific “idiomatic” patterns.
4. Target for generation.

- multiple languages implies GOOL must capture
    - features common to all targeted languages, and
    - *language-required* features not common to all languages:
        - public/private scoping
        - dynamic/static binding
        - explicit destructors
        - explicit types in IO statements
- when rendering, extra information can be dropped

GOOL Design

Renderer

- uses record that acts as virtual dispatch table:

```
data Config = Config {
  assignDoc :: Assignment -> Doc,
  binOpDoc :: BinaryOp -> Doc,
  bodyDoc :: Body -> Doc,
  blockDoc :: Block -> Doc,
  callFuncParamList :: [Value] -> Doc,
  conditionalDoc :: Conditional -> Doc,
  declarationDoc :: Declaration -> Doc,
  exprDoc :: Expression -> Doc,
  funcDoc :: Function -> Doc,
  -- and many more
}
```

- one for each language (7)
- Text.PrettyPrint used to make rendered code look nice

GOOL Design

GOOL syntax

- designed to look like a programming language by using

GOOL Design

GOOL syntax

- designed to look like a programming language by using
  1. custom infix operators:

```
-- logical operators
(?!), (?<), (?<=), (?>), (?>=), -- etc.
-- arithmetic operators
(#~), (#/^), (#|), (#+), (#-), -- etc.
-- assignment operators
(&=), (&-=), (&++), (&~-) -- etc.
```

GOOL Design

GOOL syntax

- designed to look like a programming language by using
  1. custom infix operators:

```
-- logical operators
(?!), (?<), (?<=), (?>), (?>=), -- etc.
-- arithmetic operators
(#~), (#/^), (#|), (#+), (#-), -- etc.
-- assignment operators
(&=), (&-=), (&++), (&~-) -- etc.
```

  2. smart constructors:

```
bool, int, float, char, string,
true, false,
pubClass, privClass,
pubMethod, privMethod,
print, printLn
```

and so on (about 100).

GOOL Design

Consider the function $f(x, y) = 2x/y$. In GOOL:

```
f :: FunctionDecl
f = pubMethod (methodType float) "f"
  [param "x" float, param "y" float]
  (oneLiner $ return $
    (litFloat 2) #* (var "x") #/ (var "y"))
```

Implementation Choices

(Of generated code)

- programming language to be rendered?

Implementation Choices

(Of generated code)

- programming language to be rendered?
- generate library or program?

Implementation Choices

(Of generated code)

- programming language to be rendered?
- generate library or program?
- logging?
  - function calls
  - assignments

Implementation Choices

(Of generated code)

- programming language to be rendered?
- generate library or program?
- logging?
  - function calls
  - assignments
- documentation in code?
  - commented classes
  - commented functions
  - commented variables

Design Choices

- structure of input/output?
  - file vs console
  - how data is arranged

Design Choices

- structure of input/output?
  - file vs console
  - how data is arranged
- behaviour on failed constraints?
  - print warning
  - quit with error
  - throw exception

Design Choices

- structure of input/output?
  - file vs console
  - how data is arranged
- behaviour on failed constraints?
  - print warning
  - quit with error
  - throw exception
- which (common) algorithms to use?
  - sorting algorithm
  - searching algorithm
  - matrix multiplication, solving
  - ODE solver
  - etc.

Design Choices

- structure of input/output?
  - file vs console
  - how data is arranged
- behaviour on failed constraints?
  - print warning
  - quit with error
  - throw exception
- which (common) algorithms to use?
  - sorting algorithm
  - searching algorithm
  - matrix multiplication, solving
  - ODE solver
  - etc.
- Inline code or library calls?

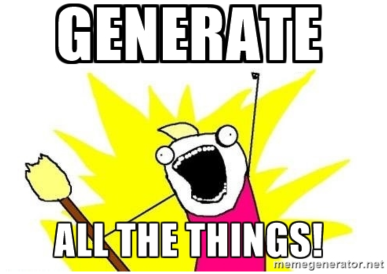
Open Questions

How to classify *choices*.

Open Questions

How to classify *choices*.

Proper Language(s) of choices.



NO