Groovy-Based DSL for cTAKES

Will Thompson

Center for Biomedical Research Informatics
NorthShore University HealthSystem

January 15, 2014

Motivation

- Make it easy to create pattern-based annotators
- Reduce need for boilerplate code ("uimaFIT++")
- Mix and match with other cTAKES/UIMA components
- Make common things easy, uncommon things possible
- Approach taken: Groovy-based internal DSL

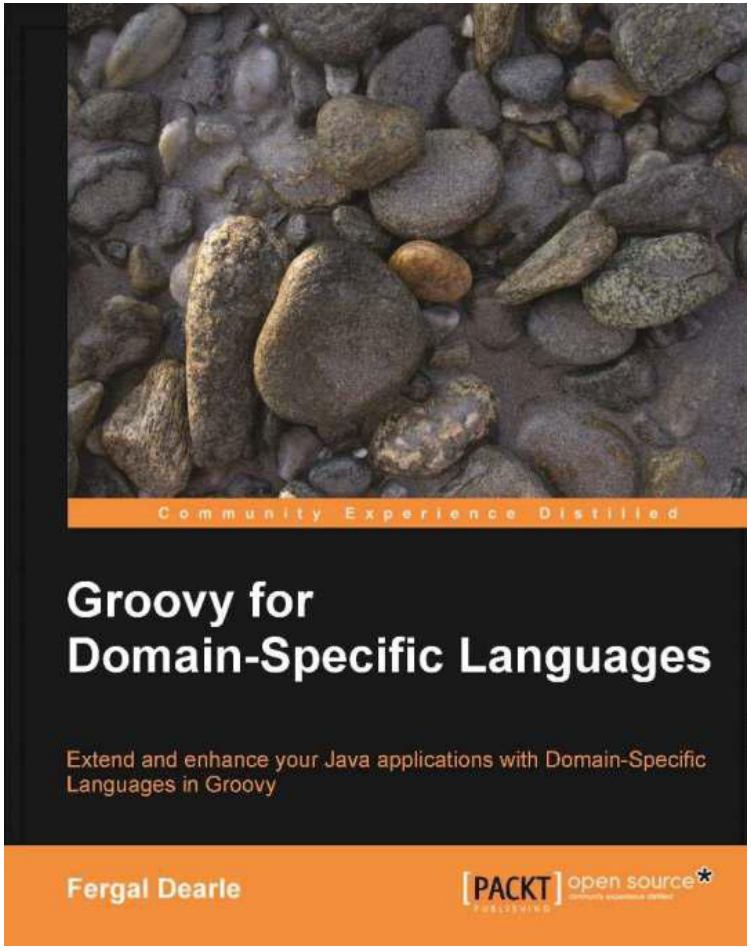
What is an Internal DSL?

Martin Fowler:

DSLs come in two main forms: external and internal. An external DSL is a language that's parsed independently of the host general purpose language: good examples include regular expressions and CSS. ... Internal DSLs are a particular form of API in a host general purpose language, often referred to as a fluent interface.

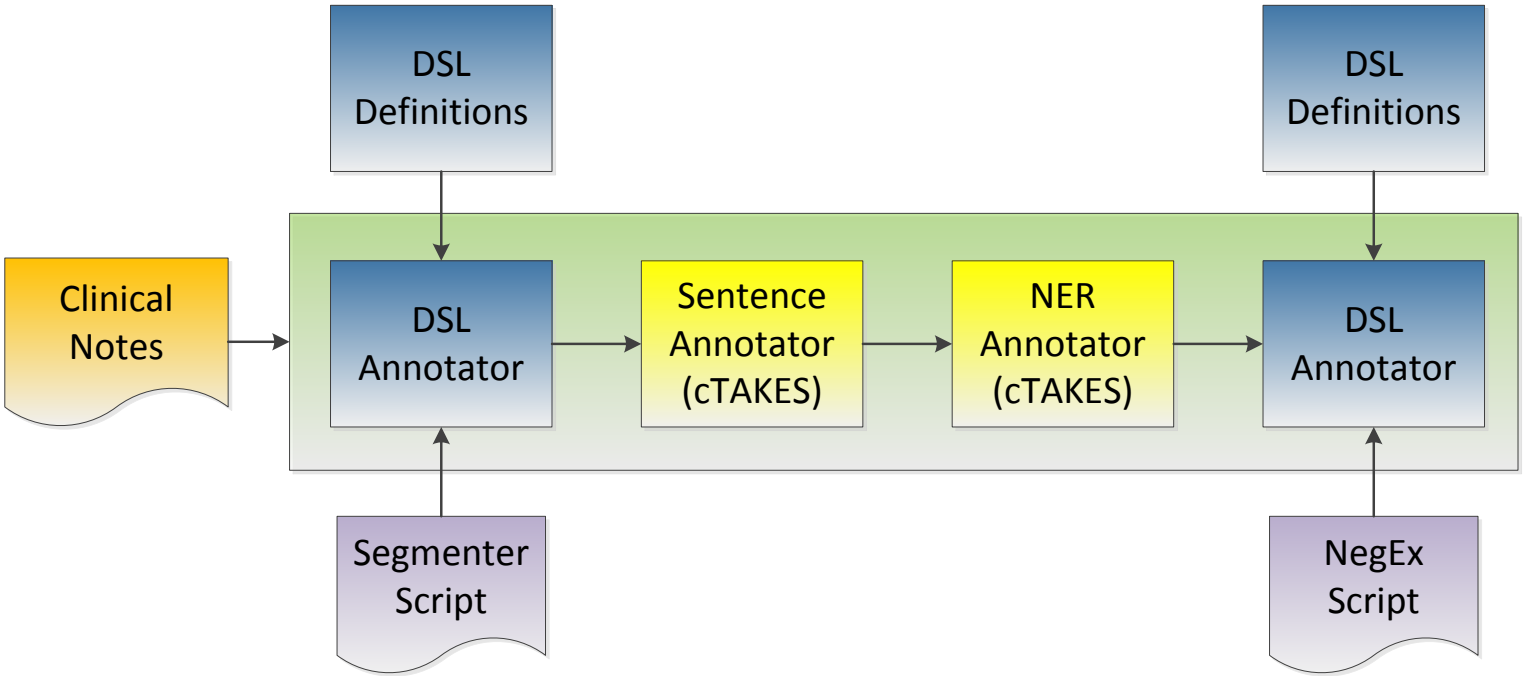
`http://martinfowler.com/books/dsl.html`

Groovy for DSLs



- Agile and dynamic language for the JVM
- Power features inspired by languages like Python, Ruby and Smalltalk
- Supports DSLs and other compact syntax
- Seamlessly integrates with all existing Java classes and libraries
- Compiles straight to Java bytecode so you can use it anywhere you can use Java

UIMA Pipeline



This is a Valid UIMA Analysis Engine

```
create
    type:Segment,
    begin:0,
    end:jcas.documentText.length()
```

```
AnalysisEngineFactory.createEngineDescription(
  GroovyAnnotator,
  GroovyAnnotator.PARAM_SCRIPT_FILE,
  "groovy/SimpleSegmenter.groovy")
```

1. Create a groovy script file
2. Instantiate a `GroovyAnnotator` instance and point at script

GroovyAnnotator

```
@Override
void initialize(UimaContext context) {
  super.initialize(context)
  config = new CompilerConfiguration()
  config.setScriptBaseClass(
    "org.northshore.cbri.UIMAUtil")
  shell = new GroovyShell(config)
  // load in script file contents
  this.script = shell.parse(scriptContents)
}
```

```
@Override
void process(JCas jcas) {
    UIMAUtil.setJCas(jcas)
    this.script.run()
}
```

DSL Functionality

3 types of functions

- Selecting annotations
- Creating annotations
- Matching text

Selecting Annotations (Simple Example)

```
// select all Sentences containing
// an EntityMention
sents = select
    type:Sentence,
    filter:contains(EntityMention)
```

- Optional typing and parentheses
- Class names denote class
- Named arguments (collected into a Map)
- Implemented internally as calls to `uimaFIT` methods

Selecting Annotations (Simple Example)

```
// select all Sentences containing
// an EntityMention
sents = select
    type:Sentence,
    filter:contains(EntityMention)
```

- Optional typing and parentheses
- Class names denote class
- Named arguments (collected into a Map)
- Implemented internally as calls to `uimaFIT` methods
- What's the point?

Selecting Annotations (Complex Example)

```
// select all Sentences contained in a findings
// Segment that end in "tubular adenoma." and
// do not contain an EntityMention
select(type:Segment).grep { seg ->
 seg.id == "FINDINGS"}.each {
  select(type:Sentence, filter:(
   and (coveredBy(seg),
    {it.coveredText==~/.+tubular\s+adenoma\./},
    not(contains(EntityMention))))
  }}
```

- Property syntax, collection closures
- Pre-defined and on-the-fly filters (closures)
- Compositional filters w. boolean functions
- First class support for regex strings & operators

Creating Annotations (Simple Example)

```
Segment seg = create
  type:Segment,
  begin:0,
  end:documentText.length(),
  id:'DEFAULT'
```

- Optional parentheses
- Named arguments (internally a Map instance)
- Annotation automatically added to CAS index
- JCas instance hidden (but accessible if needed)

Creating Annotations (Complex Example)

```
create(type:EntityMention,
  begin:0, end:10,
  polarity:1, uncertainty:0,
  ontologyConcepts:[
   create(type:UmlsConcept, cui:"C01"),
   create(type:UmlsConcept, cui:"C02")
  ]
)
```

- Embedded `create` call
- List literal
- Extension to `IdentifiedAnnotation` class to auto-convert `FSArray` to List

Extending Existing Classes

```
IdentifiedAnnotation.metaClass.
 getOntologyConcepts = {
  delegate.ontologyConceptArr == null ? []:
  select(delegate.ontologyConceptArr,
    OntologyConcept)
}
IdentifiedAnnotation.metaClass.
 setOntologyConcepts = { concepts ->
    array = new FSArray(jcas, concepts.size())
    int i = 0
    concepts.each {
        array.set(i, it)
        i += 1
    }
    delegate.ontologyConceptArr = array
}
```

Matching Annotations (Simple Example)

```
sents = select(type:Sentence)
patterns = [~/(?i)(tubular|villous)\s+adenoma/]

match(sents, patterns,
 { create(type:EntityMention,
    begin:it.start(1), end:it.end(1),
    polarity:1, uncertainty:0,
    ontologyConcepts:[
     create(type:UmlsConcept, cui:"C01")]
 )})
```

- All patterns applied to all annotations (text)
- Closure applied to every match
- Action taken can be anything (create annotation one possibility)

Matching Annotations (Complex Example)

```
pat = (~/(?s)(?<h1>@Head)(?=(?<h2>@Head)|\Z)/)
AnnotationMatcher matcher =
  pat.matcher(includeText:false)

matcher.each{ Map binding ->
  create(type:Segment,
    begin:binding.get("h1").begin,
    end:(binding.get("h2") ?
      binding.get("h2").begin
      : jcas.documentText.length()))}
```

- Regular expressions over annotations + text
- Returns a binding, a map from group names to matched annotations

Similar Projects

- GATE's Java Annotation Patterns Engine (JAPE)
- Apache Rule-based Text Annotation (RUTA)

Future Work

- Extend DSL (RUTA and JAPE functionality)
- Groovy DSL descriptor for Eclipse editor support
- Contribute to cTAKES sandbox
- Evaluation?