Comparative Study of DSL Tools

N K Vasudevan and L Tratt
School of Design, Engineering and Computing
Bournemouth University
What are Domain Specific Languages (DSLs)?
- Mini languages tailored for a specific domain
- Example – SQL

Advantages
- Level of abstraction
- Quick and effective programs
Literature

Traditional techniques
- LEX & YACC, ANTLR
- Advantages
  - Better representation
- Disadvantages –
  - High start-up costs
  - re-usability

Contemporary techniques
- Embedding approaches
- Heterogeneous
  - Transformation (e.g.: Stratego, Silver)
- Homogeneous
  - Compile-time meta-programming (e.g.: Template Haskell)
  - Pure embedded (e.g.: Ruby)
- Other approaches – IDE based (e.g.: MPS)
Payoff of DSL technology
Motivation

- Which approach is better?
  - Dimensions - to compare the suitability of approaches

Objective

- Comparative study of DSL tools
- Case study – state machine language

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approach</td>
<td>Lines of code</td>
</tr>
<tr>
<td>Guarantee</td>
<td>Aspects to learn</td>
</tr>
<tr>
<td>Reuse</td>
<td></td>
</tr>
<tr>
<td>Error reporting</td>
<td></td>
</tr>
</tbody>
</table>
Case Study

- Implementing the state machine language in different DSL approaches
  - States and transitions
  - Variables

```
coin [credit + 1 < 3] / credit = credit + 1

coin [credit + 1 == 3] / credit = 0
```

```
doorOpen/alarm = true
reset/alarm = false, credit = 0
```

```
doorClose
```

Locked

Violation

Unlocked
DSL tools

- DSL tools selected
  - Ruby
    - Homogeneous – Pure embedded
    - Syntax cannot be changed
  - Converge
    - Homogeneous - Compile-time meta-programming
    - Customised syntax
  - Stratego
    - Heterogeneous - Program transformation
    - Transformation between arbitrary languages
DSL Implementation - Ruby

- Ruby
  - Dynamically-typed, object-oriented GPL

- DSL implementation
  - Combination of features – code blocks, dynamic typing, evaluations, and flexible syntax
  - Code blocks are closures
    - encode domain specific information
  - Runtime meta-programming
    - Dynamic dispatch – ‘responds_to?’ and ‘method_missing’
DSL Implementation - Ruby

Code block

```ruby
transition "charging" do |t|
  t.from_state 'locked'
  t.to_state = 'locked'
  t.guard do |credit|
    if (credit + 1) < 3
      true
    end
  end
end
```

Block argument

```ruby
class Fsm
  def transition(name, &aBlock)
    transition = Transition_class.new(name)
    transition.load_block(&aBlock)
    @transitions.push(_transition)
    end
  ...
end
```
**Flexible syntax and ‘yield’**

```ruby
class Transition_class
  def from_state(from_state)
    @from_state = from_state
    End

  def to_state=(to_state)
    @to_state = to_state
    End

  def load_block
    yield self
    End

  ...
```
**DSL Implementation - Stratego**

- **Stratego/XT**
  - Framework – transformation between arbitrary languages
  - Transformation pipeline
  - Concrete Syntax Vs Abstract Syntax Tree (AST) terms
    - $[[x := e ]]$ rather than using nested AST Terms—Assign(Var(x),Expr(e))

```
<table>
<thead>
<tr>
<th>Parse table (DSL)</th>
<th>Parse table (Stratego + DSL + Java)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Program (DSL)</td>
<td>Parse (sglri)</td>
</tr>
</tbody>
</table>
```
DSL Implementation - Stratego

- **DSL**

  ```
  int : credit : 0
  state locked
  transition unlocking from locked to unlocked : coin [credit + 1 == 3 ]/ credit := 0
  ...
  ```

- **Term rewrite rules using concrete syntax**

  - **Rule_name**: source **AST pattern** \(\rightarrow\) target **AST pattern**

  ```
  var-dec :|[ srt1 : x_2 : i1 ]| -> |[ private t x_2 = i1; ]|
  where <builtin-java-type> srt1 => t

  var-init : |[ state x_s ]| -> |[ this.states.add("~x_s"); ]|

  guard-init :|[ transition x_t from x_a to x_b : x_e ttail1 ]|->
  |[ if (...) { bstm_1 } ]|| where <trans-tail> ttail1 => bstm_1

  trans-tail :|[ guard1 ]| -> |[ if ( e_1 ) { _guard = true; ...} ]|
  where <guard> guard1 => e_1
  ```
DSL Implementation - Converge

- Converge
  - Dynamically typed
  - Syntax rich modern language
  - Unifies concepts from Python, Template Haskell

- Compile-time meta-programming
  - Macros
  - Splicing annotation ($<...>$)
  - Quasi-Quotes ([|...|])
  - Insertion ($${...}$$)
DSL Implementation - Converge

- **DSL block**
  - Variant of splice syntax ( \( \langle\langle expr\rangle\rangle \) )
  - DSL block translated to Converge AST at compile-time

```plaintext
TurnstileFSM := $<<FSM_Translator::mk_itree>>:

... alarm := 0
state locked
transition unlocking from locked to unlocked : coin [ credit + 1 == 3 ] / credit := 0
...

func main():
    turnstile := TurnstileFSM.new()
    turnstile.event("coin")

- **DSL implementation function**

```plaintext
func mk_itree(dsl_block, src_infos):
    parse_tree := parse(dsl_block, src_infos)
    return _Translator.new().generate(parse_tree)
```
Converge - src info

event --> "coin"

Event coin causes transition from locked to state locked
before :: Dict("credit" : 0, "alarm" : 0)
state changed >>> locked
after :: Dict("credit" : 1, "alarm" : 0)

event --> "coin"

Event coin causes transition from locked to state locked
before :: Dict("credit" : 1, "alarm" : 0)
state changed >>> locked
after :: Dict("credit" : 2, "alarm" : 0)

event --> "coin"

Traceback (most recent call at bottom):
1: File "/Users/nvasudevan/CodeSpace/converge/fsm/v7-final/runfsm.cw", line 22, column 1, length 23
   turnstile.event("coin")
   if not self.transition(e):
3: File "/Users/nvasudevan/CodeSpace/converge/fsm/v7-final/FSM_Translator.cw", line 159, column 67, length 14
   if tn.from -- self.state & tn.event -- e & tn.guard(self):
4: File "/Users/nvasudevan/CodeSpace/converge/fsm/v7-final/FSM_Translator.cw", line 203, column 48, length 18
   return [top.src_infos| $c(lhs) == $c(rhs) ]
   File "/Users/nvasudevan/CodeSpace/converge/fsm/v7-final/runfsm.cw", line 12, column 69, length 2
   transition unlocking from locked to unlocked : coin [ credit + 1 == "3" ] / credit := 0
   File "/Users/nvasudevan/CodeSpace/converge/fsm/v7-final/FSM_Translator.cw", line 238, column 23, length 9
   return $c(guard)
   File "/Users/nvasudevan/CodeSpace/converge/fsm/v7-final/FSM_Translator.cw", line 286, column 159, length 9
   return [(node.src_infos| Transition.new($c(CE::istring(node[1].value))), $c(CE::istring(node[3].value)), $c(CE::istring(node[5].value))), $c(event), $c(guard), $c(action))]
   File "/Users/nvasudevan/CodeSpace/converge/fsm/v7-final/FSM_Translator.cw", line 183, column 31, length 19
   transitions := $c(CE::iList(tns))
5: (internal), in Int.
Type Exception: Expected arg 2 to be conformant to Number but got Instance of String.
~/CodeSpace/converge/fsm/v7-final$
## Comparison

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Ruby</th>
<th>Stratego</th>
<th>Converge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approach</td>
<td>Lambda abstractions</td>
<td>Term rewriting</td>
<td>Compile-time meta-programming</td>
</tr>
<tr>
<td>Guarantee</td>
<td>Syntax valid (runtime)</td>
<td>No</td>
<td>Well-typed (compile-time)</td>
</tr>
<tr>
<td>Reuse</td>
<td>Limited</td>
<td>SDF grammar</td>
<td>Limited</td>
</tr>
<tr>
<td>Error reporting</td>
<td>Yes (runtime)</td>
<td>Limited (end language)</td>
<td>Compile-time</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Ruby</th>
<th>Stratego</th>
<th>Converge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code (Grammar; transformation; and DSL program)</td>
<td>n/a, 89, 55</td>
<td>79, 95, 12</td>
<td>36, 173, 11</td>
</tr>
<tr>
<td>Aspects to learn</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
Discussion

- **Ruby**
  - DSL implementation using host language features
  - Documentation – extensive resources on the web

- **Drawbacks**
  - DSL programs not so succinct
  - Syntax can not be extended
Discussion

- **Stratego**
  - Transformation between arbitrary languages
  - Context-sensitive transformation – dynamic rewrite rules

- **Drawbacks**
  - Ambiguities
  - Hudak’s argument – cost vs. Benefits
Discussion

- Converge
  - Systematic approach
  - Src info
    - Unique to Converge
    - Error reporting in terms of source DSL

- Drawbacks
  - Limited user base
  - Integrated DSLs are aesthetically jarring
Conclusion

- Implemented three different embedded approaches
  - Ruby
  - Stratego
  - Converge
- First study
- Limitations – future work
- Merits and demerits
- Guideline for future implementations
- Source code: http://navkrish.net