Comparative Study of DSL Tools

N K Vasudevan and L Tratt
School of Design, Engineering and Computing
Bournemouth University

Literature

- What are Domain Specific Languages (DSLs)?
  - Mini languages tailored for a specific domain
  - Example – SQL
- Advantages
    - Level of abstraction
    - Quick and effective programs

Literature

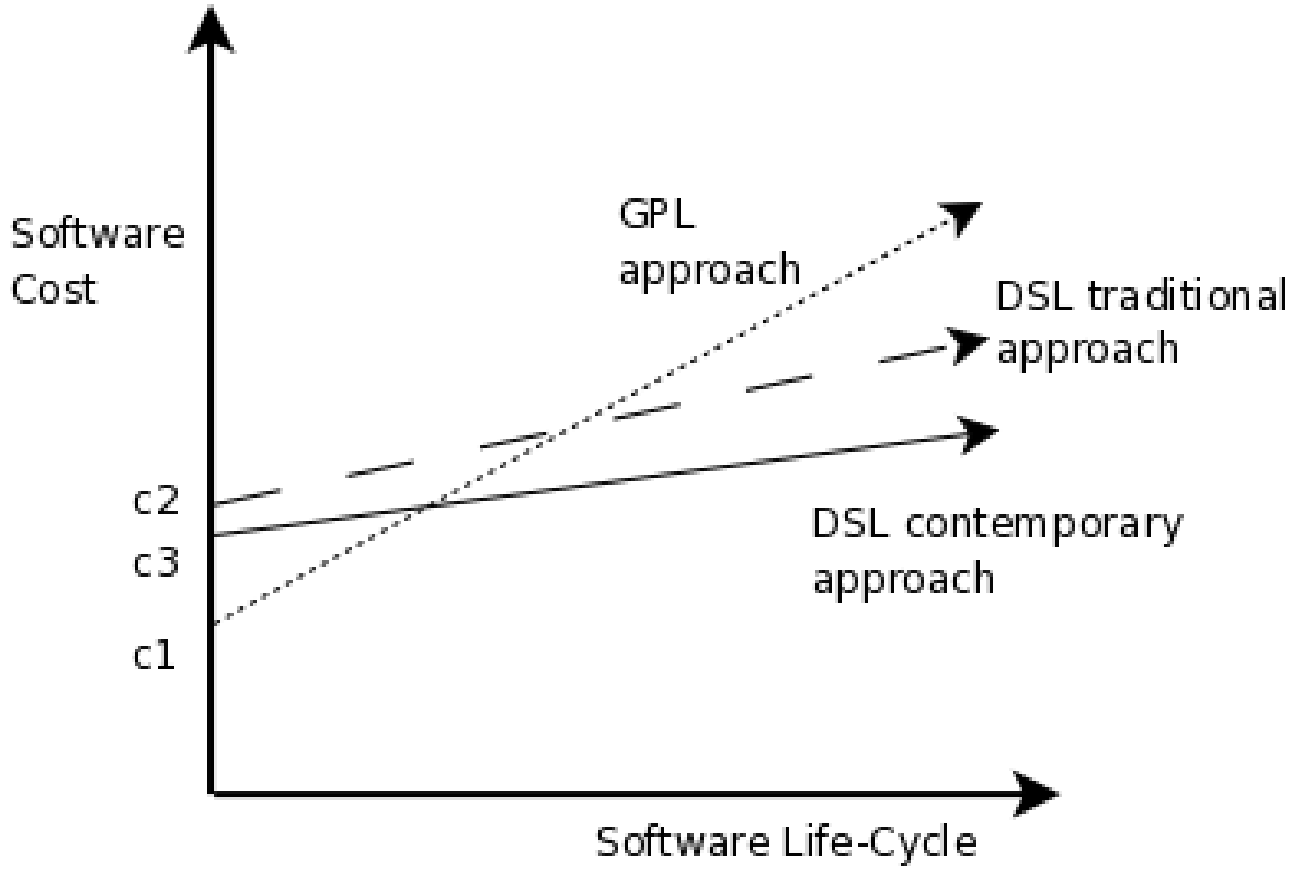
Traditional techniques

- 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



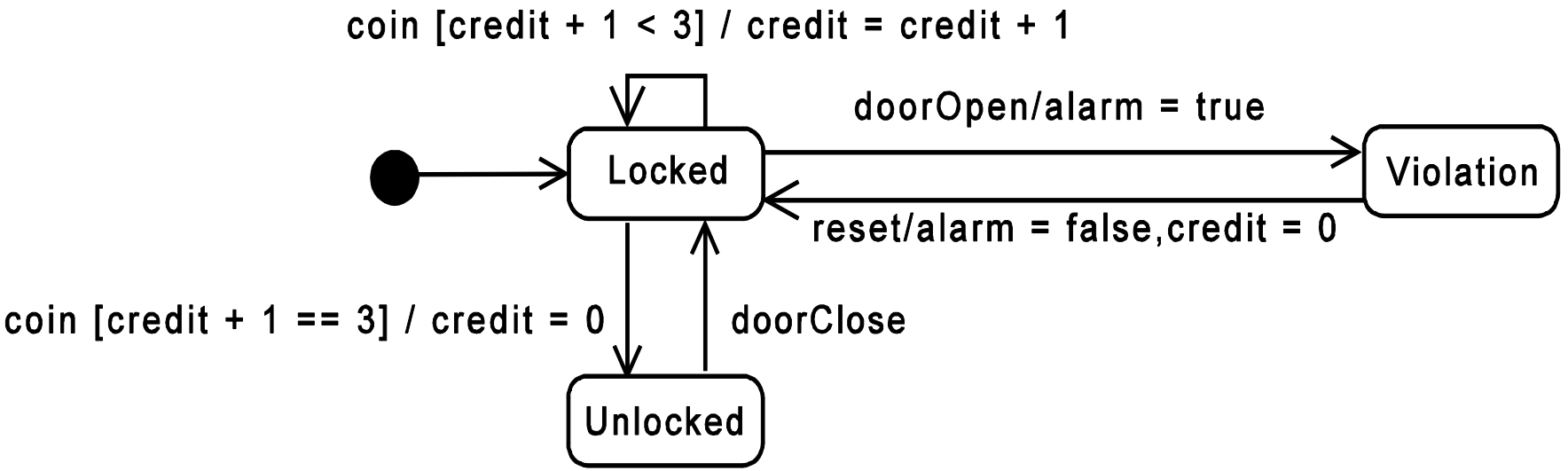
Motivation

- Which approach is better?
  - Dimensions - to compare the suitability of approaches
- Objective
  - Comparative study of DSL tools
  - Case study – state machine language

Dimensions	Metrics
▪ Approach ▪ Guarantee ▪ Reuse ▪ Error reporting	▪ Lines of code ▪ Aspects to learn

Case Study

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



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

```
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

```
class Fsm
  def transition(name, &aBlock)
    transition = Transition_class.new(name)
    transition.load_block(&aBlock)
    @transitions.push(_transition)
  end
  ...
```

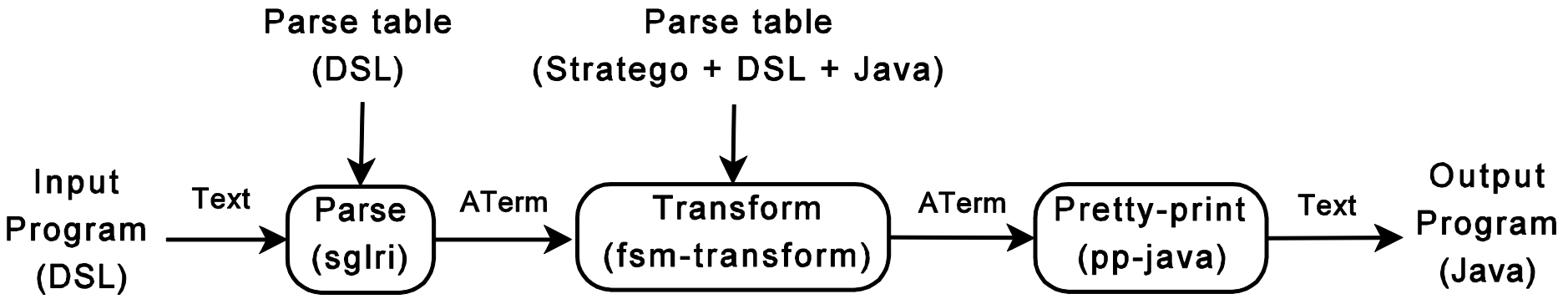
DSL Implementation - Ruby

- Flexible syntax and 'yield'

```
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))



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* ⟶ *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 ($<<*expr*>>)
  - DSL block translated to Converge AST at compile-time

```
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

```
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.cv", line 22, column 4, length 23
       turnstile.event("coin")
  2: File "/Users/nvasudevan/CodeSpace/converge/fsm/v7-final/FSM_Translator.cv", line 148, column 31, length 18
       if not self.transition(e):
  3: File "/Users/nvasudevan/CodeSpace/converge/fsm/v7-final/FSM_Translator.cv", 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.cv", line 293, column 40, length 18
       return [<op.src_infos>| ${lhs} == ${rhs} |]
     File "/Users/nvasudevan/CodeSpace/converge/fsm/v7-final/runfsm.cv", 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.cv", line 238, column 23, length 9
       return ${guard}
     File "/Users/nvasudevan/CodeSpace/converge/fsm/v7-final/FSM_Translator.cv", line 206, column 159, length 9
       return [<node.src_infos>| Transition.new(${CEI::istring(node[1].value)}, ${CEI::istring(node[3].value)}, $
c{CEI::istring(node[5].value)}, ${event}, ${guard},  ${action}) |]
     File "/Users/nvasudevan/CodeSpace/converge/fsm/v7-final/FSM_Translator.cv", line 103, column 31, length 19
       transitions := ${CEI::ilist(tns)}
  5: (internal), in Int.==
Type_Exception: Expected arg 2 to be conformant to Number but got instance of String.
~/CodeSpace/converge/fsm/v7-final$
```

Comparison

Dimension	Ruby	Stratego	Converge
Approach	Lambda abstractions	Term rewriting	Compile-time meta-programming
Guarantee	Syntax valid (runtime)	No	Well-typed (compile-time)
Reuse	Limited	SDF grammar	Limited
Error reporting	Yes (runtime)	Limited (end language)	Compile-time

Metric	Ruby	Stratego	Converge
Lines of code (Gram-Mar; transformation; and DSL program)	n/a, 89, 55	79, 95, 12	36, 173, 11
Aspects to learn	1	4	2

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