Language Engineering with Language Workbenches

Markus Voelter
independent/itemis
voelter@acm.org

www.voelter.de
voelterblog.blogspot.de
@markusvoelter
+Markus Voelter
A DSL is a focussed, processable language for describing a specific concern when building a system in a specific domain. The abstractions and notations used are natural/suitable for the stakeholders who specify that particular concern.
Concepts (abstract syntax) (concrete) Syntax semantics (generators) Tools and IDE
Shorter Programs
More Accessible Semantics
For a limited Domain!

Domain Knowledge encapsulated in language
General Purpose

C

Components

State Machines

Sensor Access

Domain Specific

LEGO Robot Control
<table>
<thead>
<tr>
<th></th>
<th><strong>more in GPLs</strong></th>
<th><strong>more in DSL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domain Size</strong></td>
<td>large and complex</td>
<td>smaller and well-defined</td>
</tr>
<tr>
<td><strong>Designed by</strong></td>
<td>guru or committee</td>
<td>a few engineers and domain experts</td>
</tr>
<tr>
<td><strong>Language Size</strong></td>
<td>large</td>
<td>small</td>
</tr>
<tr>
<td><strong>Turing-completeness</strong></td>
<td>almost always</td>
<td>often not</td>
</tr>
<tr>
<td><strong>User Community</strong></td>
<td>large, anonymous and widespread</td>
<td>small, accessible and local</td>
</tr>
<tr>
<td><strong>In-language abstraction</strong></td>
<td>sophisticated</td>
<td>limited</td>
</tr>
<tr>
<td><strong>Lifespan</strong></td>
<td>years to decades</td>
<td>months to years (driven by context)</td>
</tr>
<tr>
<td><strong>Evolution</strong></td>
<td>slow, often standardized</td>
<td>fast-paced</td>
</tr>
<tr>
<td><strong>Incompatible Changes</strong></td>
<td>almost impossible</td>
<td>feasible</td>
</tr>
</tbody>
</table>
Big Language

with many first class concepts!
Small Language

with a few, orthogonal and powerful concepts
Modular Language

with many optional, composable modules
2 Examples
appliance KIR {

    compressor compartment cc {
        static compressor c1
        fan ccfan
    }

    ambient tempsensor at

    cooling compartment RC {
        light rclight
        superCoolingMode
door rcdoor
        fan rcfan
        fan rcfan
        evaporator tempsensor rceva
    }

}
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
  entry { state noCooling }

state noCooling:
  check ( (RC->needsCooling) && (cc.c1->stehzeit > 333) ) {
    state rccooling
  }
  on isDown ( RC.rcdoor->open ) {
    set RC.rcfan->active = true
    set RC.rclight->active = false
    perform rcfanabschalttask after 10 {
      set RC.rcfan->active = false
    }
  }

state rccooling:
  entry { set RC.rcfan->active = true }
  check ( !(RC->needsCooling) ) {
    state noCooling
  }
  on isDown ( RC.rcdoor->open ) {
    set RC.rcfan->active = true
    set RC.rclight->active = false
    set tuerNachLaufSchwelle = currStep + 30
  }
  exit {
    perform rcfanabschalttask after max( 5, tuerNachLaufSchwelle-currStep ) {
      set RC.rcfan->active = false
    }
  }

Example
Refrigerators
Refrigerators

Example

```prolog
prolog {
    set RC->accumulatedRuntime = 80
}

step 10
assert-currentstate-is noCooling

mock: set RC->accumulatedRuntime = 110
step

mock: set RC.rceva->evaTemp = 10
assert-currentstate-is abtauEnd
assert-value cc.c1->active is false
perform rcfanabschalttask after 10 {
    set RC.rcfan->active = false
}

mock: set RC->accumulatedRuntime = 0
step 15
assert-currentstate-is abtauEnd
assert-value cc.c1->active is false

state noCooling:
    entry { state noCooling }
    check ( (RC->needsCooling) && (cc.c1->steht) )
        state rccooling
    on isDown (RC.rceva->open) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }

state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling) )
        state noCooling
    on isDown (RC.rceva->open) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachLaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachLaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
}
module CounterExample from countered imports nothing {

var int theI;

var boolean theB;

var boolean hasBeenReset;

statemachine Counter {
  in start() <no binding>
    step(int[0..10] size) <no binding>
    out someEvent(int[0..100] x, boolean b) <no binding>
    reseted() <no binding>
  vars int[0..10] currentVal = 0
    int[0..100] LIMIT = 10
  states (initial = initialState)
    state initialState {
      on start [ ] -> countState { send someEvent(100, true && false || true); }
    }
    state countState {
      on step [currentVal + size > LIMIT] -> initialState { send reseted(); }
      on step [currentVal + size <= LIMIT] -> countState { currentVal = currentVal + size; }
      on start [ ] -> initialState { }
    }
  }
} end statemachine

var Counter c1;

exported test case test1 {
  initsm(c1);
  assert(0) isInState<c1, initialState>;
  trigger(c1, start);
  assert(1) isInState<c1, countState>;
} test1(test case)
module CounterExample from counterd imports nothing {

  var int theI;
  var boolean theB;
  var boolean hasBeenReset;

  statemachine Counter {
    in start() <no binding>
      step(int[0..10] size) <no binding>
      out someEvent(int[0..100] x, boolean b) <no binding>
      reseted() <no binding>
    vars int[0..10] currentVal = 0
         int[0..100] LIMIT = 10
    states (initial = initialState)
    state initialState {
      on start [ ] -> countState { send someEvent(100, true && false || true); }
    }
    state countState {
      on step [currentVal + size > LIMIT] -> initialState { send reseted(); }
      on step [currentVal + size <= LIMIT] -> countState { currentVal = currentVal + size; }
      on start [ ] -> initialState {
        }
      end state
    }
    }

  var Counter c1;

  exported test case test1 {
    initsm(c1);
    assert(0) isInState<c1, initialState>;
    trigger(c1, start);
    assert(1) isInState<c1, countState>;
    test1(test case)
  }
}
2 Tools
The End.

This material is part of my upcoming (early 2013) book DSL Engineering. Stay in touch, it may become a free eBook 😊

www.voelter.de
voelterblog.blogspot.de
@markusvoelter
+Markus Voelter