Domain Specific Languages
and
Requirements (Engineering)

Andreas Graf
Andreas.graf@itemis.de

Markus Voelter
www.voelter.de
voelter@acm.org
What are Requirements?
... a requirement is a singular documented need of what a particular product or service should be or perform.
... specifies a verifiable constraint on an implementation that it shall undeniably meet or (a) be deemed unacceptable, or (b) result in implementation failure, or (c) result in system failure.
... what a system should do, and with which quality attributes, without presupposing a specific implementation.
Cohesive
Complete
Consistent
Atomic
Traceable
Current
Feasible
Unambiguous
Mandatory
Verifiable
Brain [Domain Person]

Prose

Brain [Developer]

Code
Brain [Domain Person] lossy
Prose lossy
Brain [Developer] buggy
Code
Brain [Domain Person]

???

→

Code
Brain [Domain Person]

???

↓

Code
What

How

The System shall be 99.9 % reliable

Failover, Replication, RAID
Us vs. Them
We specify the system...

...the offshore folks implement it
OEM specifies functions / interfaces ... 

... the E/E vendor develops system
Domain vs. Software
Domain

VS.

Software

Insurance contract rules...

... the actual realization as JEE app
Domain vs. Software

Communication protocol spec

The protocol handler state machine
Us -> Them

Domain -> Software

What -> How
Domain -> Software

What -> How

Informal -> Formal

Us -> Them
Informal -> Formal
Informal -> Formal
Informal -> Formal
Informal -> Formal
Formal vs. Informal
Formal Requirements & Design
FORMAL

processable by tools
Cohesive
Complete
Consistent
Atomic
Traceable
Feasible
Unambiguous
Mandatory
Verifiable

processable by tools
Requirements & Design

What
Requirements

iteration 1
Requirements & Design

What -> How

Diagram:

- Iteration 1
- Iteration 2
- Iteration 3

What Requirements

How Design
Requirements & Design

What Requirements

Iteration 1

Iteration 2

Iteration 3

How Design

Iteration 1

Iteration 2

Iteration 3

What -> How
Informal -> Formal
Requirements & Design

What
Requirements

How
Design

Iteration 1

Iteration 2

Iteration 3

...
Formal requirements specify what a system should do from a domain perspective, and with which quality attributes, without presupposing a specific software implementation, but processable by tools.
Requirement/Informal:

It shall not be possible to get radiated when operating a microwave.
Requirement/Informal:
It shall not be possible to get radiated when operating a microwave.

Design/Informal:
The radiator may only work iff the door of the microwave is closed.

+ door isolation
+ some quality requirements
Requirement/Informal:

It shall not be possible to get radiated when operating a microwave.

Design/Formal:
Domain Specific Languages
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.
general purpose
domain specific
tailor made

effective++

specialized, limited

used by experts

together with other specialized tools
execute?
DSL Program
(aka Model)

automated!

map

GPL Program
Generation
Transformation
Compilation
Interpretation
Example 1:

Embedded Protocol Handler
Factory Control System
Plug-in Cards
Componentized System
Cards have to communicate via predefined protocol

Protocol only specified as „plain text and pictures“

Verification tough, no automatic processing
Define DSL for describing the protocol, formally define protocol with it

Generate Handler

Express test cases
Component Specification

```c
// DigitalIn "BI" module

processing DigitalIn "BI" module
type 0x08 hal = DigitalInHAL {

data types {
    SinglePointIndicationWithoutTime;
    SinglePointIndicationWithTime;
    DoublePointIndicationWithoutTime;
    DoublePointIndicationWithTime;
    BitStringType8BitWithoutTime;
    BitStringType8BitWithTime;
}

parametertypes {
    DataType default {
        subattr db0 # intendedDataType == pdt SinglePointIndicationWithTime;
    };
    DebounceFilterTime default {
        attr filterTimeInMs == 0x02;
        subattr db1 # SP == 0x00;
        subattr db1 # IN == 0x00;
    };
    MaximumOscillatingFrequency;
}

function READDATA () : ProcessData;
function WRITEDATA(input : ProcessData);

struct ProcessData {
    int8 channel;
    int8 fixData[4];
}

struct Memory {
    int8 state;
    ProcessData data;
}

instance memory Memory;
```
procedure writeRegisterNumberZ requestCode 0x29 {
request: struct request1 {
  int8 acc pattern {
    2:b00;
    6:parentRequestCode;
  }
  int8 registerAddress;
};
reply: struct dontCareReply {
  int8 statusByte patternref statusByte;
  int8 dontCare patternref defaultReturn;
};
request: struct request2 {
  int8 registerType pattern {
    4:b0000;
    4:registerType;
  }
  int8 registerAddress;
  int8 registerdata [2];
} ;
}
**Testing**

```c
procedure writeRegisterNumber2 requestCode 0x29 {
    request: struct request1 {
        int8 acc pattern {
            2:b00;
            6:parentRequestCode;
        }
        int8 registerAddress;
    };
    reply: struct dontCareReply {
        int8 statusByte pattern ref statusByte;
        int8 dontCare pattern ref defaultReturn;
    };
    request: struct request2 {
        int8 registerType pattern {
            4:b0000;
            4:registerType;
        }
        int8 registerAddress;
        int8 registerdata [2];
    };
}
```

```c
test writeRegisterNumber2 for dip writeRegisterNumber2 {
    send request1 {
        attr registerAddress == reg parameterInstruction;
    };
    expect dontCareReply {
        subattr statusByte # standardStatus == 2;
    };
    send request2 {
        subattr registerType # registerType == 3;
        attr registerAddress == reg parameterInstruction;
        attr registerdata == 0x77;
        subattr registerdata # channelNumber == 5;
    };
}
```
Testing could be simplified and automated

Handler could be generated

Mistakes in Spec could be found automatically

Second „version“ could be done trivially
Eclipse Modeling
Eclipse Xtext
Example 3:
Radar Systems Engineering
Radar Systems for satellites need to be designed.

Radar requirements influence sensor design, influences satellite bus, influences launch vehicle ... circular.
Requirements cannot simply be written down because tradeoffs studies need to be performed

These need to be numerical.
Break system down into components

Use approximate numerical formulae to how requirements and design effect other components
import "classpath:/test.mm"

quantity voltage is double
quantity temperature is double

source component Sensor {
    produces m: (measurement: voltage, sensortemp: temperature)
    behavior {
        m.measurement <= sensorM[]
        m.sensortemp <= sensorT[]
    }
}

processing component TempCalibration {
    consumes input: Sensor::m
    produces calibrated: { measurement: voltage }
    behavior {
        calibrated.measurement <= m.measurement
    }
}

processing component Processor {
    consumes input: { measurement, earthtemp }
    produces earthtemp: { temp }
    behavior {
        earthtemp.temp <= processor(temp)
    }
}

sink component Output {
    consumes t: Processor::earthtemp
}

system satellite {
    s: Sensor
tc: TempCalibration
p: Processor
o: Output

    s.m -> tc.input
tc.calibrated -> p.input
p.earthtemp -> o.t

    export o.t.temp as temperature
export s.m.measurement as originalMeasurement
}
Component Behavior Specification

```
BeginPackage["MappingSatellite"]

Begin["'Private'"]

sensorM[t_Int] := 23*t
sensorT[t_Int] := 300

calibrate[ m_Double, temp_Double ] := m - temp/10

process[ m_Double ] := m^3

End[ ]

EndPackage[ ]

(* Mathematica Raw Program *)
```
Resulting System Behaviour

```plaintext
BeginPackage["satellite"]
Begin["Private"]

temperature[t_Int] := process[ calibrate[ sensorM[t], sensorT[t] ] ]
originalMeasurement[t_Int] := sensorM[t]

End[]
EndPackage[]
```
Analysis

```
In[29] = Import["L:\mathematica-test-35\mathem-test\src-gen\satellite.m"]

In[29] = Plot[{temperature[x], originalMeasurement[x]}, {x, 0, 300},
  Filling -> {1 -> 2}, AxesLabel -> {t[s], {temp, original}}]
```
Numerical approximate requirements/design tradeoffs can be performed
Eclipse Modeling
Eclipse Xtext
Wolfram Mathematica
Mathematica Workbench
Example 4:

Alarm System Menus
Alarm Systems
Operator Panels
The structure of the UI and menus of the operator panel was described in Word, and then manually implemented.

Error prone, slow, ...
Use a DSL to describe menu structures directly, and generate various artifacts from it:

- flash simulator data
- C code for implementation
- i18n templates
import "classpath:/units.md"
import "classpath:/software.swc"

namespace sl

uses units

condition Locked
condition BlinkingLight

menu Normal label "Standard menu"
  item unlockNow sys(TurnOffAlarm) if Locked
  button label "Unlock"
  submenu Manual label "Manual Settings"
    item alarmLevel sys(AlarmLevel)
      valuerange SoundLevel restrict 10..80
    item useLight sys(TurnOffAlarm) if BlinkingLight
      bool
  end
  submenu autoLocking label "Automatic Locking"
    item startTime sys(TurnOnAlarm)
    valuerange Time
    item endTime sys(TurnOffAlarm)
    valuerange Time
    template areaSettings [size=15, area=1, sw=sys(TurnOnAlarm)] area1Settings
    template areaSettings [size=10, area=2, sw=sys(TurnOnAlarm)] area2Settings
  end
end

template [size: int, area: int, sw: swref] areaSettings
  item onOrOff sys(TurnOffAlarm) labelExpr "Autolock +area+ on/off"
  bool true = label(size) "On" false = label "Off"
  item test sys(AlarmLevel) label "Test"
  bool
  item alarmLevel sys(AlarmLevel)
    valuerange SoundLevel restrict size..80
end

menu Expert extends Normal
  item master sys(UnlockNow) afterItem unlockNow
    bool
end
Software Components

```plaintext
message TurnOffAlarm
message TurnOnAlarm
message AlarmLevel
message UnlockNow

component AlarmManager {
    receives TurnOffAlarm
    receives TurnOnAlarm
    receives AlarmLevel
}

component MasterSwitch {
    receives UnlockNow
}
```
Various non-software artifacts could be generated

Integration with software structure simpler

Fewer errors, faster...
Eclipse Modeling
Eclipse Xtext
Example 5:

Requirements Tracability
Embedded Systems developed with a C-derivative
Tracability to textual requirements is necessary.
Import Requirements into the tool and then use traceability annotations to refer to them from any program element.
Imported Requirements

DUMMY REQUIREMENTS (to be replaced by interface to real RE tool)

show trace true

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>The system should start operating only after it has been initialized property [refines TwoPhases]</td>
</tr>
<tr>
<td>Efficient</td>
<td>The program should be as small regarding memory footprint as possible [refines Init refines MaxSpeed]</td>
</tr>
<tr>
<td>Cyclic</td>
<td>The actual control of the device should be based on a cyclic task</td>
</tr>
<tr>
<td>Calibration</td>
<td>The black/white values should be easily calibrated</td>
</tr>
<tr>
<td>MaxSpeed</td>
<td>Speeds per motor can only be up to 80</td>
</tr>
<tr>
<td>OptionalOutput</td>
<td>Display output should be optional</td>
</tr>
<tr>
<td>TwoPhases</td>
<td>Initialization should be separate from operation</td>
</tr>
<tr>
<td>ConsistentSetting</td>
<td>Motor settings have to be updated consistently</td>
</tr>
</tbody>
</table>
trace Cyclic
doc This is the cyclic task that is called every 1ms to do the actual control of the

    task run cyclic prio - 1 every - 2 {
        trace TwoPhases
        stateswitch linefollower
            state running
                int8 bump = 0;
bump = ecrobot_get_touch_sensor(SENSOR_PORT_T::NXT_PORT_S3);
            if ( bump == 1 ) {
                event linefollower:bumped
                terminate;
            }
            trace Init
            int32 light = 0;
            light = ecrobot_get_light_sensor(SENSOR_PORT_T::NXT_PORT_S1);
            if ( light < ( WHITE + BLACK ) / 2 ) {
                trace ConsistentSetting
                updateMotorSettings(SLOW, FAST)
            } else {
                trace ConsistentSetting
                updateMotorSettings(FAST, SLOW)
            }
        state crash
            updateMotorSettings(0, 0);
        default
            <noop>;
    }
Selecting from the Requirements

trace Cyclic

do This is the cyclic task that is called every 1ms to do the actual control of the
task run cyclic prio = 1 every = 2 {
    trace TwoPhases
    states
        Calibration: The black/white values should
        ConsistentSetting: Motor settings have to be upda
        Cyclic: the actual control of the device
        Efficient: The program should be as small
        Init: The system should start operat
        MaxSpeed: Speeds per motor can only be u
        OptionalOutput: Display output should be optio
        TwoPhases: Initialization should be separ
    }
    trace Init
    int32 light = 0;

Find Usages of Requirements

Usages

4 usages found
- LineFollower (4)
  - Ifmain (4)
  - LineFollower (4)
  - DummyRequirement (role: links; in: RequirementsTrace)
  - DummyRequirement (role: links; in: RequirementsTrace)
  - DummyRequirement (role: links; in: RequirementsTrace)
  - DummyRequirement (role: links; in: RequirementsTrace)
JetBrains MPS
What if I don‘t yet have a language?
Actually, this is the normal case!

Domain Specific Language
Building Languages
As you understand the domain...
...develop a language to express it!
Language resembles domain concepts
Then express the design with the language.
Clear understanding of the domain from building the language
Iterate!

Understand Domain

Define Language

Use Language
Iterate!

Understand Domain
- Domain Expert
- Language Engineer

Define Language
- Language Engineer

Use Language
- Domain Expert
- Domain User
Iterate!

Understand Domain
Domain Expert
Language Engineer

Define Language
FORMAL!

Use Language
FORMAL!
Like Analysis!
Like Analysis!
... but with an executable result
DSL Engineering Tools
Notations
Editors
Multi-Languages
Debugger
Testing

Groupware
Scalable
Open Source
Eclipse Public License
Large world wide community
graphical, textual and form-based DSLs
Developed by JetBrains
Open Source
Apache 2.0
Projectional Editor all kinds of notations, mainly textual
Notational Flexibility?
Rich IDEs
Rich IDEs
Languages and Editors are easier to build
Languages and Editors are easier to build

Evolve Language and simple editor as you understand and discuss the architecture, in real time!
Integrates easily with current infrastructure:
CVS/SVN diff/merge
Model evolution is trivial, you can always use grep.
Many Developers prefer textual notations.
When a graphical notation is better, you can visualize.
3.3 Commutatiegetallen op 1 leven

\[ D_x = \frac{y}{100} \times 6 \text{ Dec}(3) \]

\( \omega - x \)

\[ N_x = \sum_{t=0}^{\omega} D_x \times t \text{ Dec}(3) \]

3.6 Contante waarde 1 leven/2 levens

\[ D_x = \frac{x + n}{x} \times 19 \text{ Dec}(4) \]

\[ a_x = \frac{a_x - a_x}{x} \times 21 \text{ Dec}(3) \]

\[ \bar{a}_x = \frac{a_x - 0.5}{x} \times 22 \text{ Dec}(3) \]

\[ \frac{N_x - N_x}{D_x} = 23 \text{ Dec}(3) \]

\[ \bar{a}_{x+n} = \frac{a_x - 0.5}{x} + 0.5 * E_x \times 25 \text{ Dec}(3) \]

4 BN(_ris) koopsommen
Mixed
Multiple

```plaintext
var avg(int, 10) currentSonar = 250;

doc Statemachine to manage the transition between

statemachine linefollower {
    event initialized;
    event bumped;
    event blocked;
    event unblocked;
    initial state initializing {
        initialized [true] -> running
    }
    state paused {
        entry int16 i = 1;
        unblocked [true] -> running
    }
    state running {
        blocked [true] -> paused
        bumped [true] -> crash
        unblocked [true] -> crash
    }
    state crash {
        initialized [true] -> paused
    }
}

initialize {
    ecrobot_set_light_sensor_active(int);
}
```

---

<table>
<thead>
<tr>
<th>linefollower</th>
<th>initialising</th>
<th>paused</th>
<th>running</th>
<th>crash</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialized</td>
<td>true</td>
<td></td>
<td>running</td>
<td>true paused</td>
</tr>
<tr>
<td>bumped</td>
<td></td>
<td>true crash</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blocked</td>
<td></td>
<td>true paused</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unblocked</td>
<td></td>
<td>true running</td>
<td>true crash</td>
<td></td>
</tr>
</tbody>
</table>

initialize {
    ecrobot_set_light_sensor_active(SENSOR_PORT_T::NXT_PORT_S1);
}
Standards & UML
You can model everything with UML
You can model everything with UML somehow!
Problem

Shoehorning domain abstractions into the generic language
Problem

Sidetracked by existing abstractions and notations
Problem

UML

Theory
Notations/Abstractions extensible via Profiles

Practice
Very Limited Tool Support!
Problem

Meta Model Complexity!
But! But don’t reinvent the wheel either.
Where are standards useful?
People have to learn underlying concepts anyway.
Is UML with a profile still a standard language?
On which meta level do I want to standardize? M2 (UML), M3 (MOF)?
Isn’t a **DSL** based on **MOF** as „standard“ as a profile based on **UML**?
Similar statements can be made relative to BPMN
Textual Requirements?
Textual Requirements

still necessary.
left Arm and right Arm is equal to "50" accelerometer_speed is greater than "100" accelerometer_gain.
Tracing

doc This is the cyclic task that is called every 1ms to do the actual control of the

```c
trace Cyclic
task run cyclic prio = 1 every = 2 {
    trace TwoPhases
    stateswitch linefollower
        state running
            int8 bump = 0;
            bump = ecrobot_get_touch_sensor(SENSOR_PORT_T::NXT_PORT_S3);
            if ( bump == 1 ) {
                event linefollower::bumped
                terminate;
            }
        } trace Init
            int32 light = 0;
            light = ecrobot_get_light_sensor(SENSOR_PORT_T::NXT_PORT_S1);
            if ( light < (WHITE + BLACK) / 2 ) {
                trace ConsistentSetting
                updateMotorSettings(SLOW, FAST)
            } else {
                trace ConsistentSetting
                updateMotorSettings(FAST, SLOW)
            }
        state crash
            updateMotorSettings(0, 0);
        default
            <noop>;
    
    ```
trace Cyclic

This is the cyclic task that is called every 1ms to do the actual control of the task.

task run cyclic prio = 1 every = 2 {
    trace TwoPhases

    states
        Calibration: The black/white values should
        ConsistentSetting: Motor settings have to be updated
        Cyclic: the actual control of the dev
        Efficient: The program should be as small as possible
        Init: The system should start operation
        MaxSpeed: Speeds per motor can only be used
        OptionalOutput: Display output should be optional
        TwoPhases: Initialization should be separated

    state run
        int8 bump = 0;
        int8 bu = 1;
        if ( bu = 0 )
            states
                Calibration: The black/white values should
                ConsistentSetting: Motor settings have to be updated
                Cyclic: the actual control of the dev
                Efficient: The program should be as small
                Init: The system should start operation
                MaxSpeed: Speeds per motor can only be used
                OptionalOutput: Display output should be optional
                TwoPhases: Initialization should be separated

        int32 light = 0;
        light = ecrobot_get_light_sensor(SENSOR_PORT_T::NXT_PORT_S1);

        if ( light < ( WHITE + BLACK ) / 2 ) {
            trace ConsistentSetting
            updateMotorSettings(SLOW, FAST)
        } else {
            trace ConsistentSetting
            updateMotorSettings(FAST, SLOW)

        state crash
            updateMotorSettings(0, 0);
        default
            <noop>;
Things to keep in mind
Limit Expressiveness
Notation is important!
Graphical vs. Textual
Invest in good constraints
Testing may be an important benefit of your DSL
Simulations („play around“ with the models)
Who are 1st Class Citizens?
Learn from - but don‘t copy – programming languages
Support for Reuse and Variations
Tooling Matters!
Annotation Models to add technical details
Develop the language iteratively!
Co-Evolve Language and Concepts
Can domain users actually „program“?
Domain Users vs. Experts
Summary
Cohesive
Complete
Consistent
Atomic
Traceable
Current
Feasible
Unambiguous
Mandatory
Verifiable
Cohesive
Complete
**Consistent**
Atomic
Traceable
Current
Feasible
Unambiguous
Mandatory
Verifiable

Validation
Checking
Simulation
Cohesive
Complete
Consistent
Atomic
Traceable
Current
Feasible
Unambiguous
Described Formally
Mandatory
Verifiable
Cohesive
Complete
Consistent
Atomic
Traceable
Current
Feasible
Unambiguous
Mandatory
Verifiable

Domain Expert involved in Definition and Review
Cohesive  Complete  Consistent  Atomic  Traceable  Current  Feasible  Unambiguous  Mandatory  Verifiable

**Executable**  Automatic Refinement downstream, Code Gen.
Cohesive
Complete
Consistent
Atomic
Traceable
Current
Feasible
Unambiguous
Mandatory
Verifiable
Executable

Reward for the additional effort of formalization!
And Developers???
And Developers???

... Languages
... Technology Evaluation
... Generators
... Testing
... Operations

... what they want to do anyway!
Brain [Domain Person]

Code
One more thing:
One more thing:
Why all these pictures?
I like airplanes.
„Build a research airplane for high AoA and thrust vector control“
„Build a research airplane for high AoA and thrust vector control“

Not enough!
“Build a research airplane for high AoA and thrust vector control”

Not enough!

Systems Engineering
“Build a research airplane for high AoA and thrust vector control“

Not enough!

Systems Engineering

Requirements ... Design ... continuous. Early Formalization of many aspects.
And:
The airplane is custom-built for the task.
One size does not fit all.
THE END.

coordinates

web      www.voelter.de
email    voelter@acm.org
skype    schogglad
xing     http://www.xing.com/profile/Markus_Voelter
linkedin http://www.linkedin.com/pub/0/377/a31