Working with multiple languages

Why and How

Markus Voelter
independent/itemis
voelter@acm.org

based on the
DSL Engineering book
and the mbeddr
tool

www.voelter.de
voelterblog.blogspot.de
@markusvoelter
+Markus Voelter
I think there is a world market for maybe five computers.

Thomas J. Watson
640K is more memory than anyone will ever need.

Bill Gates
The Macintosh uses an experimental pointing device called a "mouse". There is no evidence that people want to use these things.

John C. Dvorak
Why would you want support for (modular) language composition?

You can always "cobble together" a few separate DSLs.
(1) Concepts
(2) Tools
(3) Demo
1 Concepts

Introduction
<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>

**General Purpose**

- **C**
- **Components**
  - **State Machines**
  - **Sensor Access**
- **Domain Specific**
  - **LEGO Robot Control**
Big Language

with many first class concepts!

Small Language

with a few, orthogonal and powerful concepts
Modular Language

with many optional, composable modules
Shorter Programs

More Accessible Semantics

For a limited Domain!

Domain Knowledge encapsulated in language
Multi-Stage Transformation
What does it all mean?

Execution Semantics

Transformation
Multi-Stage

Modularization

Multi-Stage: Reuse

Reusing Later Stages Optimizations!
Multi-Stage: Reuse

Robot Control
State Machine

Components

C (MPS tree)

C Text

Example
Extended C

Multi-Stage: Reuse

Robot Control
State Machine

Components

C (MPS tree)

C Text

Example
Extended C
Multi-Stage: Reuse

Reusing Early Stages
Portability

Multi-Stage: Reuse

Java
C#

Example
Extended C
Multi-Stage: Preprocess

Adding an optional, modular emergency stop feature
Separation of Concerns - Referencing

Several Concerns
... in one domain
Several Concerns
... in one domain

integrated into
one fragment

separated into
several fragments

Viewpoints

Hardware

Behaviour

Tests

Example
Refrigerators
**Viewpoints:** Why?

- Sufficiency
- Different Stakeholders
- Different Steps in Process - VCS unit!

**Referencing**

Fragment

Fragment

Fragment
Viewpoints

Hardware \( \rightarrow \) Behaviour

Behaviour \( \rightarrow \) Tests

Tests \( \rightarrow \) Hardware

Example
Refrigerators
Progressive Refinement
Referencing

<table>
<thead>
<tr>
<th>Reuse</th>
<th>Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referencing</td>
<td>Extension</td>
</tr>
</tbody>
</table>

Dependent

No containment
Referencing

Example
Refrigerators
Growing a Language - Extension

Extension

more specialized domains
more specialized languages
more specialized domains
more specialized languages

Extension

more specialized domains
more specialized languages

Extension

All programs writable in C
more specialized domains
more specialized languages

more specialized domains
more specialized languages
Extension

\[ D_{n+1} \]

\[ \Downarrow \]

\[ \Downarrow \]

\[ \begin{array}{c}
D_n \\
\end{array} \]

\[ \begin{array}{c}
\text{==} \\
\end{array} \]

\[ \begin{array}{c}
\text{\textbullet} \\
\end{array} \]
Extension

$D_{n-1}$

$D_n$

Good for **bottom-up** (inductive) domains, and for use by **technical** DSLs (people)
module main imports OsekKernel, ECAFI, BitLevelUtilities {
  constant int WHITE = 500;
  constant int BLACK = 700;
  constant int SLOW = 20;
  constant int FAST = 40;

  statemachine linefollower {
    event initialized;
    initial state initializing {
      initialized [true] -> running
    }
    state running () {
    }
  }

  initialize {
    ecrobot_set_light_sensor_active [SENSOR_PORT_T::NXT_PORT_S1];
    event linefollower:initialized
  }

  terminates {
    ecrobot_set_light_sensor_inactive [SENSOR_PORT_T::NXT_PORT_S1];
  }

  task run cyclic prio = 1 every = 2 {
    stateswitch linefollower
    state running
    int32 light = 0;
    light = ecrobot_get_light_sensor [SENSOR_PORT_T::NXT_PORT_S1];
    if ( light < ( WHITE + BLACK ) / 2 ) {
      updateMotorSettings(SLOW, FAST);
    } else {
      updateMotorSettings(FAST, SLOW);
    }
    default
    <noop>
  }

  // void updateMotorSettings( int left, int right ) {
  //   nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_T::FAST, FAST);
  //   nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_T::SLOW, SLOW);
  // }

}
Extension

Diagram:

```
Module
  ↓
IModuleContent
  ↓
Struct
  ↓
Function
  ↓
State
  ↓
Machine
```

Example Extended C

```c
int[] arr = ...
for (int i=0; i<arr.size(); i++) {
    sum += arr[i];
}
```

```c
int[] arr = ...
List<int> l = ...
for (int i=0; i<arr.size(); i++) {
    l.add( arr[i] );
}
```
Overspecification! Requires Semantic Analysis!

Declarative! Directly represents Semantics.

Linguistic Abstraction
Def: DSL

A DSL is a **language** at D that provides **linguistic abstractions** for **common patterns and idioms** of a language at D-1 when used within the domain D.

---

Def: DSL cont’d

A good DSL does **not** require the use of patterns and idioms to express **semantically interesting** concepts in D. Processing tools do not have to do "semantic recovery" on D programs.

**Declarative!**
Extension

Drawbacks

tightly bound to base
potentially hard to analyze
the combined program

Extension Composition

Extension combination refers to the ability to use a number of independent extensions of the same base language in a single program.

Critically Important!
Language Embedding
Embedding often uses Extension to extend the embedded language to adapt it to its new context.
A Systematic Approach

Language Modularity, Composition and Reuse (LMR&C) increase efficiency of DSL development
Language Modularity, Composition and Reuse (LMR&C) increase efficiency of DSL development

4 ways of composition:
- Referencing
- Reuse
- Extension
- Reuse

Language Modularity, Composition and Reuse (LMR&C) increase efficiency of DSL development

4 ways of composition:
- distinguished regarding dependencies and fragment structure
Dependencies:

do we have to know about the reuse when designing the languages?

Fragment Structure:

homogeneous vs. heterogeneous („mixing languages”)

Dependencies & Fragment Structure:

<table>
<thead>
<tr>
<th>independent languages dependencies dependent</th>
<th>Reuse</th>
<th>Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referencing</td>
<td></td>
<td>Extension</td>
</tr>
<tr>
<td>homogeneous fragile structure</td>
<td></td>
<td>heterogeneous fragile structure</td>
</tr>
</tbody>
</table>

Dependencies & Fragment Structure:

Referencing

Reuse

Extension

Embedding
Concrete Syntax
Combinations

```java
// interface Decider {
    int decide(int x, int y) pre
}

// component AComp extends nothing {
    ports:
        provides Decider decider
    contents:
        int decide(int x, int y) <- op decider.decide {
            int, 0
            x == 0 x > 0
            y == 0 0 1
            y > 0 1 2
        }
    }
```
Example Tools

Parser-based LL(k)
Example Tools
Projectional

MPS

Example Tools
Parser-based GLR

Spoofax
Language Extension

C extensibility in mbeddr

```c
struct Position {
    uint16 x;
    uint16 y;
};

int8 add(int8 x, int8 y) {
    return x + y;
} add (function)

void normalizePosition(Position* p) {
    if (p->x > 100) {
        p->x = 100;
    } if
    if (p->y > 100) {
        p->y = 100;
    } if
} normalizePosition (function)

Plain C
Language Extension

C extensibility in mbeddr

Plain C

```
struct Position {
    uint16 x;
    uint16 y;
};

int8 add(int8 x, int8 y) {
    return x + y;
} add (function)

void normalizePosition(Position* p) {
    if (p->x > 100) {
        p->x = 100;
    }
    if (p->y > 100) {
        p->y = 100;
    }
}
```

C + Units

```
exported struct Trackpoint {
    int8 id;
    int8/s/ timestamp;
    int8/m/ x;
    int8/m/ y;
    uint16/m/ alt;
    int16/mps/ speed;
};
```

Language Extension

C extensibility in mbeddr

```
requirements modules: FlightJudgementRules
module StateMachines imports DataStructures, stdlib_stub, stdio_stub {
    #define TAKEDOFF = 100;
    #define HIGH_SPEED = 10;
    #define VERY_HIGH_SPEED = 20;
    #define LANDING = 100;

    statemachine HierarchicalFlightAnalyzer initial = beforeFlight {
        in next(Trackpoint* tp) < no binding>
        in reset() < no binding>
        out crashNotification() < no binding>
        readable var int16 points = 0
        state beforeFlight {
            on next (tp->alt > 0 a) -> airborne
            exit { points += TAKEDOFF; }
        }
        state beforeFlight
        composite state airborne initial = landed {
            on reset ( ) -> beforeFlight { points = 0; }
            on next (tp->alt == 0 && tp->speed == 0 mps) -> crashed
            state landed (airborne.landed) {
                entry { points += LANDING; }
            }
            state landed
            state airborne
            state crashed {
                entry { send crashNotification(); }
            }
            state crashed
        }
    }
```
Language Extension

C extensibility in mbeddr

```c
#define TAKEOFF = 100;
#define HIGH_SPEED = 10;
#define VERY_HIGH_SPEED = 20;
#define LANDING = 100;

stateMachine HierarchicalFlightAnalyzer initial = beforeFlight {
  on next(Trackpoint* tp) << bindings(
    out crashNotification() << bindings(
      readable var int points = 0
      state beforeFlight {
        on next (tp->alt > 0 m) -> airborne
        exit ( points == TAKEOFF );
      }
      state airborne
      composite state airborne_initial = landed {
        on reset [ ] -> beforeFlight ( points = 0 );
        on next (tp->alt == 0 m & & tp->speed == 0 mps) -> crashed
        state landed (airborne, landed) {
          entry ( points == LANDING );
        } state landed
      state airborne
      state crashed {
        entry ( send crashNotification(); )
      } state crashed
    }
    )
  }
}
```

C + State Machines

... and Units!

→ combination of independent extensions!

Language Extension

„Embedding“ Xbase into DSL

```java
entity Person {
  lastname : String
  firstname : String
  String fullName(String from) {
    return "Hello " + firstname + " " + lastname + " from " + from
  }
}
```

Entities using Java types and Xtend code in the body of operations.
See before … results in generator, scoping, typing, etc. with little additional work.
Language Extension
Extending Xbase with Dates

Grammar

\[
\text{XDateLiteral}: \quad \text{‘date’ ‘:’ year=INT ‘-’ month=INT ‘-’ day=INT;}
\]

\[
\text{XLiteral returns } xbase::XExpression:
\quad \text{XClosure} \mid
\quad \text{XBooleanLiteral} \mid
\quad \text{XIntLiteral} \mid
\quad \text{XNullLiteral} \mid
\quad \text{XStringLiteral} \mid
\quad \text{XTypeLiteral} \mid
\quad \text{XDateLiteral;}
\]

Repeat complete XLiteral rule, adding the XDateLiteral

Language Extension
Extending Xbase with Dates

Type system

```java
@Singleton
public class DomainModelTypeProvider extends XbaseTypeProvider {
    @Override
    protected JvmTypeReference type(XExpression expression,
            JvmTypeReference rawExpectation, boolean rawType) {
        if (expression instanceof XDateLiteral) {
            return _type((XDateLiteral) expression, rawExpectation, rawType);
        }
        return super.type(expression, rawExpectation, rawType);
    }

    protected JvmTypeReference _type(XDateLiteral literal,
            JvmTypeReference rawExpectation, boolean rawType) {
        return getTypeReferences().getTypeForName(Date.class, literal);
    }
}
```
Language Extension

Extending Xbase with Dates

**Interpreter**

```java
public class DateExtensions {
    public static long operator_minus(Date a, Date b) {
        long resInMilliSeconds = a.getTime() - b.getTime();
        return millisecondsToDays( resInMilliSeconds );
    }
}
```

**Compiler**

```java
public class DomainModelCompiler extends XbaseCompiler {
    protected void toJavaExpression(XODateLiteral expr, IAppendable b) {
        b.append("new java.text.SimpleDateFormat("yyyy-MM-dd").parse(" + expr.getYear() + "." + expr.getMonth() + "." + expr.getDay() + ")");
    }
}
```
Embedding

Useful for Meta Data

```c
#include <stdbool.h>
#include <stdio.h>

#define REALISE 10
#define HIGH_SPEED 20
#define VERY_HIGH_SPEED 30

int main() {
    // Code goes here
}
```

No Dependencies on anything.
Meta

Extending the Tool
... since it is bootstrapped!

```java
public boolean islinear(node<Expression> expr) {
    return dispatch<bool>(expr) {
        NumberLiteral => true
        EnumLiteralRef => true
        LiteralWithUnit => #it.value
        BinaryComparisonExpr => #it.left && #it.right
        BinaryLogicalExpression => #it.left && #it.right
        PlusExpression => #it.left && #it.right
        LocalVarRef => true
        ArgumentRef => true
        MemberRef => true
        StdArrowExpression => #it.member
        StdDotExpression => #it.member
        NotExpression => #it.expression
        GlobalVarRef => true
        MultiExpression => it.left.isStaticallyEvaluable() || it.right.isStaticallyEvaluable()
        DivExpression => it.left.isStaticallyEvaluable() || it.right.isStaticallyEvaluable()
        default false
    }
}
```
Extending the Tool

... since it is bootstrapped!

Extending MPS is (almost) like extending any other language!
Multi-Stage Trafo

MPS' Transformations

```java
module Statemachine imports nothing {

    statemachine Counter {
        in events
        start() {
            step(int[0..10] size)
        }
        out events
        started() {
            resettod()
            incremented(int[0..10] newVal)
        }
        local variables
        int[0..10] currentVal = 0
        int[0..10] LIMIT = 10
        states (initial = start)
        state start {
            on start [ ] -> countState (send started();)
        }
        state countState {
            on step [currentVal + size < LIMIT] -> start (send resettod());
            on step [currentVal + size >= LIMIT] -> countState {
                currentVal = currentVal + size;
                send incremented(currentVal);
            }
            on start [ ] -> start (send resettod());
        }
        Counter c1;
        Counter c2;
        void #function() {
            trigger(c1, start);
        }
    }
}
```
Multi-Stage Trafo
MPS’ Transformations

Transformations
with concrete Syntax & IDE

IDE support for target language
Macros contain template code
M2M with M2T „feel“
Transformations with concrete Syntax & IDE

```
template weaveStateMachineTypesStuffIntoModule
input StateMachine
content node:
module dummy imports nothing {
  <TF [exported enum $[statemachineInEvents] { $LOOPS[$anEvent]; }] TF>
  <TF [exported enum $[statemachineStates] { $LOOPS[$aState]; }] TF>
  <TF [exported struct $[statemachineData] {
    ->$[statemachineStates] __currentState;
    $LOOPS[$COPY_SRC$[int8_t]$[smLocalVar]; ]
  };
};
}
```

```
comment : <none>
mapping label : <no label>
mapped nodes : (node, genContext, operationContext)->sequence<node> { node.states; }
```

Transformations with concrete Syntax & IDE

```
template generateSwitchCase
input StateMachine
content node:
module dummy imports nothing {
  enum events { anEvent; }
  enum states { aState; }
  struct statemachineData {
    states __currentState;
  };
  void statemachineFunction(statemachineData* instance, events event) {
    switch ( instance->__currentState ) {
      $LOOPS$ case ->$[aState]; {
      $LOOPS$ case ->$[anEvent]; {
        $LOOPS$ if ($COPY_SRC$[true]) {
          $COPY_SRC$[int8_t] transitions;
          $COPY_SRC$[int8_t entryActions];
          instance->__currentState ->$[aState];
          $COPY_SRC$[int8_t exitActions];
          if;
        } else;
        break;
      } switch
      case break;
    } switch
  } statemachineFunction (function)
```

```
Transformations with concrete Syntax & IDE

Transformations with local dependencies

A transformation only specifies its relative priority to other transformations as necessary.

This leads to composability also of transformations.
Find References

Custom Finders

```java
finder findProviders for concept Interface
    description: Providers
    find(node, scope)->void {
        nlist<> refs = execute NodeUsages ( node , <same scope> );
        foreach r in refs.select(it|it.isInstanceOf(ProvidedPort)) {
            add result r.parent ;
        }
    }

getCategory(node)->string {
    "Providers";
}
```

Find References

Custom Finders
Parser/Syntax Testing

Declarative Approach

abstract syntax

```java
// Test multiply and add [[1 + 2 * 3]] parse to Add(_, Mul(_, _))
// Test add and multiply [[1 * 2 + 3]] parse to Add(Mul(_, _), _)
// Test add and add [[1 + 2 + 3]] parse to Add(_, _, _)
```

concrete syntax

```java
// Test multiply and add [[1 + 2 * 3]] parse to [[1 + (2 * 3)]]
// Test add and multiply [[1 * 2 + 3]] parse to [[[1 * 2] + 3]]
// Test add and add [[1 + 2 + 3]] parse to [[[1 + 2] + 3]]
```

Constraints/Type Sys.

Declarative Approach

```
Test case testSubtyping
    nodes
        ([<check types module TestSubtyping imports nothing { >})
            <node double d = 10;>
            double d2 = <def d>;
            double d3 = <node d has type double>;
            int8 i = <node d has error>;
       ]
    test methods
        test testReference {
            assert dref.var == dnode;
        }
```

Test annotations on the test subject - very concise.
## Testing Semantics

Expressing tests in language

```java
module UnitTestDemo {
    int32 main(int32 argc, int8* argv) {
        return test testMultiply;
    }

    test case testMultiply {
        assert (0) times2(21) == 42;
        assert (1) times2(0) == 0;
        assert (2) times2(-10) == -20;
    }

    int8 times2(int8 a) {
        return 2 * a;
    }
}
```

Generate/Interpret the tests along with the application code

## Testing Editors

Using Moritz Eysholdt's Xpect

```java
// XPECT.TEST org.example.MyJUnitContentAssistTest END.TEST

// XPECT contentAssist at |Hel --> Hello
Hello Peter!

// XPECT contentAssist at |! --> !
Hello Heiko!
```

Test expectations in comments.

Can test various language aspects such as scopes, code completion, type system; also extensible.
Inspecting/Debugging
Debugging DSLs

Transformations & Code Gens

Xtext

Debugging DSLs

Transformations

MPS

Multi-Level, Declarative:
show intermediate models and the templates involved in trafo.
Debugging DSL Programs
Xtext

Xbase Automatic Debugging

Make your DSL extend Xbase
Use a JVMModelInferrer → Java
you then get Debugging for free

Debugging DSL Programs
MPS

Extensible Debuggers

```java
int8 s = 0;
int8[] a = {1, 2, 3};
foreach (a sized 3) {
    s += it;
}
```
Debugging DSL Programs

Extensible Debuggers

Language Extension

int8 s = 0;
int8[] a = {1, 2, 3};
foreach (a sized 3) {
    s += it;
}

Generated C Code

int8 s = 0;
int8[] a = {1, 2, 3};
for (int __c = 0; __c < 3; __c++) {
    int8 __it = a[__c];
    s += __it;
}

Stepping

```c
void contributeStepOverStrategies(list<IDebugStrategy> res) {
    ancestor
    statement list: this.body
}
```

```c
void contributeStepIntoStrategies(list<IDebugStrategy> res) {
    subtree: this.array
    subtree: this.len
}
```
Debugging DSL Programs

Extensible Debuggers

```java
int8 s = 0;
int8[] a = {1, 2, 3};
foreach (a sized 3) {
    s += it;
}
```

Watches

```java
void contributeWatchables(list<UnmappedVariable> unmapped,
                          list<IWatchable> mapped) {
    hide "_c"
    map "_it" to "it"
    type: this.array.type : ArrayType.baseType
    category: WatchableCategories.LOCAL_VARIABLES
    context: this
}
```

Debugging DSL Programs

Debugger Architecture

![Diagram of Debugger Architecture]
3 Demo
An extensible Version of C (and an IDE and a Debugger)
The End.

voelter.de | dslbook.org

@markusvoelter
+Markus Voelter