10 Thoughts
2 Demos
* Discussions
Embedded Development

Two Classes in Embedded Systems Development

(Yes, this is a slight simplification)
Plan Oriented
Top Down
Big Systems
Big Companies
Modeling

Grown
Bottom Up
Small/Medium Systems
Smaller Companies
C Code
Programming vs. Modeling

programming? vs. modelling?
different? the same?

We don’t want to model, we want to program!

... at different levels of abstraction
... from different viewpoints
... integrated!
We don’t want to model, we want to program!

... with different degrees of domain-specificity
... with suitable notations
... with suitable expressiveness

And always:
precise and tool processable
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.
Linguistic Abstraction

What’s the Problem here?

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

// B
List<Integer> l = ...
for (int i = 0; i < arr.size(); i++) {
    l.add(arr[i]);
}
```

Much better with new **linguistic abstraction**

```java
// A'
for (int i in arr) {
    sum += i;
}

// B'
seqfor (int i in arr) {
    l.add(arr[i]);
}
```

No sophisticated analysis required to understand the semantics of a construct and treat it “correctly”.

Linguistic Abstraction

What’s this? And what’s the Problem?

```java
var linefollower_states_enum linefollower_currentstate = linefollower_states_enum.EVENT_INITIALIZED;
enum linefollower_events_enum { EVENT_INITIALIZED, EVENT_BOUNDED, EVENT_BLOKKED, EVENT_UNLOCKED }
enum linefollower_states_enum { STATE_INITIALIZING, STATE_RUNNING, STATE_ERROR, STATE_IDLE }
void linefollower_ghosts(linefollower_states_enum event) {
    if (linefollower_currentstate == linefollower_states_enum::EVENT_INITIALIZED) {
        if (true) {
            linefollower_currentstate = linefollower_states_enum::STATE_RUNNING;
            return;
        }
    }
...
```
Linguistic Abstraction

**Much better!**

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

Linguistic Abstraction

**Much better!**

<table>
<thead>
<tr>
<th>linefollower</th>
<th>initializing</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></td>
</tr>
<tr>
<td>bumped</td>
<td></td>
<td></td>
<td></td>
<td>true crash</td>
</tr>
<tr>
<td>blocked</td>
<td></td>
<td>true paused</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unblocked</td>
<td></td>
<td></td>
<td></td>
<td>true running</td>
</tr>
</tbody>
</table>
Domains are Hierarchical

Creating Linguistic Abstractions
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.

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.

**Modular Languages**
Programming Languages are not MODULAR enough.

Programming Languages are not COMPOSABLE enough.
Programming Languages are not CONFIGURABLE enough.

Programming Languages are not ADAPTABLE enough.
Programming Language Syntax is not FLEXIBLE enough.

Big Language? with many first class concepts!
Small Language?

with a few, orthogonal and powerful concepts

Modular Language

with many optional, composable concepts
Modular Language

Like frameworks and libraries,

but with syntax and IDE support

5

Concerns & Viewpoints
Viewpoints
suitable abstractions and notations for each

Viewpoints
Integrated via symbolic references and seamless transitions
Several Concerns in a Domain

A: mixed
B: separate Viewpoints
6
Domain-Specificity

Viewpoints
General Purpose
predefined library
configure
Viewpoints
Domain Specific

custom
purpose-built
create/include

Viewpoints
Domain Specific
Custom
Notations
real
business
expert integration
The LWES Project
Incremental Extension of C with DSLs for Embedded Systems, integrated with Formal Methods and support for PLE and Requirements Tracing
Language Workbenches

Freely define languages and integrate them

(Martin Fowler)
Language Workbench
(Martin Fowler)

use persistent abstract representation

Language Workbench
(Martin Fowler)

language ::= schema + editors + generators
Language Workbench
(Martin Fowler)

projectional editing

Language Workbench
(Martin Fowler)
persist incomplete or contradictory information
Language Workbench
(Martin Fowler)

powerful editing + testing refactoring debugging groupware

language definition implies IDE definition

Language Workbench
(Martin Fowler)

support for "classical" programming "classical" and modeling
Syntax primarily textual

with more symbols
think: mathematics
Syntax
primarily
textual
sometimes
box&line style

Syntax
primarily
textual
sophisticated
visualizations
Projectional Editing

Parser-based

... to tree
... to text
Projectional tree
... to text-lookalike (editor)
... to other trees ... [*]
... to text

Language Composition
There’s no parsing.
Unique Language Element Identity.
Unlimited language composition.
Flexible Notations

Textual
like ASCII

Graphical
box & line

Semi-Graphical
mathematical

} treated the same
can be mixed

Automatic IDE Extension

tool support is inherent
for languages build with
projectional tools

language definition
implies IDE definition
Multiple Notations

... for the same concepts
e.g. in different contexts
or for different tasks

Partial Projections

... different views
... for different roles/people
... only a particular variant
Storage

!= Schema

... store arbitrary meta data
  change log
  conflicting information
  variability annotations

... independent of language schema!

... „aspects“, overlay

Live

Programs

think: spreadsheet

a change to one part of program can lead to (dependent) changes in other parts
Tree Editing

... is different from editing text

... try to make it feel like text

... takes some getting used to

but: for more flexible notations
a more general editing paradigm is needed

Infrastructure Integration

... storage is not text

... diff/merge must be in tool

... existing text tools don’t work
Proprietary Tools

... no standards

... no interop

(4)

Modular Languages

ctd.
Modular Language

with many optional, composable concepts

When DSL (for several concerns) are developed from scratch, as a group, then dependencies between the concerns can be materialized as dependencies between the languages and the language concepts.
A language B extends another language A if B **contains additional language concepts.** This means that for programs written in B, all concepts from A are available, plus those defined in B.

A language has been developed to be used in contexts not known at the time of development. **No dependencies allowed!** The reusable language has to be extended so it can reference concepts from context languages.
Language Embedding

Composition is a **special case of reuse**, where the reused language is **syntactically embedded** into languages from the context.

JetBrains MPS
http://jetbrains.com/mps

Open Source (Apache 2.0)
Projectional Editor
Very good at lang. Compososition
Version 2.0 a month ago:
  - Improved performance
  - Unified generate/compile/build
  - Debug MPS in MPS
  - Tables in the editor
Open Source (Apache 2.0)
Projectional Editor
Very good at lang. Composition
Version 2.0.1 today:
  Bug fixes. Dozens of them 😊

Open Source (Apache 2.0)
Projectional Editor
Very good at lang. Composition
Version 2.0.1 today:
Version 2.0.1 soon
  More Bug fixes.
Open Source (Apache 2.0) 
Projectional Editor
Very good at lang. Compososition
Version 2.0.1 today:
Version 2.0.1 soon 
Version 2.1 early 2012

Graphical Editors, Several Editors per Concept, Wiki-Language and more.