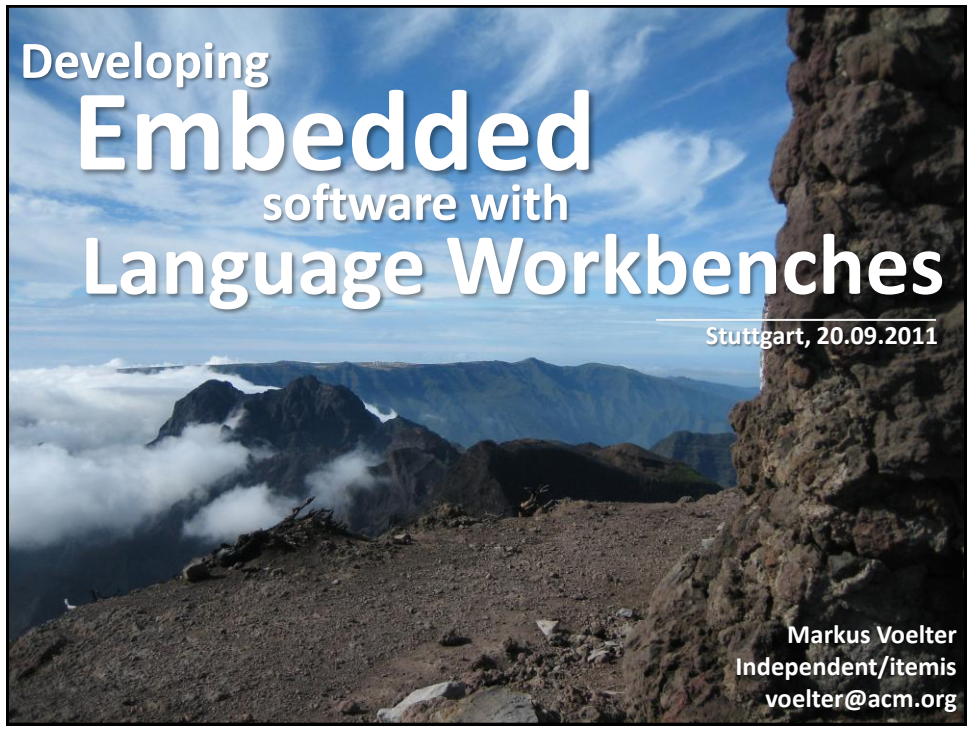


10 Thoughts

2 Demos

* Discussions

1

Embedded Development

Two Classes in Embedded Systems Development

(yes, this is a slight simplification)

Plan Oriented
Top Down
Big Systems
Big Companies

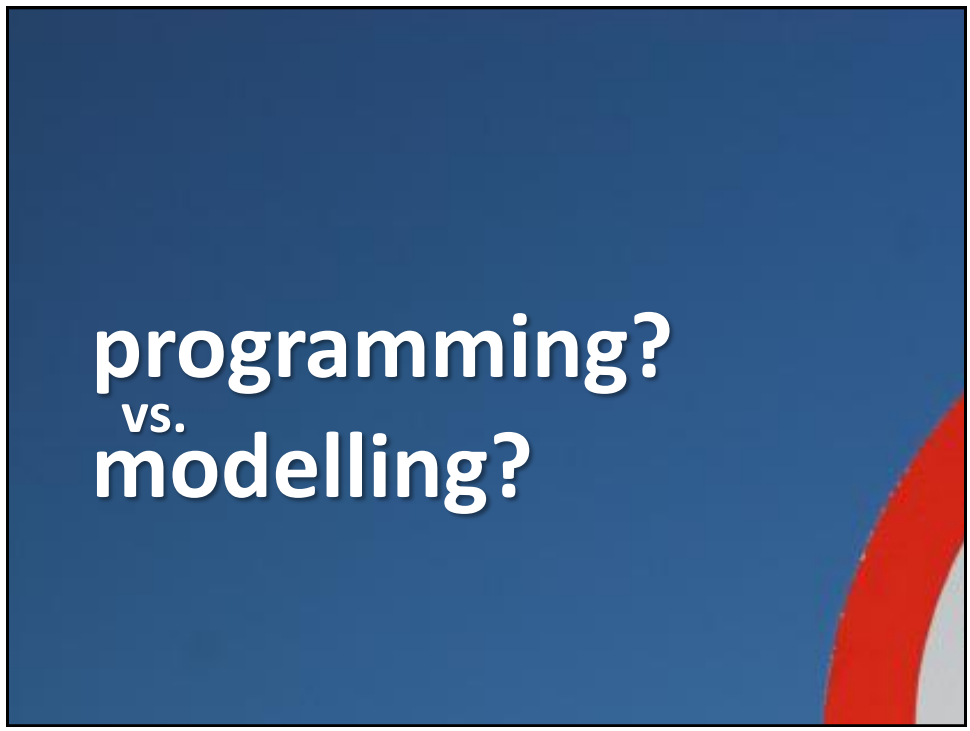
Modeling

Grown
Bottom Up
Small/Medium Systems
Smaller Companies

C Code

2

Programming vs. Modeling



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**

We don‘t want to
model,
we want to
program!

And always:
precise and **tool processable**

3 Domain-Specific Languages

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.

Linguistic Abstraction

What's the Problem here?

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

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

Much better with new linguistic abstraction

```
// 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 contruct and treat it "correctly".

Linguistic Abstraction

What's this? And what's the Problem?

```
var linefollower_states_enum linefollower_currentstate = linefollower_states_enum::STATE_INITIALIZING;

enum linefollower_events_enum { EVENT_INITIALIZED, EVENT_BUMPED, EVENT_BLOCKED, EVENT_UNBLOCKED }

enum linefollower_states_enum { STATE_INITIALIZING, STATE_PAUSED, STATE_RUNNING, STATE_CRASH }

void linefollower_execute( linefollower_events_enum event ) {
  if ( linefollower_currentstate == linefollower_states_enum::STATE_INITIALIZING ) {
    if ( event == linefollower_events_enum::EVENT_INITIALIZED ) {
      if ( true ) {
        linefollower_currentstate = linefollower_states_enum::STATE_RUNNING;
        return;
      }
    }
  }
  if ( linefollower_currentstate == linefollower_states_enum::STATE_PAUSED ) {
    if ( event == linefollower_events_enum::EVENT_UNBLOCKED ) {
      if ( true ) {
        linefollower_currentstate = linefollower_states_enum::STATE_RUNNING;
        return;
      }
    }
  }
  if ( linefollower_currentstate == linefollower_states_enum::STATE_RUNNING ) {
    if ( event == linefollower_events_enum::EVENT_BLOCKED ) {
      if ( true ) {
        linefollower_currentstate = linefollower_states_enum::STATE_PAUSED;
        int16 i = 1;
        return;
      }
    }
    if ( event == linefollower_events_enum::EVENT_BUMPED ) {
      if ( true ) {
        linefollower_currentstate = linefollower_states_enum::STATE_CRASH;
        return;
      }
    }
  }
}
```

Linguistic Abstraction

Much better!

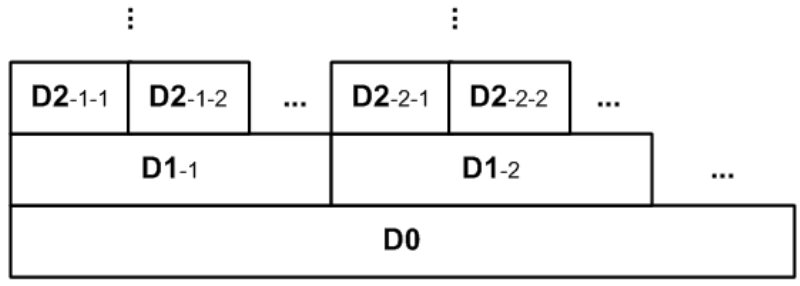
```
statemachine linefollower {
  event initialized;
  event bumped;
  event blocked;
  event unblocked;
  initial state initializing {
    initialized [true] -> running
  }
  state paused {
    entry int16 i = 1;
    unblocked [true] -> running
  }
  state running {
    blocked [true] -> paused
   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

Linguistic Abstraction

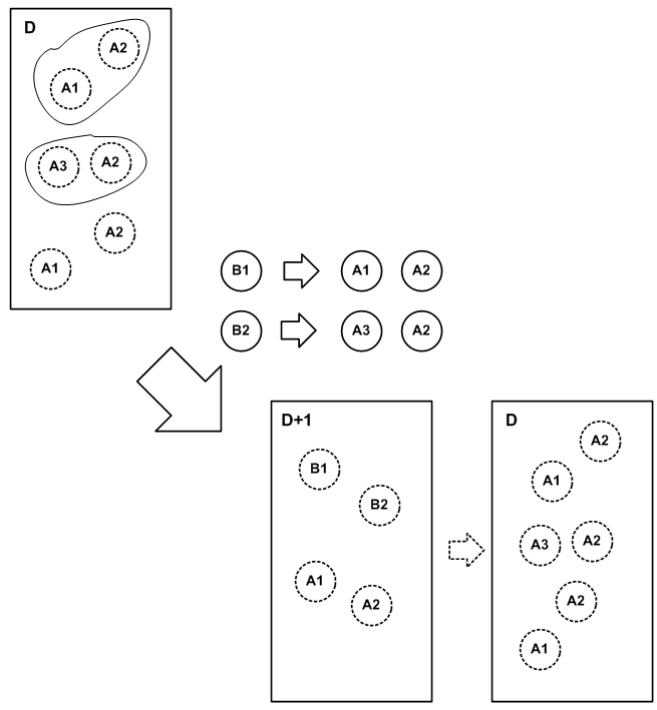
Much better!

linefollower	initializing	paused	running	crash
initialized	true running			
bumped			true crash	
blocked			true paused	
unblocked		true running		

Domains are Hierarchical



Creating Linguistic Abstractions



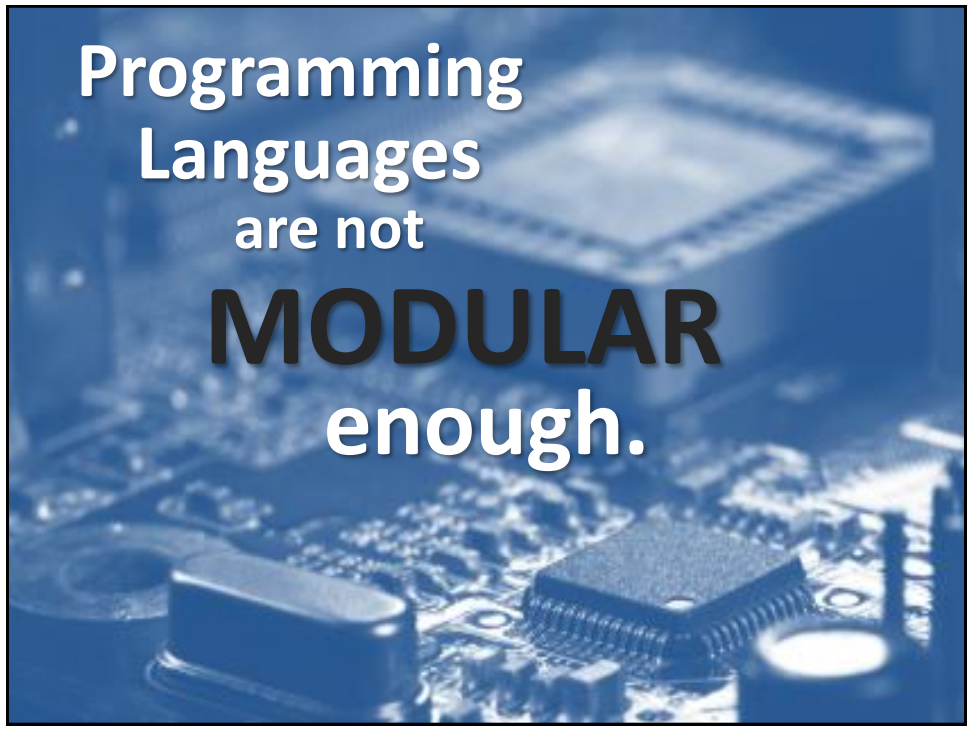
Domain Specific Languages

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.

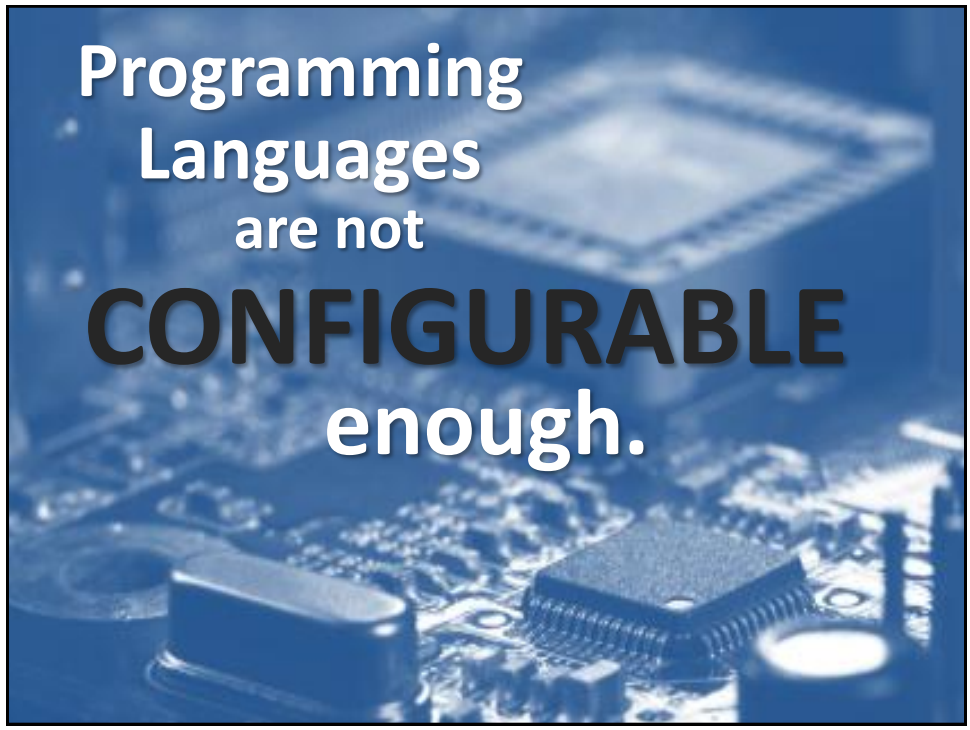
A **good** DSL does **not** require the use of patterns and idioms to express semantically interesting concepts in D. Processing tools do **not** have to do "semantic recovery" on D programs.

4

Modular Languages


Programming
Languages
are not
MODULAR
enough.

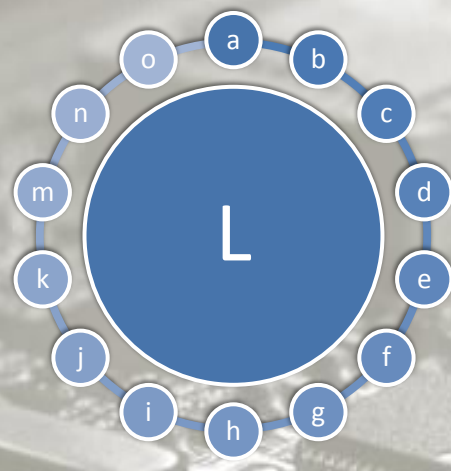
Programming
Languages
are not
COMPOSABLE
enough.


Programming
Languages
are not
CONFIGURABLE
enough.

Programming
Languages
are not
ADAPTABLE
enough.

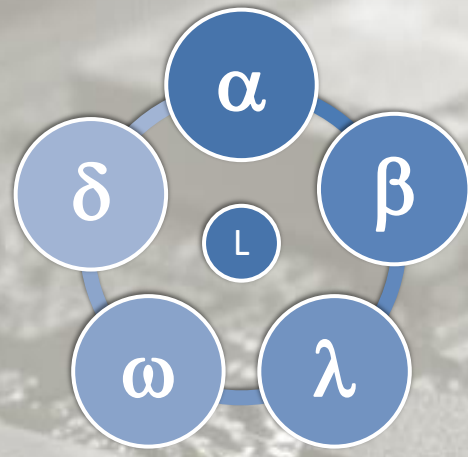
Programming
Language Syntax
is not
FLEXIBLE
enough.

Big Language?



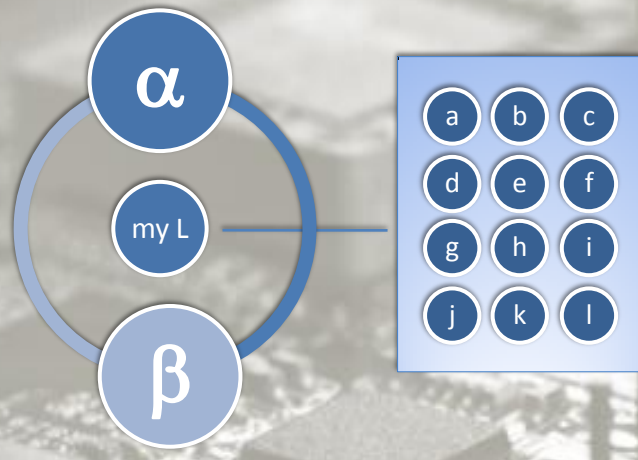
with **many** first class concepts!

Small Language?

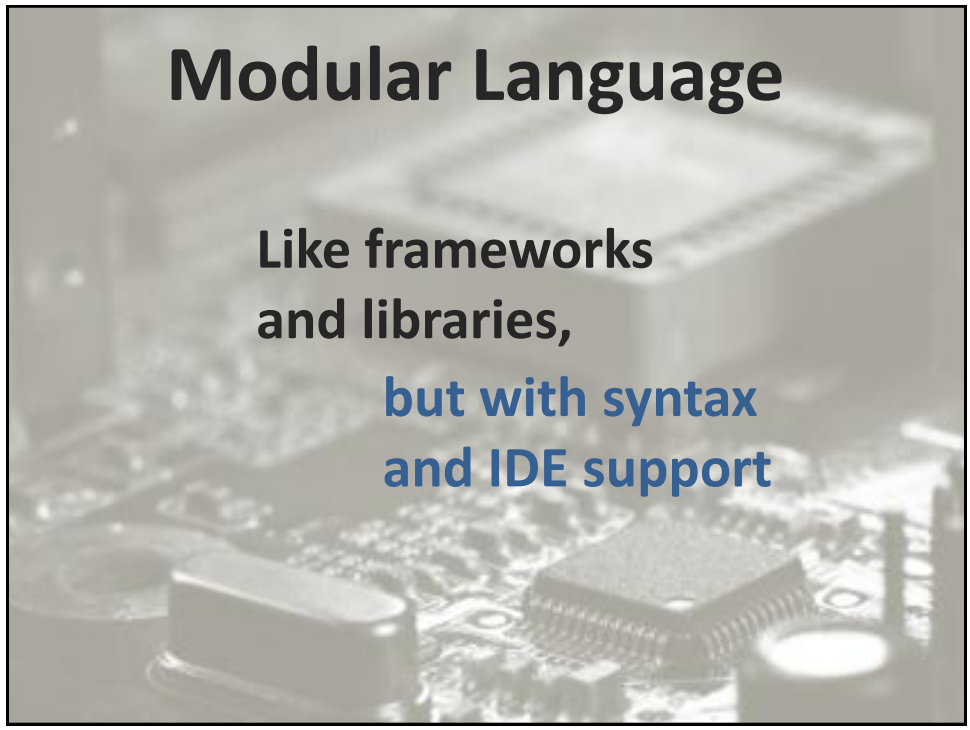


with a few, orthogonal and poweful concepts

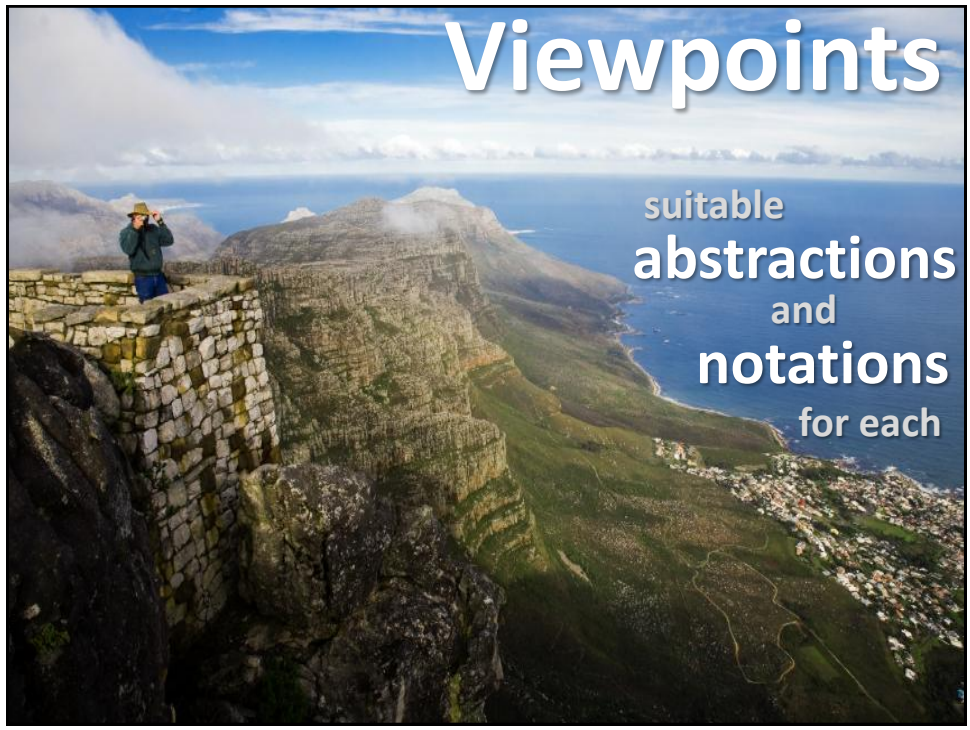
Modular Language

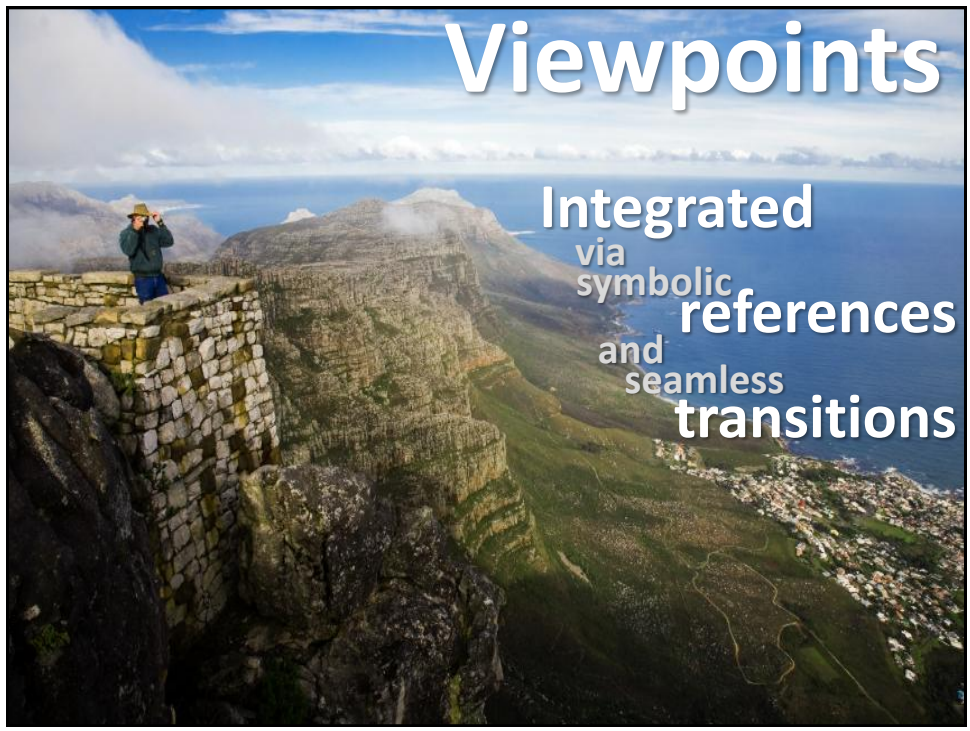


with many optional, composable concepts

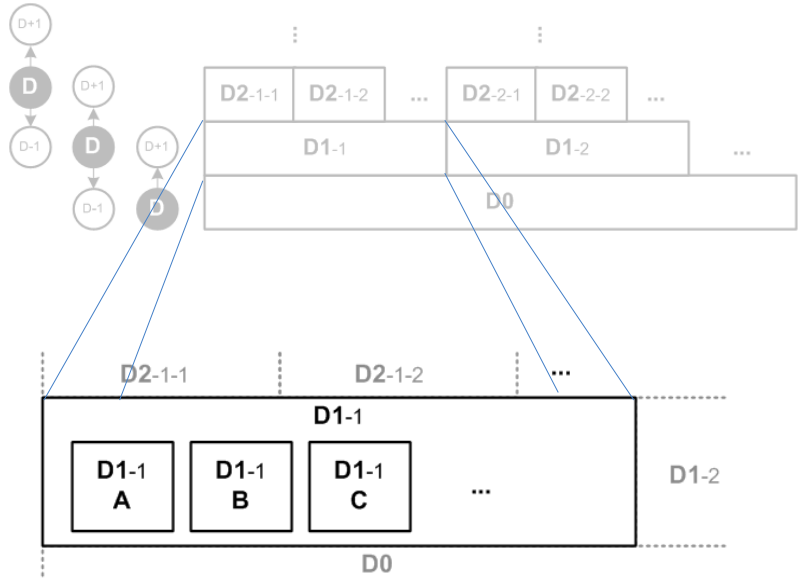


5 Concerns & Viewpoints

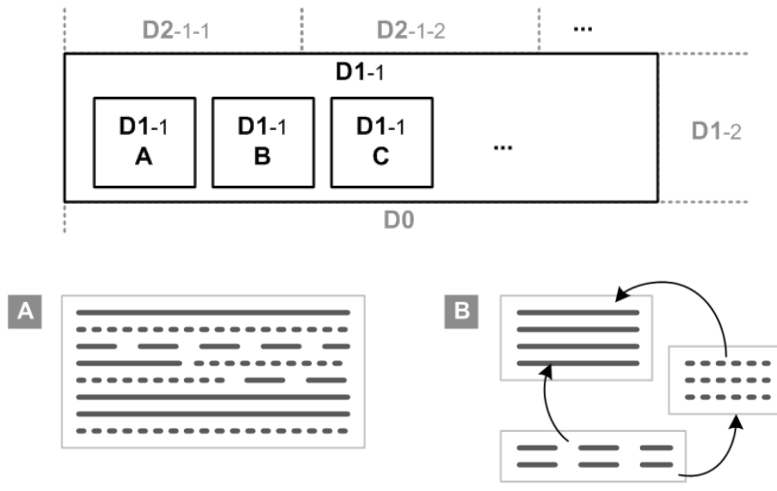

Viewpoints
suitable
abstractions
and
notations
for each


Viewpoints
Integrated
via
symbolic
references
and
seamless
transitions

Several Concerns in a Domain



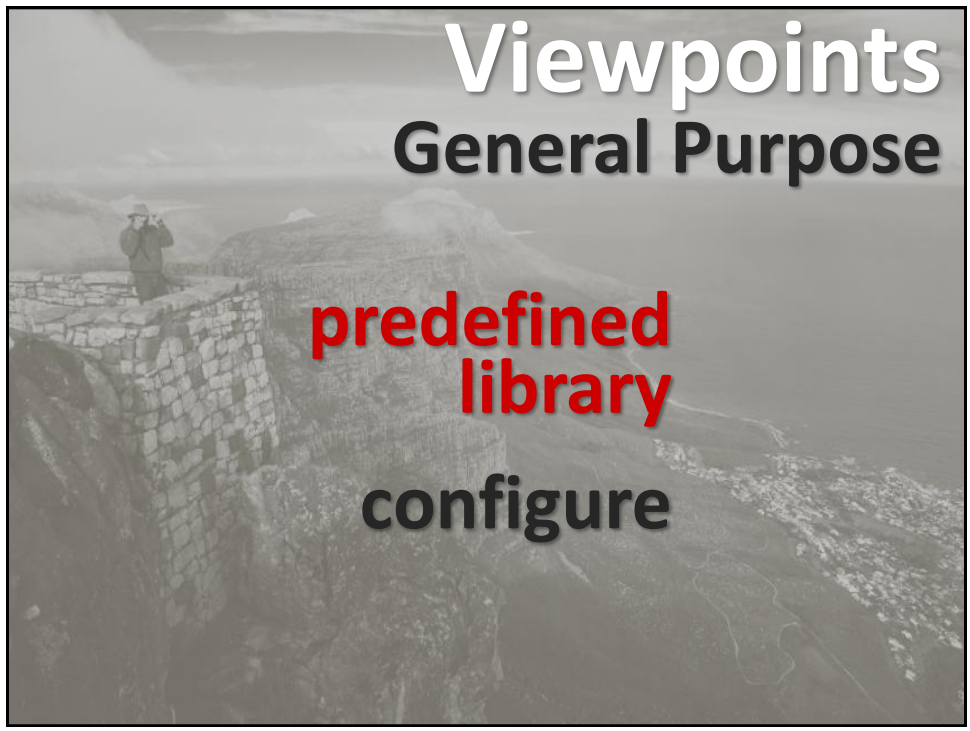
Viewpoints



A: mixed
B: separate Viewpoints

6

Domain-Specificity



Viewpoints
Domain Specific

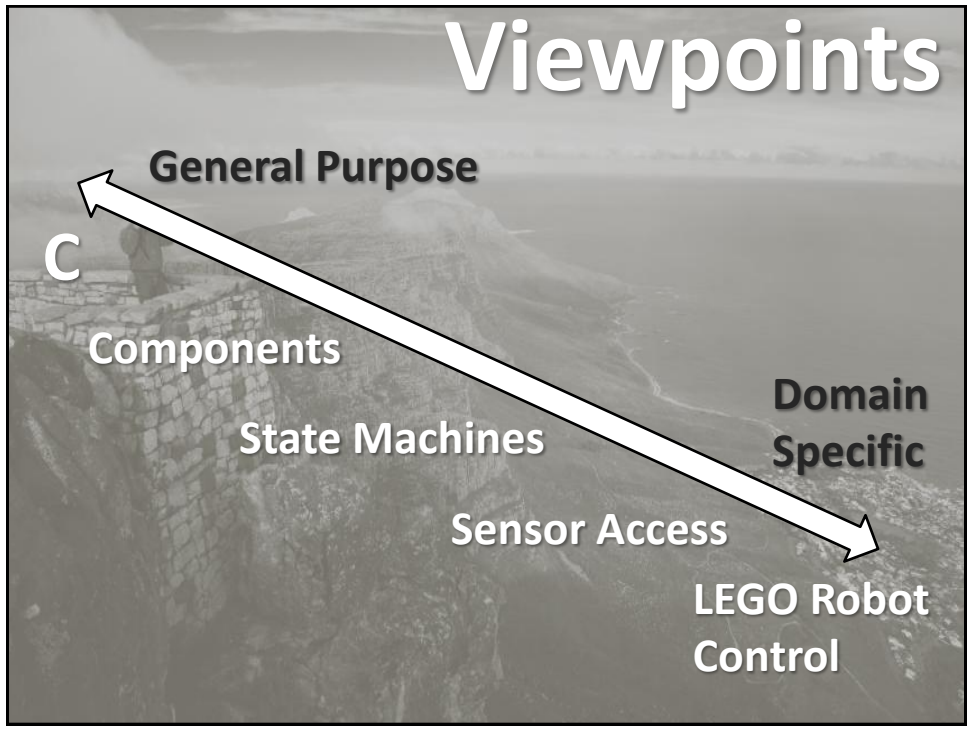
custom
purpose-built

create/include

Viewpoints
Domain Specific

Custom
Notations

real
business
expert integration



7

The LWES Project

Bundesministerium für Bildung und Forschung

http://mbeddr.com

Incremental Extension of C with DSLs for Embedded Systems, integrated with Formal Methods and support for PLE and Requirements Tracing

Bundesministerium für Bildung und Forschung

http://mbeddr.com

LWES
Language Workbenches
for Embedded Systems

Bundesministerium für Bildung und Forschung

http://mbeddr.com

itemis fortiss innovation in software and systems SICK Sensor Intelligence. LEAR CORPORATION

mbeddr.com

Home | Overview | Publications | MPS | Showcase | Code | Team | Search

Feeds: Posts Comments

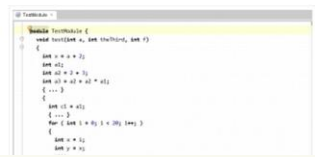
LWES
Language Workbenches
for Embedded Systems
http://mbeddr.com

Bundesministerium für Bildung und Forschung

gefördert durch das Bundesministerium für Bildung und Forschung, Förderkennzeichen 01|S11014

First C Code working
July 17, 2011 by mpscmed

As you may know, our project relies on the idea of extending the C programming language with domain specific extensions. For that to work, we first have to make C available in MPS. While we had done this to some extent in our proof of concept, we are now implementing C much more thoroughly. As you can see in the screenshot below, some essential things are already working.



Archives
July 2011 (3)
June 2011 (2)
January 2011 (2)
July 2010 (1)
June 2010 (2)

Categories
code (3)
demos'n'stuff (4)
dev progress (1)
news (4)
Uncategorized (2)

Pages

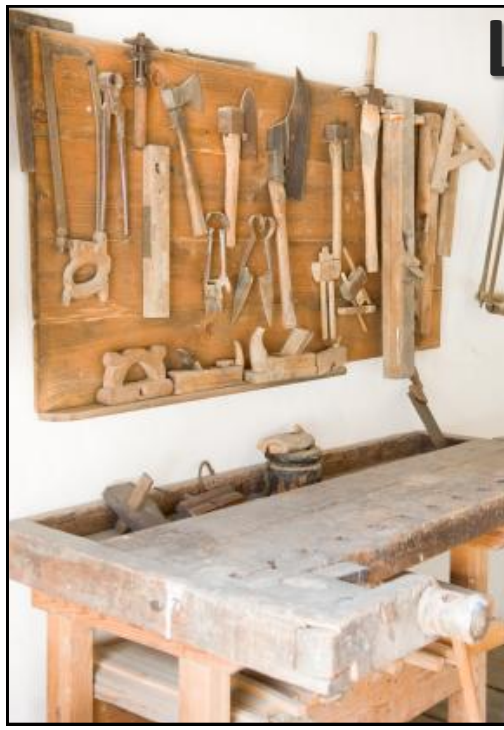
8

Language Workbenches

Language Workbench

(Martin Fowler)

Freely
define
languages and
integrate
them

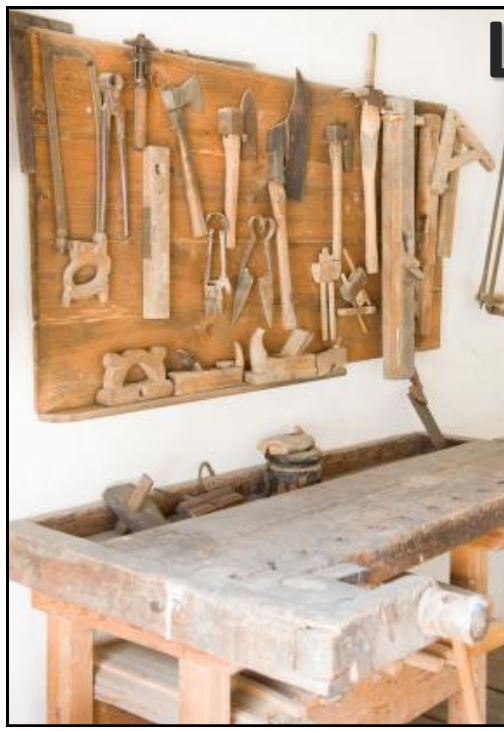


Language Workbench

(Martin Fowler)

use
persistent
abstract
representation

?



Language Workbench

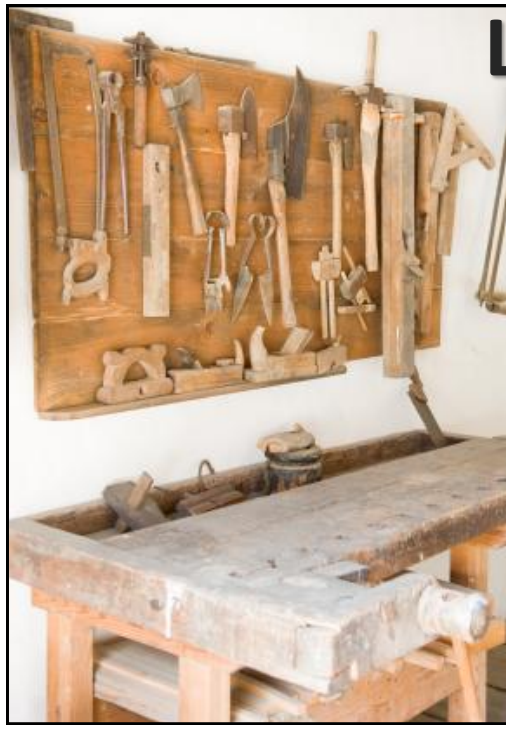
(Martin Fowler)

language ::=
schema
+ editors
+ generators

Language Workbench

(Martin Fowler)

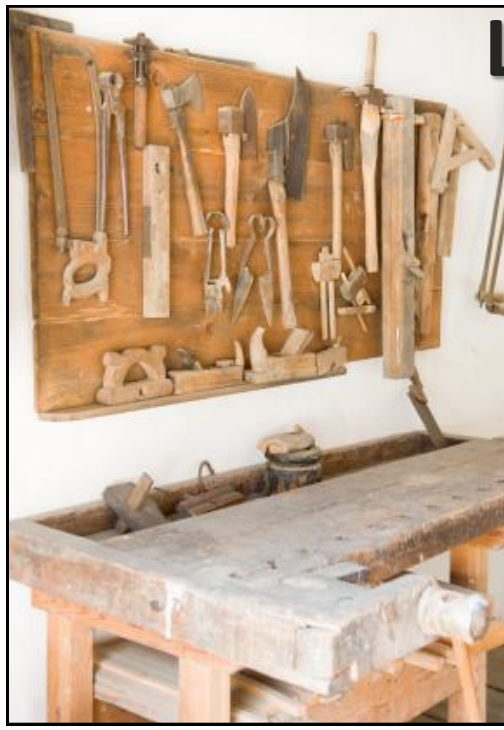
projectional editing ?



Language Workbench

(Martin Fowler)

persist incomplete or contradictory information



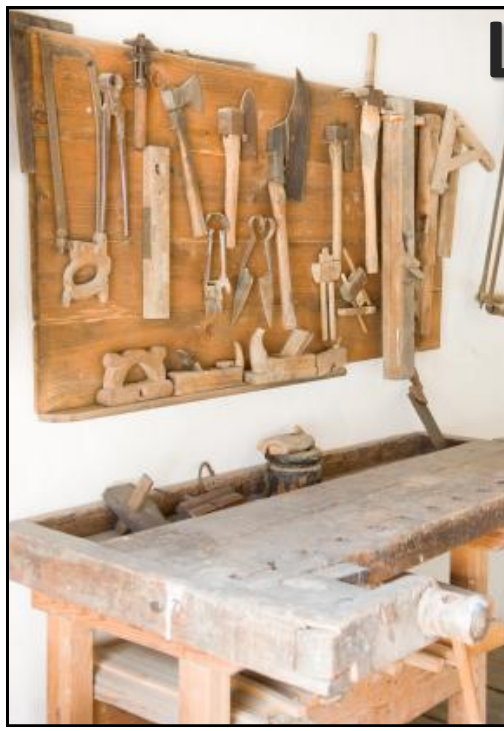
Language Workbench

(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

Syntax primarily textual

with more symbols

think: mathematics

Syntax primarily textual

sometimes box&line style

Syntax primarily textual

sophisticated visualizations

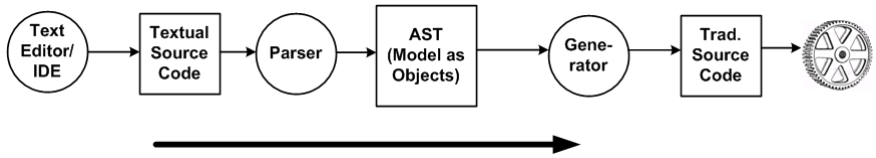
9 Projectional Editing

Parser-based

text

… to tree

… to text



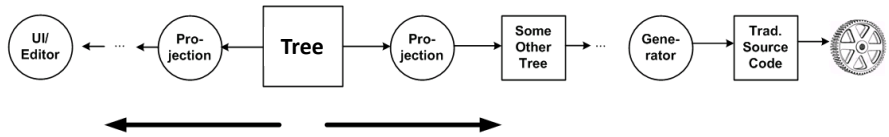
Projectional

tree

… to text-lookalike (editor)

… to other trees … [*]

… to text



Language Composition

There‘s no parsing.

Unique Language Element Identity.

Unlimted 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 arbitraty 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

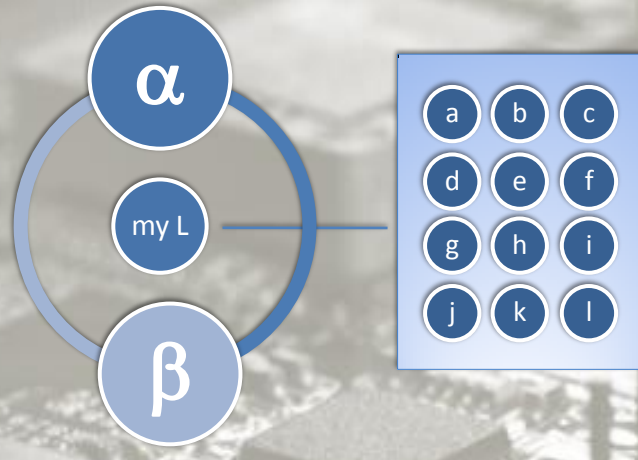
Proprietary Tools

… no standards

… no interop

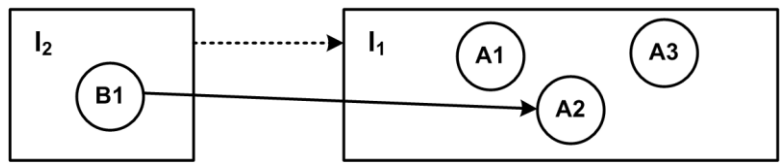
(4) Modular Languages ctd.

Modular Language



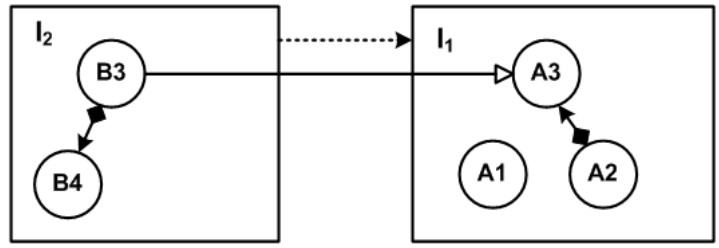
with many optional, composable concepts

Langugae Referencing



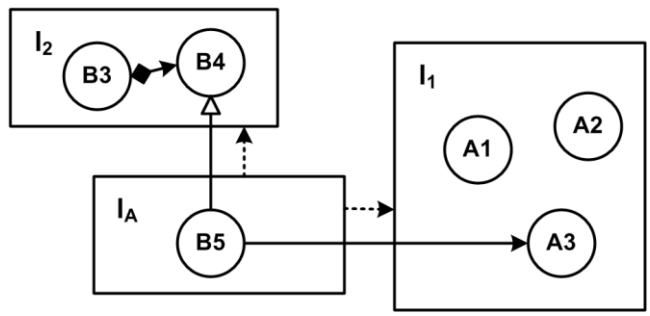
When DSL (for several concerns) are developed from scratch, as a group, then **dependencies between the concerns** can be materialized as dependencies between the languages and the language concepts

Langugae Extension



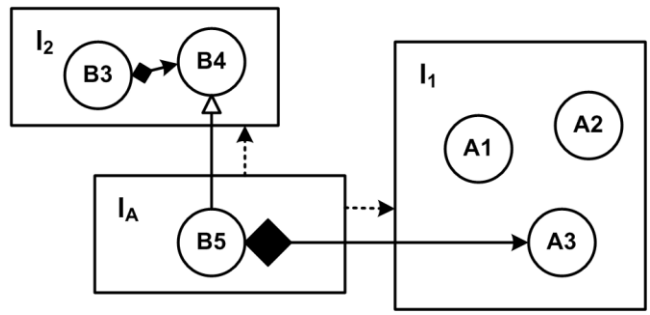
A language B extends another language A if B **contains additional language concepts**. This means that for programs written in B, all concepts from A are available, plus those defined in B.

Langugae Reuse



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.

Langugae Embedding



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

10

JetBrains MPS

JetBRAINS

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. Compososition

Version 2.0.1 today:

Bug fixes. Dozens of them ☺

Open Source (Apache 2.0)

Projectional Editor

Very good at lang. Compososition

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.