Programming Languages

C# Erlang
C++ Python
Ruby Java
Groovy
Fortran C

Programming Languages are not expressive enough.
Programming Languages are not high-level enough.

Programming Languages are not abstract enough.
Programming Languages are not enough.

C#  Erlang
C++  Python
Ruby  Java
Groovy  Fortran

30
Frameworks are not enough. So?
Modeling
... Higher Level
... Domain Specific
Concepts & Notations
... Code Generation
Interpretation
Solves the Problem!

DSL
general purpose

domain specific
tailor made

effective++

specialized, limited

used by experts

together with other specialized tools

But:
Different Worlds
Programming Tools ≠ Modeling Tools

Different Worlds
Modeling Tool ≠ Modeling Tool
Different Worlds
Mix Models and Programs
AST Navigation & Query
Integration of 3GL code
Code Constraints
Domain Specific Notations and Abstractions

Flexible!

Limited!

Frameworks Libraries (Fluent) APIs
Graphical
Textual
Forms
Tables
Flexible!

Limited!
Textual
Trees

Customize
Generator
or
Interpreter
Flexible!

Limited!
Reflection
Meta Programs
Open Compilers
Define custom Query or Navigate or Transform

Flexible!

Limited!

AST APIs
Static Analysis
Regex

Custom Validation or Error Checks

Flexible!

Limited!

IDE plugins
Static Analysis
Open Compilers
Different Representations and Projections

Flexible!

Limited!
Text is Text
Code Folding
Tree Views
Visualizations

Mixing and Composing Languages

Flexible?

Limited!
Python-to-C-like
Internal DSLs
Embed-As-String
Specific: LINQ
Mature!

Brittle!

Scalable
Usable
IDE Support

Modeling Tools...!??
Mature!

Brittle!

for some tools...

Versioning Diff, Merge Branching

Jobs done.

everybody agrees...

Gets some Jobs done.

some people doubt that...

Gets the Job Done!

everybody agrees...
Why
the difference?

History?
Why
the difference?

It is time for ...
... a Different Perspective

Modeling  =>  Programming
Programming  =>  Modeling
modeling == programming

... at different levels of abstraction
... from different viewpoints
... integrated!
modeling == programming

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

modeling == programming

And always: precise and tool processable
Language Workbench
(Martin Fowler)

Freely define languages and integrate them

Language Workbench
(Martin Fowler)

use persistent abstract representation
language ::= 
  schema
+ editors
+ generators

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

think: mathematics
Syntax
primarily
textual
sometimes
box&line style

When a graphical notation is better, you can visualize.
Available Tooling

Eclipse Xtext
Modeling as Programming

... (Mostly) Textual Notations
... Concrete Syntax Storage
... (Fancy) ASCII Editors
... Read-Only Visualizations

Custom Syntax

Graphical Textual Symbolic++
IDE Support

Teamwork Debugging Custom Editors

Complete Symbolic Integration

Goto Def Find Refs Refactoring
Infrastructure Integration

... storage is text
... diff/merge with existing tools
... existing text tools work well!

Language Composition

Grammar composition with traditional parsers is tough!
More advanced parsers currently research
Limited to Unicode

how to handle non-character symbols

Graphics != Text

two worlds...
separate editors
... per syntax/viewpoint
... models can still be ref integrated
Xtext: Specify Grammar

```
<namespace> name="ID" (featureClause=featureClause)* ? ""
  |"unique"="String" ?
  |"xmlns"="Namespace" ?
  | component="Component" ?
  | dtype="Type" ?
  | interface="Interface" ?
  | compositions="Composition" ?

<using> namespace=[[Namespace|qualId]];

<component> (portName=PortName)? "component" name="ID" ?tag="TagOrClass" ?
  (featureClause=featureClause)* ?

<port>
  MessagePort | DataPort;

<port> ProvidedPort | RequiredPort;

<providedPort> "provided" name="ID" ?interface="[Interface]" ?featureClause=featureClause;

<requiredPort>;
```

http://eclipse.org.xtext
Xtext: Generated Editor

Syntax Coloring
Custom Keyword Coloring

Xtext: Generated Editor

Realtime Constraint Validation
Xtext: Generated Editor

Customizable Outlines

Code Folding
Xtext: Generated Editor

- Goto Definition
- Find References
- Cross-File References
- Model as EMF
Building DSLs with Eclipse Xtext
Available Tooling
Jetbrains’ Meta Programming System

Parser-based

text
... to tree
... to text
Projectional

**tree**

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

Programming as Modeling

... (Mostly) Graphical Notations
... Abstract Syntax Storage
... Projecting Editors
... Different editable views for model
Programming as Modeling

... (Mostly) Graphical Any kind of Notations
... Abstract Syntax Storage
... Projecting Editors
... Different editable views for model

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

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
licensed under
Apache 2.0

released in
Q2 2009

currently
1.1 RC1
Build new **standalone** DSLs

Build new **standalone** DSLs
Build DSLs that **reuse** parts of other languages
Build new **standalone** DSLs

Build DSLs that **reuse** parts of other languages

(MPS comes with **BaseLanguage**)  
**extend** base language

---

Build new **standalone** DSLs

Build DSLs that **reuse** parts of other languages

(MPS comes with **BaseLanguage**)  
**extend** base language

build DSLs that **reuse** parts of **BaseLanguage**
Language Extension Example

Old

```java
ReadWriteLock l = ... 
l.readLock().lock();
try {
    //code
} finally {
    l.readLock().unlock();
}
```
Language Extension Example

Old
ReadWriteLock l = ... 
l.readLock().lock();
try {
   //code
} finally {
   l.readLock().unlock();
}

New
ReadWriteLock l = ... 
 lock (l) {
    //code
 }
Structure • Editor • Typesystem • Generator

```
concept LockStatement extends Statement
    implements <name>

    instance can be root: false

    properties:
    << ... >>

    children:
    Statement|Body: 1..* [specializes: <name>]
    Expression|LockExpression: 1..* [specializes: <name>]

    references:
    << ... >>

    concept properties:
    alias = lock

    concept links:
    << ... >>

    concept property declarations:
    << ... >>

    concept link declarations:
    << ... >>
```
Structure  Editor  Typesystem  Generator

```
lockStatement

editor for concept LockStatement
node cell layout:

lock (lockExpression) { lockExpression }

inspected cell layout:
<choose cell model>
```
Structure • Editor • **Typesystem** • Generator

```
lockStatement = \rightarrow typeof_lockStatement

rule typeof_lockStatement {
  applicable for concept \rightarrow LockStatement as lockStatement
  overrides false
  child type restrictions << ... >>
  do {
  }
}
```

[quotedNode] ClassifierType <as name>: [129229555263] in jaxker.typesystem

---

Structure • Editor • **Typesystem** • **Generator**

```
main

conditional root rules:
  << ... >>

mapping rules:
  << ... >>

mapping rules:
  << ... >>

redaction rules:
  \rightarrow reduce_lockStatement
  inheriters false
  condition <always>

abandon roots:
  << ... >>

pre-processing script:
  << ... >>
```
Structure ▶ Editor ◀ Typesystem ◀ Generator

```
public class MyClass extends <name> implements <name> {
  <static initializers>
  <fields>
  <properties>
  <static initializers>
  public MyClass() {
    <no statement>
  }

  public void someMethod() {
    lock (this);  // Copy src[2].lock();
    if (copy[src[2][wait]]) {
      copy[src[2][wait]].unlock();
      return;
    }
    copy[src[2][wait]].unlock();
  }

  <static methods>
  <static inner class>[
}
```

```
Language Extension Example
Result behaves like a native base language construct
Language Extension Example
Result behaves like a native base language construct

```java
package jaxdemo.sandbox.sandbox;
import java.util.concurrent.locks.Lock;
public class DemoClass {
    private Lock lock;
    public DemoClass() {
        try {
            this.getLock().lock();
            SharedResouce.instance().doSomething();
        } finally {
            this.getLock().unlock();
        }
    }
    private Lock getLock() {
        return this.lock;
    }
    private static class unsettledCode
    {
    }
}
```

Language Extension Example
Translated to regular Java code based on the generator

```java
package jaxdemo.sandbox.sandbox;
import java.util.concurrent.locks.Lock;
public class DemoClass {
    private Lock lock;
    public DemoClass() {
        try {
            this.getLock().lock();
            SharedResouce.instance().doSomething();
        } finally {
            this.getLock().unlock();
        }
    }
    private Lock getLock() {
        return this.lock;
    }
    private static class unsettledCode
    {
    }
}
```
DEMO II

Building DSLs with JetBrains MPS