….. the current state of the art of computer programming reflects inadequacies in our stock of paradigms, ….. and in the way our programming languages support, or fail to support, the paradigms of their user communities. (Robert W. Floyd, 1978)

Anforderungen an eine Programmiersprache

Hartmut Krasemann, 2006

aus Erfahrung
in Programmierung und SW-Projekten

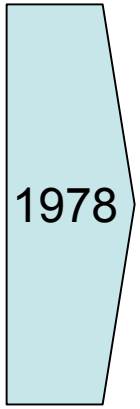
Mit Dank für Ideen und die vielen Diskussionen an
J. Brauer und C. Crasemann

Inhaltsübersicht

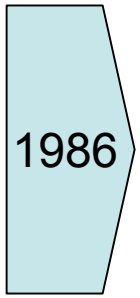
The real romance is out ahead and yet to come. The computer revolution hasn't started yet. Don't be misled by the enormous flow of money into bad defacto standards for unsophisticated buyers using poor adaptations of incomplete ideas.
- Alan Kay

- Paradigmen und Mächtigkeit einer Programmiersprache (Floyd)
- Sprache und Umgebung sind Eins!
- Defizite heutiger Programmiersprachen
  - Paradigmen fehlen
  - Feedback fehlt
  - die Modellierungslücke ist zu groß
- Überbrücken der Modellierungslücke
  - generieren
  - erweitern der OO-Sprache
  - metaprogrammieren
  - XML
  - Language Oriented Programming
- Domain Specific Languages
- Programmieren übermorgen? subtext
- Vision: Die umfassende Language Workbench
- Ausblick
- Lesestoff

Paradigmen einer Programmiersprache (Floyd 1978) bestimmen ihre Mächtigkeit



- Strukturierte Programmierung
- Information Hiding (Levels of Abstraction, abstrakte Datentypen)
- Rekursion und Coroutinen
- Simultaneous Assignment
- Hierarchie von Sprachen
  (Reduce/Lisp, Cobol/Assembler, C++/C, Prolog/St-V, DSL)



- Objekte (nahtlose Erweiterung der Menge der Operatoren)
- Klassen (Objektfabriken) und Vererbung
- Typen (Interfaces)
- Polymorphismus

- Metaprogrammierung: Lisp (Clos), Smalltalk

Sprache und Umgebung sind Eins!

- historisch:
  - Fortran Runtime Bibliothek 1965
  - Ada / APSE (Ada Programming Support Environment) 1980
  - Smalltalk (Virtuelle Maschine + Entwicklungsumgebung) 1980
  - Java / Eclipse und und und ... JVMDI/JVMTI 2000
- Programmierparadigmen (Floyd) erlauben, zu sagen
  - wie das System aussehen soll und was es tun soll
- Paradigmen der EU (Entwicklungsumgebung) bestimmen
  - mit welchen Mitteln es gebaut werden kann
  - und zeigen, wie es aussieht (Feedback)
- Laufzeit-Tools oder APIs sagen
  - was es wie tut (Feedback)

➢ **Programmiersprache, EU und Laufzeitumgebung gehören zusammen!**

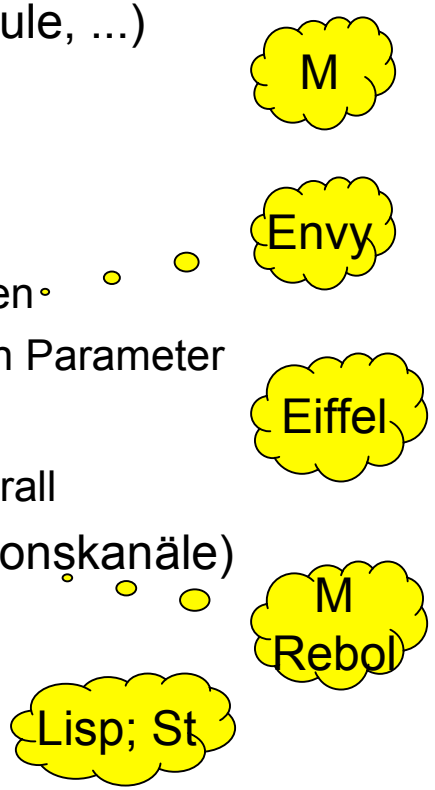
The design of a language should be preceeded by enumeration of those paradigms including a study of the deficiencies in programming … - Floyd

Defizite heutiger Programmiersprachen: Paradigmen fehlen oder sind im Weg

- Patterns
  - Lösungsmuster oberhalb der Programmiersprache
  - sind implizite Defizite der Programmiersprache
- Paradigmen müssen optional sein
- Viele Paradigmen sind nicht universal und manchmal im Weg:
  - Objektorientierung funktioniert nur in einem Adressraum
  - Beispiel Java:
    - Klassen erfordern das Singleton-Pattern
    - Behälter erfordern Iteratoren
    - Kommunikation über Rechnergrenzen erfordert Marshalling/Unmarshalling
    - Statische Typisierung verhindert
      - Flexible Behälter
      - Wiederverwendung

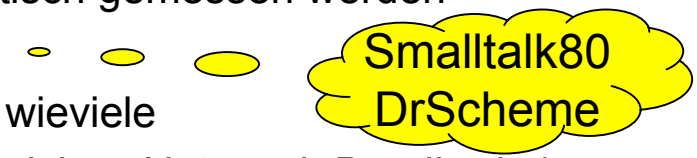
Defizite heutiger Programmiersprachen: Paradigmen fehlen - mal hier - mal da

- Viele Paradigmen fehlen und erfordern expliziten Code
  - rekursive Kapselung oberhalb der Klassenebene (Module, ...)
  - deklarative Constraints (Plausis, GUI Prüfungen)
  - Persistenz
  - Konfiguration und Parametrisierung
    - Selbstbeschreibung in Form einer Stückliste mit Versionen
    - Beschreibung der mit Umwelteigenschaften verbundenen Parameter
  - Kontrakte
    - optional auf Schnittstellen statt statische Typprüfung überall
  - Abstraktion vom Netzwerk ((a)synchrone Kommunikationskanäle)
- Metaprogrammierung ist sehr stiefmütterlich
  - Java Reflection, C++-Templates



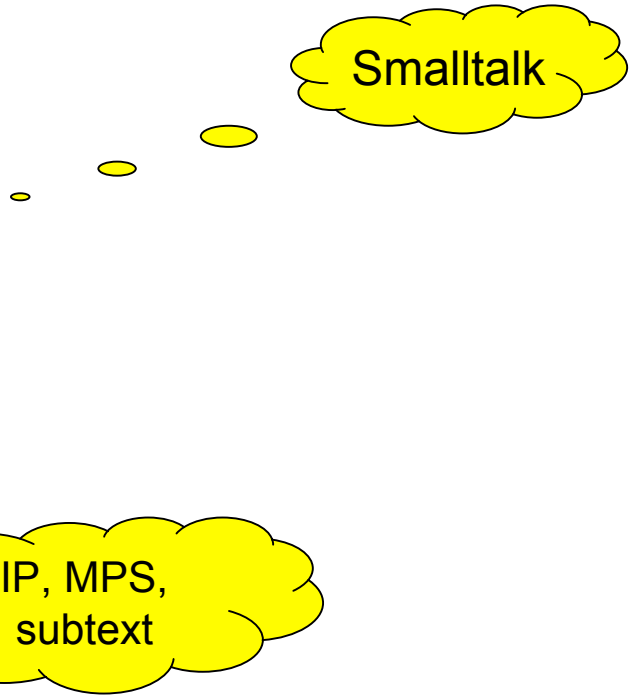
Defizite heutiger Programmiersprachen Feedback fehlt

- Die Programmstruktur ist versteckt
  - Visualisierung der Konstruktion fehlt („*Ansicht einer SW*“)
    - Kopplung, Modularisierung
    - Verteilung
  - Automatische Qualitäts-Metriken fehlen („*Statik einer SW*“)
    - Größe (LOC), Zahl der Teile (Module, Klassen, etc.)
    - Kopplung, Abhängigkeiten
    - Robustheit auf dem Netzwerk (Toleranz gegenüber der Middleware)
    - Robustheit an externen Schnittstellen (Toleranz gegenüber externen Schnittstellen)
- Die Laufzeitumgebung verheimlicht fast alles („*Performanz einer SW*“)
  - Zeitverhalten: Reaktionszeiten könnten automatisch gemessen werden
  - Profiling: welcher Code wird wie oft ausgeführt
  - Fehleranfälligkeit: Fehlerspeicher - wann - wo - wieviele
  - Durchsatz und Ressourcenverbrauch (CPU, Speicher, Netzwerk-Bandbreite)



Defizite heutiger Programmiersprachen Feedback fehlt

- Der Erstellungsprozess ist mühsam
  - edit
  - compile
  - link/build
  - test
- Dynamisch getypte Sprachen sind hier besser
  - edit
  - compile
  - test
- Wozu ein Compiler?
  - edit
  - test



Defizite heutiger Programmiersprachen: Die Modellierungslücke ist zu groß

- Spezifikation ist nur Kommentar (alle Ebenen oberhalb des Codes)
- Semantik ist nicht spezifiziert, Beispiel Stack oder Queue

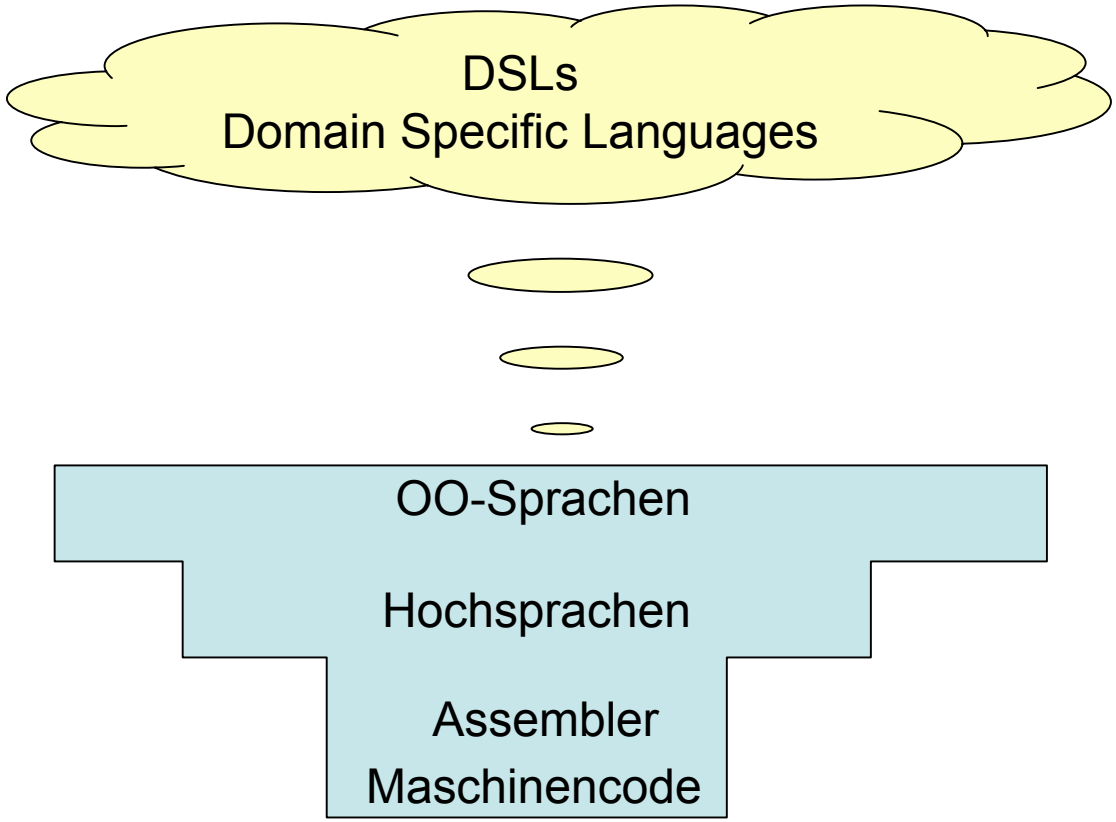
```
public interface Stack {
    public int size() ;
    public boolean isEmpty();
public void push(Object element);
    public Object pop()
        throws EmptyStackException;
}
```

```
public interface Queue {
    public int size() ;
    public boolean isEmpty();
public void put(Object element);
    public Object get()
        throws EmptyQueueException;
}
```

  - wo ist der Unterschied?

- Entwurf sollte Teil des Codes sein
  - explizite Abbildung der Verfeinerung
  - Design-Dokumentation aus dem Code erzeugen
  - Simulation des Systems auf Entwurfsebene

Defizite heutiger Programmiersprachen: DSLs schließen die Modellierungslücke

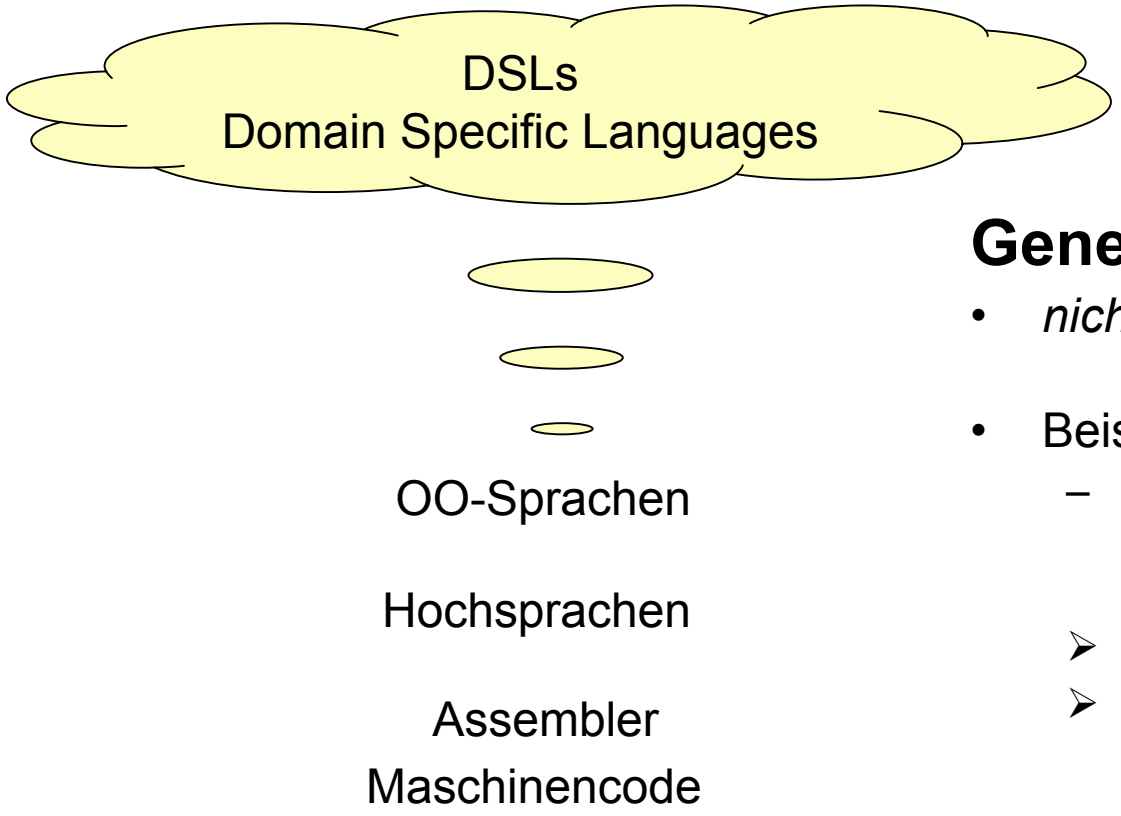


Anwendungssprachen
Anwendungsstrukturen

Jedes Problem braucht seine Sprache

Standard-Algorithmen
Abstrakte Datentypen
Basis-Algorithmen
Basis-Datentypen
Symbolische Adressen
Register

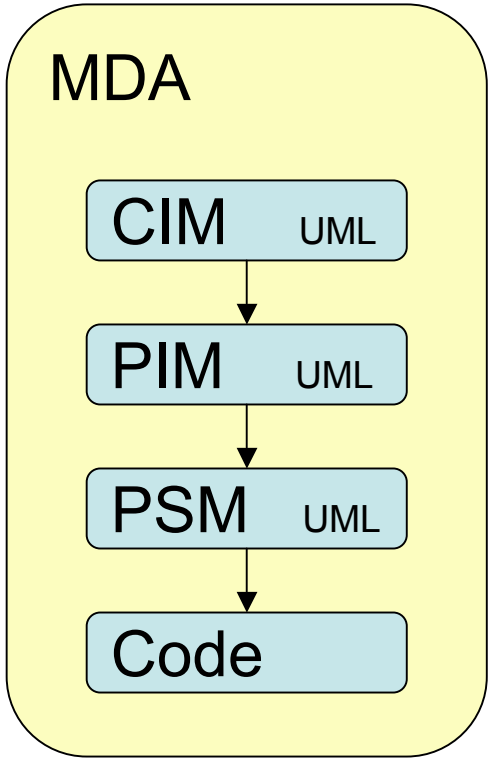
Überbrücken der Modellierungslücke: Generieren



Generatives Programmieren

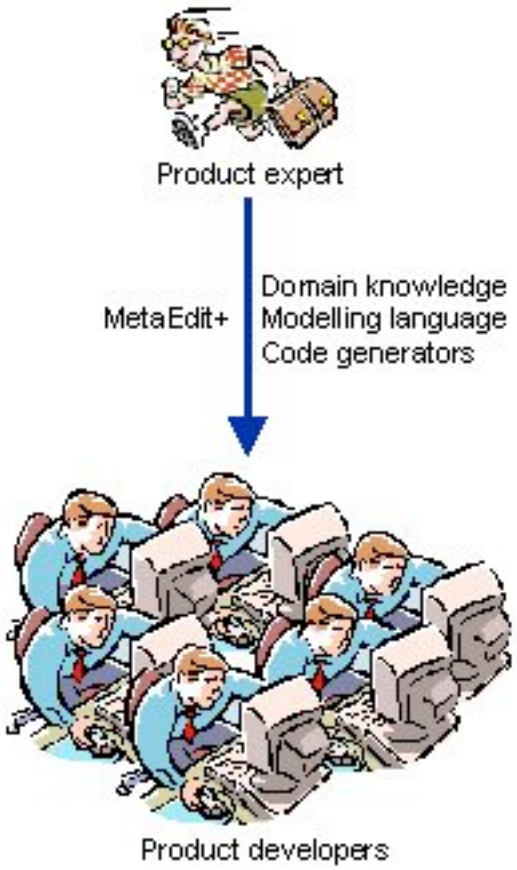
- *nicht reflektive* Sprache
- Beispiele:
  - Historisch
    - C++, ObjectiveC, Eiffel Precompiler
    - St/C
  - ➢ **UML, MDA**
  - ➢ **DSM Domain Specific Modelling**
    - MetaCase

Überbrücken der Modellierungslücke: Generieren: UML und MDA



- UML 2.1 auf dem Weg zur grafischen PS
  - sehr komplexes Metamodell
  - sehr breite Notationen
    (13 Diagrammtypen, >1000 Elemente)
  - die Generatoren
- MDA generieren über Stufen
  - CIM Computation Independent Model
    für die umgangssprachliche Beschreibung
  - PIM Platform Independent Model
    für Geschäftsprozesse
  - PSM Platform Specific Model
    für Architektur, Services
  - Codemodell
- OpenArchitecture Framework

Überbrücken der Modellierungslücke: Generieren: Domain Specific Modelling

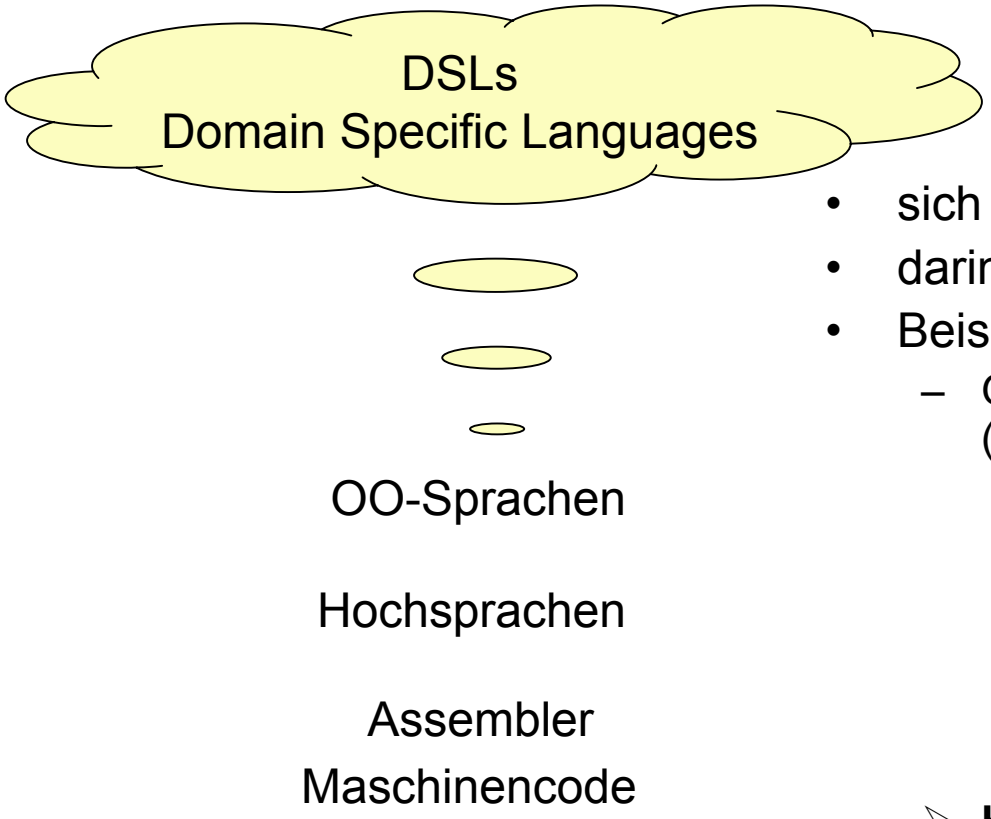


Built with Smalltalk!

- Method Workbench für Architekten/Designer
  - Konzepte definieren
  - Regeln festlegen
  - Symbole entwerfen (die Notation)
  - Generatoren bauen
- MetaEdit+ für Entwickler
  - entwerfen wie mit UML, OMT, Fusion, ...
  - vielseitige Sichten
  - 100% generieren: C++, St, Java, Delphi, SQL
  - Integrieren mit Soap / WebService / .net
  - Dokumente automatisch in Word, HTML
  - Pflege im grafischen Modell !

In Lisp, you don't just write your program down toward the language, you also build the language up toward your program. - Paul Graham

Überbrücken der Modellierungslücke: Erweitern der OO-Sprache



Hierarchy of Languages

- sich eine Domänensprache bauen
- darin das Problem formulieren
- Beispiel:
  - GFD-Projekt vorgestellt im ST-AK (29.4.1996)
    - Modulbrowser, Schnittstellen-Dokumentation
    - automatischer Dokumenter aus Code / ModulSpec / ADvance Diagramme
    - Konfigurationswerkzeug (Stückliste)
    - erweiterter ClassReporter: Metriken, Code-Qualität
    - Knoten-Kanten-Editor
  - **Konto und Überweisung in Smalltalk**

Überbrücken der Modellierungslücke: Erweitern der OO-Sprache: Beispiel St

KontoMüller überweiseEuro: 167 auf: kontoMeier

Konto

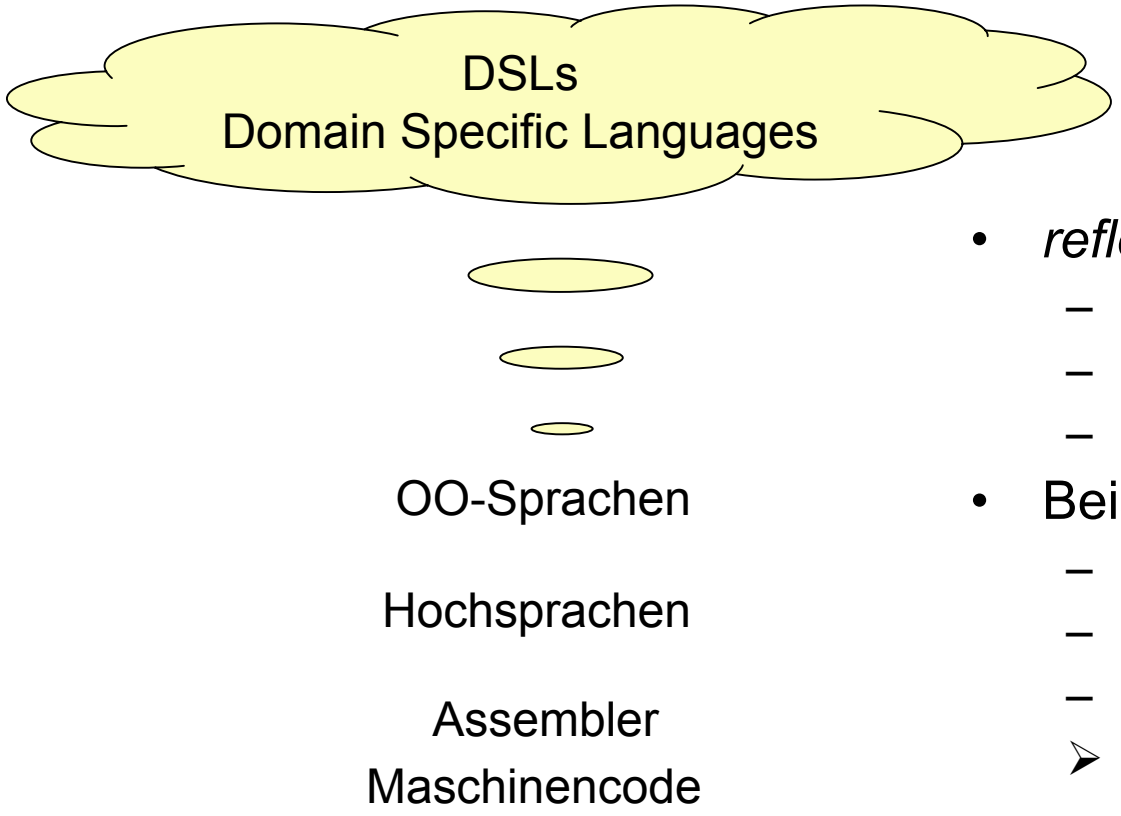
```
==================================private methods
belasteEuro: einBetrag
      kontoStand := kontoStand - einBetrag

schreibeEuroGut: einBetrag
       kontoStand := kontoStand + einBetrag

==================================public methods
überweiseEuro: einBetrag auf: einAnderesKonto
      self belasteEuro: einBetrag)
      einAnderesKonto schreibeEuroGut: einBetrag
```

Wichtigstes Merkmal der Sprache [Rebol] ist das sogenannte Dialecting, was die Möglichkeit bezeichnet, kleine Untersprachen mit eigener Syntax für alle möglichen Anwendungsfälle zu schreiben – www.wikipedia.de

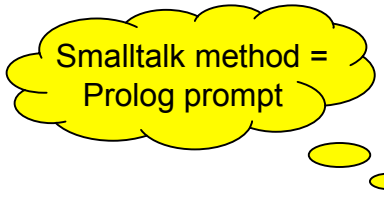
Überbrücken der Modellierungslücke: Metaprogrammieren in der OO-Sprache



Metaprogrammieren

- *reflektive* Sprache
  - neue Syntax
  - neuer Compiler
  - Neuer Debugger
- Beispiele:
  - LISP Makros
  - Rebol-Dialecting
  - Ruby, Python → Logix, ...
  - ➢ **Prolog/V (Port nach Sqeak)**

Überbrücken der Modellierungslücke: Metaprogrammieren: Beispiel Prolog/V



Family new :? grandPa('John', x)

(('Nancy') ('Jack'))

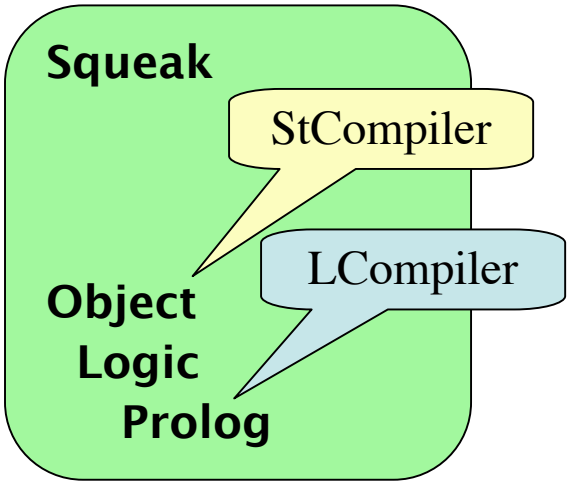
Family

==========rules

```
grandPa(x, y) :- father(x, z), OR(father(z, y), mother(z, y)).
```

==========facts

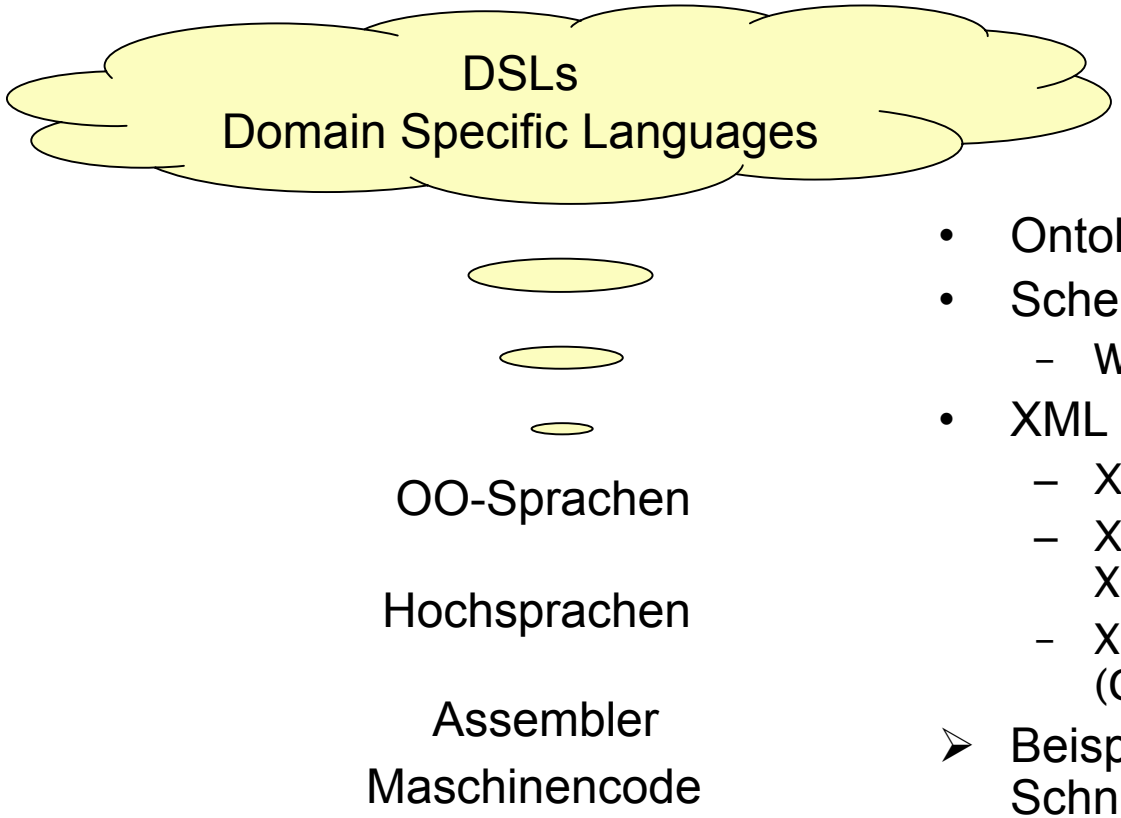
```
father('John', 'Mary').
father('John', 'David').
father('David', 'Jack').
father('Arthur', 'Nancy').
```



Überbrücken der Modellierungslücke: Metaprogrammieren: Squeak Add On‘s

Prototypes	Prototypen statt Klassen
LISP	Lisp Interpreter
Scheme	Scheme
Prolog	Prolog/V
Sorrow	Forth
...	

Überbrücken der Modellierungslücke: XML als Metasprache



XML

- Ontologie in XML-Tags
- Schema definiert Inhalte
  - WXS W3C XML Schema Metamodel
- XML ist eine Meta-Sprache!
  - XSL: Dokumente (und Code!) generieren
  - XSLT Style Sheet Transformation von XML-Dokumenten
  - XUL XML User Interface Language (Gecko – Mozilla)

➢ Beispiel: Schnittstellenspezifikation CTB

Überbrücken der Modellierungslücke: XML als Metasprache: Beispiel: Schnittstellen-Spezifikation CTB

SchemaGeraetetelegramm.xsd
HTMLGeneratorGeraetetelegramm.xsl
BeispielGeraetetelegramm.xml

SchemaJMSMessageBasis.xsd
HTMLGeneratorBasis.xsl
BeispielHTMLGenerator.xsl
BeispielSchemaJMSMessage.xml

Geräte-Telegramme (sind Strings)

==========

jede Schnittstelle hat ihr Schema
jedes Telegramm (Typ) hat sein XML-Dokument
generieren der Dokumentation via XSLT
generieren von Klassen via XSLT
    Zugriffsmethoden auf das Telegramm

JMS-Messages (sind XML-Dokumente)

==========

Ein Basis-Schema
jede JMSMessage
    hat sein Schema (referenziert Basis-Schema)
    hat seinen Generator (importiert Basis-Generator)
generieren der Dokumentation via XSLT
Parsen der JMSMessages mit JAXB

Überbrücken der Modellierungslücke: XML und LISP

XML

```
<container>
      <nummer>HCLU6783544 </nummer>
      <isocode> 42GP</isocode>
</container>
```

LISP

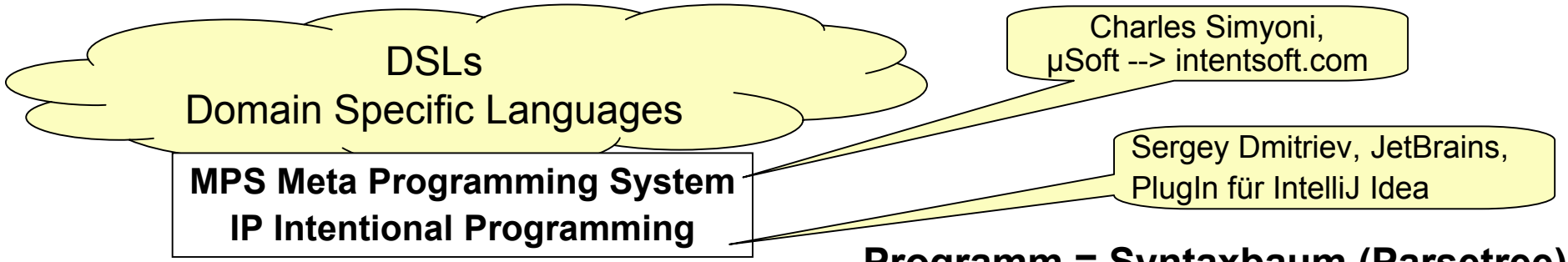
```
(container
      (nummer HCLU6783544)
      (isocode 42GP)
)
```

- Daten = Baum-Struktur
- Code = Daten, Daten = Programm

- Wurzeln in DSSSL

  Document Style Semantics and Specification Language von SGML (Scheme)

Überbrücken der Modellierungslücke: Language Oriented Programming



OO-Sprachen

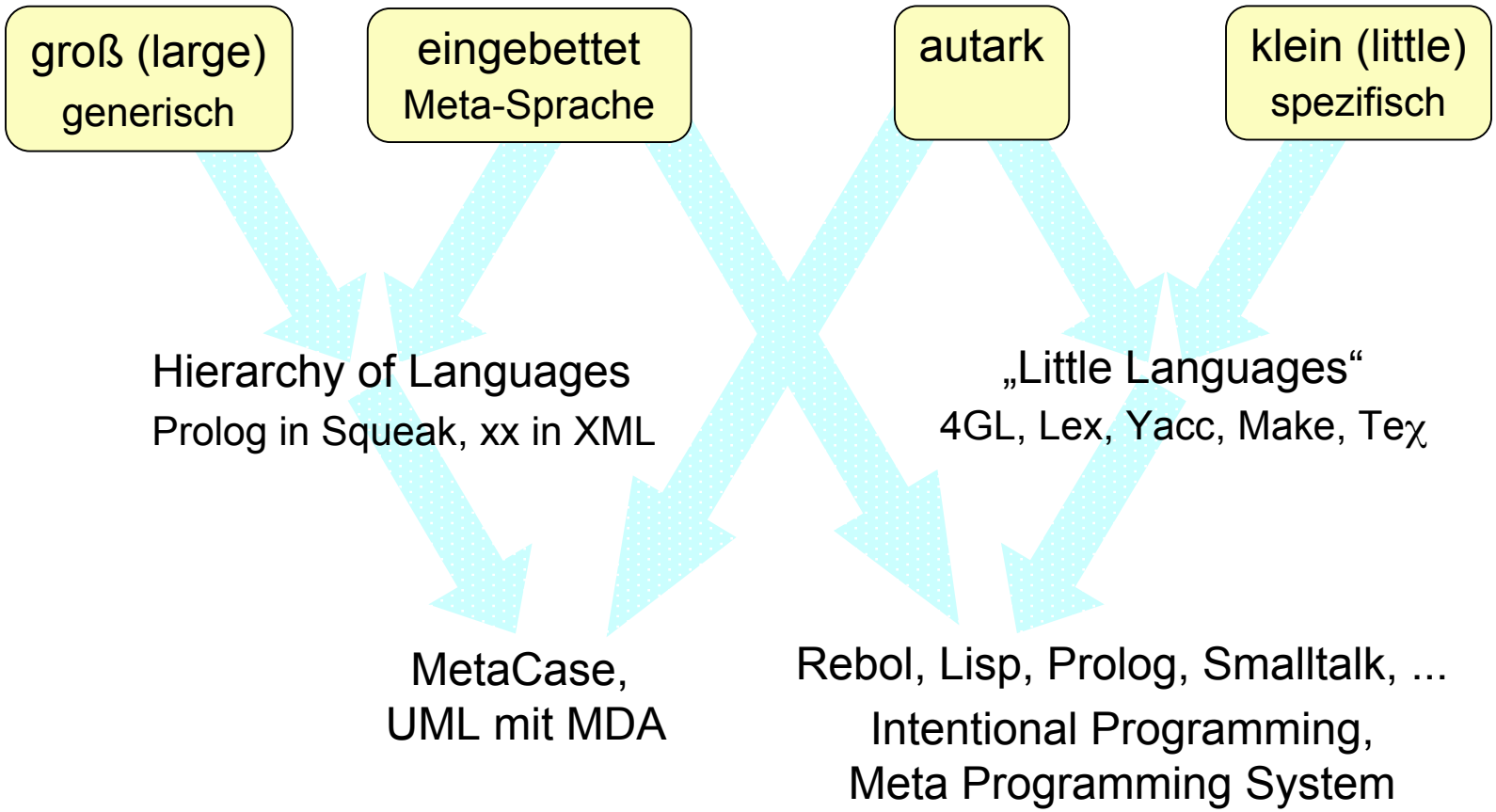
Hochsprachen

Assembler
Maschinencode

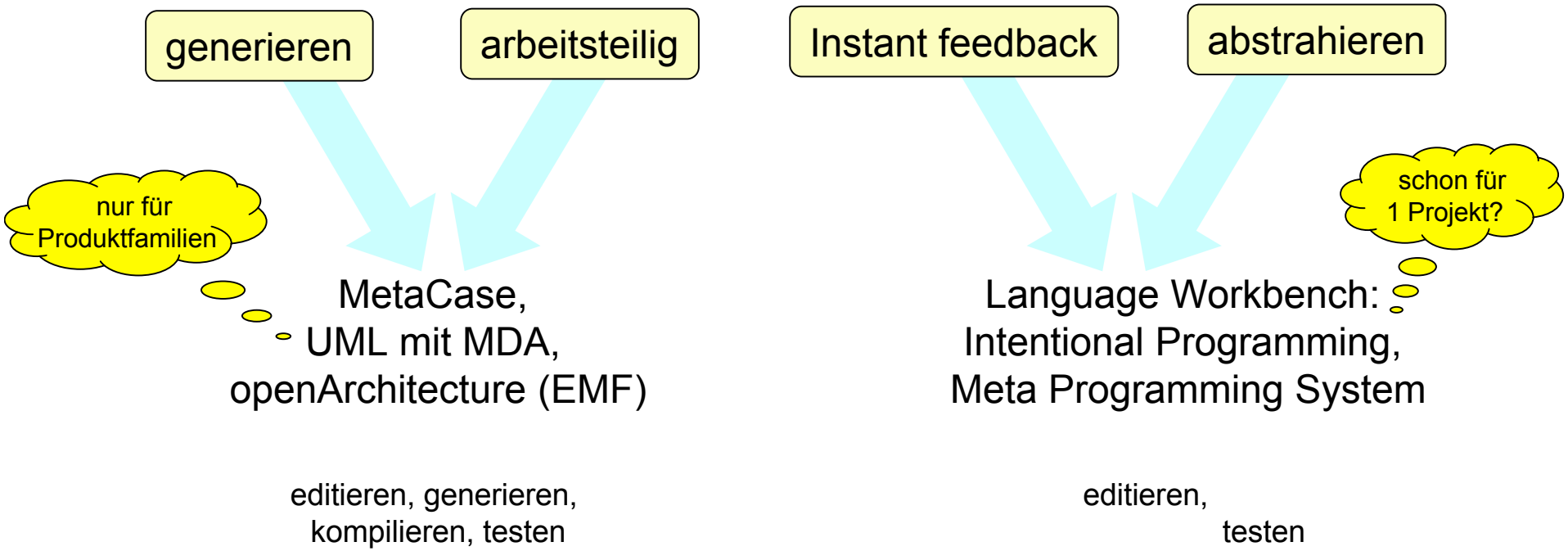
Programm = Syntaxbaum (Parsetree)

- neue Strukturen erscheinen direkt
- Editor ist zwingend
- Benutzer-Notationen
  - mathematische Formeln,
  - Notenschrift,
  - chemische Strukturformeln,
  - Schaltungssymbole,
  - UML-Symbole
  - ...

Domain Specific Languages
Eine Klasifizierung



Domain Specific Languages
Die große Alternative



Rebol, Lisp, Prolog, Smalltalk, ... haben einen Compiler

Programmieren übermorgen
subtext: noch radikaler

- subtext
- Jonathan Edwards, Research Fellow at MIT
- das Programm ist direkt sichtbar (wie bei IP, MPS)
  - programmieren auf dem Syntaxbaum (kein Compiler)
- darüber hinaus:
  - programmieren zur Laufzeit (by example)
  - edit, compile, link und run zur gleichen Zeit

➢ echtes Instant Feedback

- www.subtextual.org/demo1.html

Vision:
Die umfassende Language Workbench

Vertikale DSLs, produktspezifisch, z.B.	**Vers.-Produktm.**	**TV-Editor (CTA)**	**KK-Editor (GFD)**	**Spreadsheet**	
Metaprogrammierung					
GP und Workflow-DSL - Regeln					
Struktur-DSL - Komponenten, Konfiguration (Stückl.), Abhängigkeiten, Umgebung					
Schnittstellen-DSL - Version, Art (Queries, Kdo.s, Notif.), Inhalte (Onthologie, Syntax)					
Constraint-DSL - für Konsistenz von Domäne und User Interface					
Test-DSL - Umweltsimulation, Trigger (Ereignisse), Daten					
Netzwerktransparenz, Ansicht, Statik, Performanz-Sichtbarkeit					

Vision:
Die umfassende Language Workbench

- Programmieren durch Bauen von Anwendungs-DSLs im besten Smalltalk oder Lisp-Stil
  - Projekt zerfällt in zwei Teile
    - Architekten und Designer
      - reden mit dem Kunden
      - bauen neue DSLs
    - Implementierer bauen das System mit den DSLs
      - mit höchster Produktivität
      - Iterationen kosten wenig

Ausblick
oder: Was kommt nach Java?

Ist

es gibt Bewegung in der Programmiersprachen-Front

Revival der Lisp/Smalltalk-Techniken ohne Lisp oder Smalltalk

Spekulation

es ist noch unklar, was sich durchsetzen wird

- alte Sprachen: Lisp, Smalltalk (vermutlich nicht)
- neue Sprachen: Ruby, Python, Rebol, ... (genug drive?)
- UML, Java, XML-Mix: MDA, EMF und openArchitecture (the hard way)
- Language Workbench wie JetBrain's MetaProgrammingSystem (mehr feedback)
- oder sogar ein subtext ? (echtes instant feedback, aber noch nicht fertig)

Notwendigkeit

Thema 1 ist Sichtbarkeit von SW

- *„Ansicht, Statik"*
- *„Performanz, Ressourcenverbrauch"*

Thema 2 ist „Meta"

- Reflektion,
- Domain Specific Languages

Lesestoff

Floyd	The Paradigms of Programming. Commun. ACM 22(8): 455-460(1979)
Java 5.0	java.sun.com/j2se/1.5.0/docs/relnotes/features.html
Rebol	www.rebol.com
Logix	www.livelogix.net
MDA	www.omg.org/mda
openArchitecture	www.voelter.de
Squeak	www, squeak.org
Prolog/V	minnow.cc.gatech.edu/squeak/1000
XML	www.w3.org/XML www.xml-web.de
WXS	www.w3.org/XML/Schema
XSLT	www.w3.org/TR/xslt
XUL	xulalliance.org/

Lesestoff

Czarnecki & Eisenecker
Generative Programming, Addison Wesley, 2000, Kap. 11,
Fowler www.martinfowler.com/articles/languageWorkbench.html
LOP www.onboard.jetbrains.com/is1/articles/04/10/lop/
JetBrains www.jetbrains.net, dort MPS klicken
Intentional Software
intentional.com
DSLs compose.labri.fr/documentation/dsl/
(*Literaturliste*) homepages.cwi.nl/~arie/papers/dslbib/
(*Lisp*) lispm.dyndns.org/news?ID=NEWS-2005-07-08-1
Subtext subtextual.org
und immer wieder, inzwischen ganz hervorragend
Wikipedia www.wikipedia.org, z.B. mit „language oriented programming“
Paul Graham Hackers and Painters. Essays on the Art of Programming, O'Reilly 2004
..