Domain-Specific Languages in Perspective

Jan Heering
Centrum voor Wiskunde en Informatica
Amsterdam
www.cwi.nl/~jan

Joint work with
Marjan Mernik (University of Maribor, Slovenia)
and
Tony Sloane (Macquarie University, Sydney)

April 26, 2007
MoDSE Colloquium
Delft University of Technology
Agenda

- General properties and examples
- Opportunities
  - Software factories
  - Multicore
- Two kinds of application domain?
- DSLs vs. APIs
- DSL development
- Outlook and challenges
General properties

**DSLs**
- Notations and constructs tailored to domain
- Notations very important
  - Use accepted textual/graphical notations if any
- Expressivity gain
- Productivity gain
- Reduced maintenance effort
- Larger group of software developers
  - Domain experts, end-users
Renewed interest

4GL and other DSL legacy code

DSLs nothing new

APT (1957-58) Numerical control of machine tools

Renewed interest

Software factories Multicore revolution
Trade-off generality vs. expressiveness

“Boyle’s law” for DSLs
domain size x expressiveness = constant
(with a large grain of salt)

GPL limiting case
- Large domain
- Low expressiveness
Some widely used DSLs

- BPEL: business processes
- (E)BNF: syntax specification
- Excel: spreadsheets
- HTML: hypertext web pages
- LaTeX: typesetting
- Make: software building
- Maple: computer algebra
- SQL: database queries
- VHDL: hardware design

Application domains
Example: VHDL

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL, IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity clock2 is
    port ( clk : in std_logic;
             res : in std_logic;
             pon, pon_des: out std_logic := '0';
             res_des : out std_logic := '1');
end clock2;

architecture Behavioral of clock2 is
    signal tmp : std_logic_vector(7 downto 0) := x"00";
    signal tmp1 : std_logic_vector(7 downto 0) := x"04";
    constant N : std_logic_vector(7 downto 0) := x"07";
begin
    process(clk,res) [ ... ] end Behavioral;
```
VHDL (cont.)
Example: EBNF

**VHDL**

```vhdl
entity clock2 is
    port ( clk : in std_logic;
            res : in std_logic;
            pon, pon_des: out std_logic := '0';
            res_des : out std_logic := '1');
end clock2;
```

**EBNF rules**

```ebnf
entity_declarations ⇐
    entity identifier is
        [ port ( interface_list ) ; ]
    end [ entity ] [ identifier ] ;

interface_list ⇐
    identifier { , ... } : [ mode ] subtype_indication
        [ := expression ] { ; ... }

mode ⇐ in | out
```
(E)BNF specification

- Defines **grammar** = sentence generator
- **Not** a program: not directly executable
- Compile to **parser** = sentence recognizer

- **(Application) generator** = DSL compiler

---

**Important DSL implementation pattern**
Domain-specificity scale

- EBNF (syntax)
- VHDL (hardware)
- Cobol (business)
- Fortran (scientific)
- C
- C++
- Java
- C#
- UML

Degree of specialization

GPLs
DSL executability scale

UML 2.0 + domain-specific profile ("wide-spectrum")

Domain-specific data representations  EBNF (syntax)  Excel (spread-sheets)

Degree of executability

Not meant to be executable  Highly declarative  Domain-specific programming

Application generators
Traditional DSL roles

- Enablers of the use and reuse of domain knowledge
- Tools for large-scale software development and software generation
- Tools for end-user development and domain-specific human-computer interaction
DSL opportunity #1
Software factories

Software factory = domain-specific IDE

- Domain = system family/product-line
- Large amount of reuse
- Rapid development of system family members
  666 pages
  Few examples
Software Factories approach vs. MDD

- UML 2.0 considered insufficiently precise for describing models, code, and other software artifacts
- Use appropriate DSL(s) instead
  - DSL development becomes part of system family development
  - DSL maintenance/evolution
Software factories (cont.)

Language-specific IDE
- Eclipse
- Visual Studio

Software factory
- Product-line specific + DSL-specific IDE

LDE: IDE for language development
- ASF+SDF, DMS, Eli, Lisa, Sprint, Stratego, more
- DSL Tools in Visual Studio
- Safari for Eclipse (under development)
Software factories (cont.)

Product-line specific
- Feature model
- Product schema (how)
- Component libraries

DSL-specific
- Editing
- Consistency checking
- Analysis
- Code generation
- Transformation
- Testing

Product-specific
- Requirements

Component
- Selection
- Customization
- Adaptation
- Extension
- Assembly

Where does DSL come from?
Software factories (cont.)

Product-line development

Domain expertise

Improve

Use LDE

DSL development

Language design expertise
Software factories (cont.)

Language factory for IDE production

Domain-specific
- Language features
- Constraints
- Fragment libraries

Meta-DSL for language description
- ≈ BNF
- Attribute grammars
- (Rewrite) rules

Fragment
- Selection
- Customization
- Adaptation
- Extension
- Assembly

Powerful LDE

Domain = (family of) languages
In the past
- Don’t parallelize, just wait for faster processor
- Steady increase in clock-rate/heat

2004-2005
- Heat wall reached @ 4 Ghz (P4 Prescott)
- Turn toward multicore
- Parallel programming necessary to exploit multicore

“The free lunch is over” (Herb Sutter)
Parallel programming (cont.)

In the past high-end
- Scientific computing commodity clusters, supercomputers
- Embedded systems
- Graphics (GPUs)
- Servers
- Search engines
  Google: 1000s of PCs

From now on mainstream
- PCs, Xbox 360, PlayStation 3
Parallel programming (cont.)

Moore’s law still holds!
- Steady increase in #cores/chip
- **Scalable** parallel programming
- Platform adaptation

PP very hard
- **Domain-specific** approach
- Identify future needs
  Intel Platform 2015 workload
Parallel programming (cont.)

- **Recognition**
  - What is ...?
  - Learn pattern from data/images

- **Mining**
  - Is it ...?
  - Search dataset for instances of pattern

- **Synthesis**
  - What if ...?
  - Create instances of pattern
Parallel programming (cont.)

Many applications of high-speed RMS

- medicine
- business
- home (photos, videos, music, gaming)
Two kinds of application domain?

(A) Field of knowledge or activity characterized by a set of concepts and terminology understood by practitioners in that field

- computer algebra
- hardware design
- business processes
- syntax specification

Maple

Linear algebra
Calculus
ODEs/PDEs
Fourier series
Numerical methods
Much more and growing

Parsing
Classes of grammars
Other syntactic knowledge
Two kinds of application domain?

(B) Domain = system family = set of software systems exhibiting similar functionality

- Important
  - Software product-line engineering
  - Software factories

- Domain analysis methodologies

- Limited
  - Does not apply to Maple, VHDL, ...

Viewpoint: System family is encoded knowledge about the family’s functionality

- Type B special case of type A
Improper/crosscutting domains

Scripting
- Dynamic/interpreted/interactive use
- Perl, Python are GPLs

Modeling
- UML is a GPL
- Domain-specific modeling languages

Parallel programming
- Domain-specific parallel languages (multicore)
DSLs vs. APIs

GPL + application library ⇒ embedded DSL
• (Mostly) extension of host GPL
• GPL expertise remains valid
• API provides domain-specific vocabulary
• Implementation is free

Special case GPL = UML
• MDD
• UML + domain-specific profile ⇒ DSML

Most DSLs never get beyond application library stage
DSLs vs. APIs (cont.)

So why DSLs?

• Domain-specific notations limited by host GPL
  Java has no user-definable syntax at all
• Domain-specific constructs and abstractions not always suitable for library
  Traversals, transaction processing, error handling, ...
• Domain-specific AVOPT
  Analysis
  Verification
  Optimization
  Parallelization
  Transformation
DSL development is hard

- Requires both domain and language development expertise
- Techniques more varied than for GPLs
- Documentation
  - Tool support
  - Maintenance/evolution
  - Standardization
- Interoperability
DSL development
4GL lessons

1980s-today
- Database applications, report generators, ...
- 4GL vs. Cobol productivity: $-90\%$ to $500\%$

Many did not survive
- Often proprietary
  - Subcritical user community
- Eroding language expertise
- Increasing environmental mismatch
  - External developments: J2EE, .NET, ...

Overall: Very mixed
Identify patterns for each phase
Decision patterns

- Notation
  - Transform visual to textual notation
  - Turn API into DSL
- AVOPT
- Task automation
- System family/product-line
- Data structure representation
- Data structure traversal
- System front-end
- Interaction
- GUI construction
Domain analysis

General

Input
- Technical documents/books
- Domain experts
- Inspection/mining GPL code base

Output
- Terminology/ontology
- Abstract semantics/knowledge

Link with knowledge engineering
Analysis patterns

• Informal
  - Common case

• Formal
  - Use domain analysis methodology
  - Mainly type B domains (system families)

• Extract from code
  - Inspection
  - Mining
    - Clone detection
    - Other patterns

GAL (Video device drivers)
DARE
FAST
FODA
Others

PCSL (Parameter checking)
Analysis patterns (cont.)

Formal domain analysis

<table>
<thead>
<tr>
<th>Scope</th>
<th>Terminology</th>
<th>Domain</th>
<th>Feature model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminology</td>
<td></td>
<td></td>
<td>Commonalities</td>
</tr>
<tr>
<td>Concepts</td>
<td></td>
<td></td>
<td>Variabilities</td>
</tr>
</tbody>
</table>

DSL
FODA: Feature-Oriented Domain Analysis

Feature model
- Feature diagram
  - Hierarchy of mandatory/alternative/optional features
- Semantics of features
- Feature composition constraints
- Motivation for feature

Web service combinators
- Resources
- Browsing behavior
- Services
FODA example (cont.)
FODA example (cont.)

\[
S \leftarrow \text{url(String)} \quad \text{// basic services} \\
| \quad \text{gateway get (String+)} \\
| \quad \text{gateway post (String+)} \\
| \quad \text{index(String, String)} \\
| \quad S_1 \ ? \ S_2 \quad \text{// sequential execution} \\
| \quad S_1 \ '|' \ S_2 \quad \text{// concurrent execution} \\
| \quad \text{timeout(Real, S)} \quad \text{// timeout combinator} \\
| \quad \text{limit(Real, Real, S)} \quad \text{// rate limit combinator} \\
| \quad \text{repeat(S)} \quad \text{// repetition} \\
| \quad \text{stall} \quad \text{// nontermination} \\
| \quad \text{fail} \quad \text{// failure}
\]
Design patterns

- Language exploitation
  - Piggyback
  - Specialization
    - Rare
  - Extension
- Language invention
  - From scratch

- Informal
- Formal

Potential reuse of
- Implementation
- Tools

Potential for embedding

Use LDE

- Users need not be programmers
- Don’t “improve” accepted notations
Implementation patterns

- Interpreter
- Compiler/application generator
- Preprocessor
  - Macro’s
  - Source-to-source trafo
  - Pipeline
  - Lexical processing
- Embedding
- Extensible compiler/interpreter
- COTS
- Hybrid

- C++ template metaprogramming
- MetaBorg
- Basic form: Application library
- Broadway
- CodeBoost
- GHC
- Simplicissimus
Outlook and challenges

General

- DSLs firmly established in important roles
- Large potential
- Final verdict on Software Factories not yet in

Less than 2 pages on DSLs in “Domain-Driven Design” book by Eric Evans (Addison-Wesley, 2004)
Outlook and challenges (cont.)

Software factories

Threats

- DSL development too hard?
  Even with sophisticated tool support
- DSL maintenance/evolution too hard?
  Interaction with DSL code base
- User community too small?
  Eroding expertise
- Unfamiliar/confusing code?
  Proliferation of DSLs
Outlook and challenges (cont.)

Domain analysis

- Analysis of non type B domains
- Links with knowledge engineering
  - Knowledge capture
  - Knowledge representation
  - Ontology development
- Use of analysis results in design phase
Outlook and challenges (cont.)

**Embedding**

- **Needs user-definable syntax in host GPL**
  - Macros and user-definable operator syntax less popular
  - Java has none
  - Preprocessor approach

- **Extensible compilers**
  - Domain-specific optimization rules
  - Domain-specific code generation
  - Complicated
Outlook and challenges (cont.)

**LDEs**

- Integrate decision, analysis, and design support
- Pattern-aware development support
- Meta-DSLs for language definition are hard
- Graphical languages
Literature

Marjan Mernik, Jan Heering & Tony Sloane (2005)
When and how to develop domain-specific languages
Computing Surveys 37 4 316-344

Software Factories Wiley

Herb Sutter (2005)
The free lunch is over
www.gotw.ca/publications/concurrency-ddj.htm

Pradeep Dubey (2005)
A Platform 2015 workload model
softwarecommunity.intel.com/articles/eng/1224.htm