Supporting the DSL Spectrum

David S. Wile

Presented by Michael N. Christoff
October 3, 2012
The author

- David Wile
- Ph.D. in Computer Science at Carnegie-Mellon
- Has published 50 papers in total*
- Published 10 papers while at Teknowledge
- Most recent work focused on adaptive systems and using COTS software (e.g.: Excel, Access, etc...) as platforms for DSLs*

* - Source: ACM Digital Library: Author Profile
What is a DSL?

- DSL = Domain Specific Language
- Focused on solving problems in a specific domain
- Language abstractions taken from the domain
- Many are not Turing complete
Benefits of DSLs

- Concise
- Comprehensible
- Faster to code in ($$)

+ Domain Experts write code

+ Developers Code:
  - DSL infrastructure, DSL Analyzers, DSL GUI front ends
DSL Example: SQL

- SQL Domain = Relational Databases
- Not Turing complete
- Focused on abstracting data from a single type of data structure
DSL ‘Jeopardy’

- People on the left: Explain what the left side code is doing
- People on the right: Explain what the right side code is doing
Java

List heights = new ArrayList();
List weights = new ArrayList();
for(People p : people) {
    if(p.getGender().equals("male") && p.getAge() >= 18) {
        heights.add(p.getHeight());
        weights.add(p.getWeight());
    }
}

SQL

select height, weight
from people
where
gender = 'male'
and
age >= 18
Fair comparison?

- You may not know the Java List interface
- You do not know the class of the people variable
- You may not know Java

- But this is the point!

- Imagine the Java code for a multi table join...
Creating a DSL : An Example

- Domain: Satellites and Ground Controllers (Elements)
- Problem: Specification of satellites and ground stations
- Goal: Reason about linkages
  - When can they be established?
  - Who is linked to who?
  - Are link security settings appropriate?
Building the DSL: What to Specify?

- Structure and Relationships
  - Links between satellites and ground stations
  - Communication protocols between links

- Constraints on interconnections

- Behaviour
Satellite Configuration

‘Elements’

<table>
<thead>
<tr>
<th>Links</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ground Stations</td>
</tr>
<tr>
<td>Static</td>
<td>Regular</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Surveillance Controller</td>
</tr>
</tbody>
</table>
Structure and Relationships

Structure of Elements
- Initial Position
- Protocols
  - Name and Security (public, private, secret)
- Field of View of Ground Stations

Relationships between Elements
- Links
Behaviour and Constraints

Behaviour of Elements
- Hours of Operation
- Velocity

Constraints amongst Elements
- Protocol Security Compatibilities
  - E.g.: - public communication between AUS, EUR, USA ok
  - secret communication *only* allowed between AUS, EUR
Approaches

1. Full language
   ◦ Eg: LEX/YACC, Cornell Program Synthesizer

2. Language Extension
   ◦ Java, Haskell

3. COTS (Common Off The Shelf) Approach
   ◦ Access, PowerPoint DE, XML IDE (XMLSoftware)
Approach 1: Full Language

Design a new language
- Create concrete and abstract syntax from scratch
- Define semantics

Issues
- Requires specialized expertise
- Language design is hard!
- May require new tools (e.g.: domain debuggers)
LEXY/YACC

**source code**

\[ a = b + c * d \]

- Lexical Analyzer
- Lex

**tokens**

\[ id1 = id2 + id3 * id4 \]

- Syntax Analyzer
- Yacc

**syntax tree**

- Code Generator

**generated code**

- load id3
- mul id4
- add id2
- store id1

Image:
- Tom Niemann,
- The Lex & Yacc Tutorial
Lexical Analyzer

- converts strings into **tokens**
- A **token** is a **categorized** string

E.g:

- 123 ≡ Integer Literal
- “hi” ≡ String Literal
- if ≡ Keyword
- i ≡ Identifier
Grammar (informally)
A set of structural rules that govern the composition of sentences in a given language.

Context Free Grammar
- One way to specify such rules
A context free grammar is a set of ‘production rules’

Production rule:

\[ \langle\text{symbol}\rangle \rightarrow \langle\text{expression}\rangle \]

A \langle\text{symbol}\rangle in a string can be replaced with an \langle\text{expression}\rangle

Why is it called ‘Context Free’?

- The \langle\text{symbol}\rangle is replaceable with \langle\text{expression}\rangle regardless of its position in the string
BNF Grammars

- BNF is a notation for describing Context Free Grammars

Example CFG for ‘assignment’ using BNF notation

Productions rules:

- `<assignment> → <id>++; | ++<id>; | <eqassign>;`
- `<eqassign> → <id> <eqsymbol> <eqassign> | <id> <eqsymbol> <id>`
- `<eqsymbol> → = | += | -= | *= | /=`

Examples:

- `i++;`
- `j = k = r;`
Example: Assignment Grammar

<assignment> → <id>++; | ++<id>; | <eqassign>
<eqassign> → <id> <eqsymbol> <eqassign> | <id> <eqsymbol> <id>
<eqsymbol> → = | += | -= | *= | /=

Example: i++;

<assignment>
<i>++; | ++<i>; | <eqassign>
<i>++;
i++;

Syntax Error Checking

- Ensure all code is derived from grammar
LEX/YACC Revisited

source code

\[ a = b + c \times d \]

Lexical Analyzer \(\rightarrow\) Lex \(\leftarrow\) patterns

tokens

\[ id1 = id2 + id3 \times id4 \]

Syntax Analyzer \(\rightarrow\) Yacc \(\leftarrow\) grammar

syntax tree

```
=  
|   
|   
id1  +  
    
|   
|   
id2  *  
    
|   
|   
id3  id4
```

Code Generator

```
load id3
mul id4
add id2
store id1
```

Image:
Tom Niemann,
The Lex & Yacc Tutorial
Example Concrete Syntax: Satellite

- Structure and Behaviour

**USDoD surveillance satellite AF**

*Position: N21 18’ W157 52’*

*Velocity: [.2 hours x 0.3 degrees x .25 miles]/hour*

*Protocol: down: secret P8*

  *up: secret S1*

*Hours: 7:00am to 9:00pm EST*
Example Concrete Syntax: Ground Station

- Structure and Behaviour

USA ground station Oakland

Position: N37 48’ W122 16’

Protocols: up: secret P8

  down: secret S1

out: public S1, secret P8

in: secret S1, public S1

Field of view: 60 degrees
Full Language : LEX/YACC

Relationships

Australia link Alice to Djkarta
USA link FortCollins to Oakland
USA link Oakland to ATT

Constraints

Compatibilities:
public => Australia, EU, USA
secret => Australia, EU
private ATT => {Australia, EU, USA } ATT
Use existing language

- Build ADTs that model constructs in Domain
- Domain programmers interact only with new ADTs
- Minimizes language knowledge required
- Exploits existing tools
**Approach 2: Language Extension**

**Issues**

- May require custom interface to hide language
- May be no one language able to cleanly ‘wrap’ all domain constructs
Language Extension : Java

Example Class Diagram Modeling Domain Elements

Diagram: Class diagram with the following elements:
- Element
- Surveillance-Controller
- Satellite
- GroundStation
- Communications-Satellite
- Surveillance-Satellite
- Protocol
- Velocity
public class Element {
    String name;
    Position pos;
    HashSet<Protocol> inProtocols;
    HashSet<Protocol> outProtocols;
    HashSet<Element> inLinks;
    HashSet<Element> outLinks;

    public void addInLink(Element iEl) {
        inLinks.add(iEl);
    }
    ...
}
public class Protocol
{
    Integer security; // 1=public, 2=private, 3=secret
    String pathType; // S1 or P8
}

public class Velocity
{
    public Time latitudinal;
    public Angle longitudinal;
    public Double altitudinal;
}
Create Structures

```java
inP = new HashSet<Protocol>();
outP = new HashSet<Protocol>();
proto = new Protocol(2, "p8");
inP.add(proto);
outP.add(proto);
el = CommunicationsSatellite(
    "ATT", 3, "ATT", new Position(21, -157), inP, outP);
elements.add(el);
```

Create Links

```java
el.addOutLink(elements.elementNamed("Alice1"));
el.addInLink(elements.elementNamed("Alice1"));
```
Approach 3: COTS

COTS = “Common Off The Shelf”

- E.g.: Access, Excel, PowerPoint, XMLSoftware
- Try to build DSL on top of COTS platform
Approach 3: COTS

Issues

- Highly dependent on level of customization afforded
- May be more restrictive than other options
COTS Approach: Access

- Tables model Domain constructs (Elements)
- Element names used as pkeys for table rows
Approach 3: COTS

Access vs. Java: CommunicationsSatellite ‘class’ as a Table

- **Java**: Element and Satellite are superclasses of ComSat
- **Access**: All collapsed into ComSat Table
- Also, **Position** class collapsed into ComSat
  - *Simplifies data entry in Access GUI*

![Diagram of ComSat Table]

**Position attributes**
Approach 3: COTS

Java vs Access: Protocols ‘class’ as a Table

- **Java**: Protocol objects contained inside Element objects
- **Access**: Separate table relating Protocols to Elements

Note use of Access GUI
Evaluation of Approaches

Background

- Paper published December 2001
- Eclipse v1.0 release November 2001*
- Open IDE Platforms not yet widespread

* - source: eclipse.org
## Evaluation of Approaches

<table>
<thead>
<tr>
<th></th>
<th>LEX/YACC</th>
<th>Java</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure</td>
<td>↑↑</td>
<td>↑</td>
<td>↑↑</td>
</tr>
<tr>
<td>Relationships</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Constraints</td>
<td>↑↑</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Behaviour</td>
<td>↑↑</td>
<td>—</td>
<td>↓</td>
</tr>
<tr>
<td>Edit</td>
<td>—</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>TypeCheck</td>
<td>↓</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Analyze</td>
<td>↓</td>
<td>↓↓</td>
<td>—</td>
</tr>
<tr>
<td>Simulate</td>
<td>↓</td>
<td>—</td>
<td>↓</td>
</tr>
<tr>
<td>Unparse</td>
<td>↓↓</td>
<td>↓↓</td>
<td>—</td>
</tr>
<tr>
<td>Persistence</td>
<td>↓</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Versions</td>
<td>↓</td>
<td>↓</td>
<td>↓↓</td>
</tr>
</tbody>
</table>

**Legend:**
- DSL-Specific
- GPL-Support
- Pragmatic Support

**Support Levels:**
- 4: Strong Support
- 3: Medium Support
- 2: Weak Support
- 1: No Support
**DSL Specific Criteria**

Structure, Relationships, Constraints, Behaviour

- All performed well, however...
- Behaviour: Access has none!
GPL Specific

Editing
- LEX/YACC : custom coded emacs plugin suggested
- Java: Powerful IDEs
- Access: Comes with editor

Type Checking
- Java: Enforces class accessor syntax
- Access: Enforces database schema
- LEX/YACC: Requires difficult-to-code custom support
Simulation

E.g.: How many times will satellite ATT be in range of both satellite AF and station Alice next week?

- Java: Easy to run but requires developer
- Access: No behaviour but data easy to access
- LEX/YACC: Provides no built-in support
Conclusions

- High number of variables to consider
- No right solution
- Depends on problem domain, resources, current environment
Thanks!