Domain Specific Langauges

it's all about the layers

Thoughts and Ideas

the power of software is the power of thoughts and ideas

“The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures.”—Fred Brooks

http://twitter.com/hartsock

Agenda

- Concept: Domain Specific Language
- Concept: Accidental Complexity
- Examples of DSL
- Implementing a DSL
  - Live Coding
- Implementing an External DSL
  - Live Coding
- Architectural Implications
- Reading List
- Thanks to the Community

What is a Domain Specific Language?

... more importantly, why do I care?

An *artfully* limited programming language

(as opposed to a General Purpose Programming Language or GP)

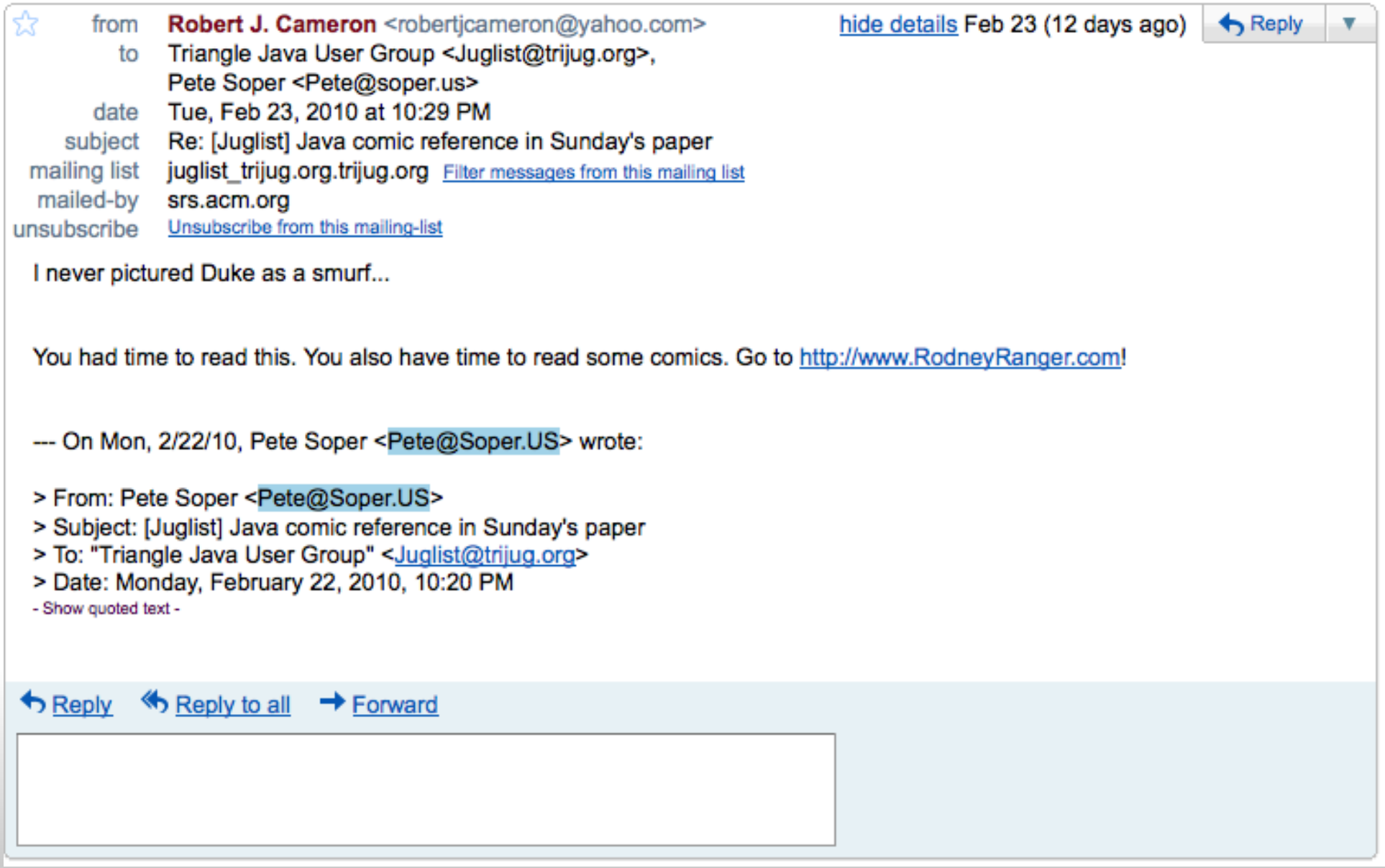
some practical examples and some background...


Google
site:ibiblio.org Groovy -Java
Search

"xml is a great step forward to nothing" - @HamletDRc #2gx OSGi + Groovy

9:41 AM Oct 22nd from twidroid Reply

aalmiray
Andres Almiray



from **Robert J. Cameron** <robertjcameron@yahoo.com> hide details Feb 23 (12 days ago) Reply

to Triangle Java User Group <Juglist@trijug.org>,
Pete Soper <Pete@soper.us>

date Tue, Feb 23, 2010 at 10:29 PM

subject Re: [Juglist] Java comic reference in Sunday's paper

mailing list juglist_trijug.org.trijug.org Filter messages from this mailing list

mailed-by srs.acm.org

unsubscribe Unsubscribe from this mailing-list

I never pictured Duke as a smurf...

You had time to read this. You also have time to read some comics. Go to http://www.RodneyRanger.com!

--- On Mon, 2/22/10, Pete Soper <Pete@Soper.US> wrote:

> From: Pete Soper <Pete@Soper.US>
> Subject: [Juglist] Java comic reference in Sunday's paper
> To: "Triangle Java User Group" <Juglist@trijug.org>
> Date: Monday, February 22, 2010, 10:20 PM

- Show quoted text -

Reply Reply to all Forward

* look at the subject line, quoted text

What do these have in common?
shell scripts, HTML, email & SQL

In a sea of computing...

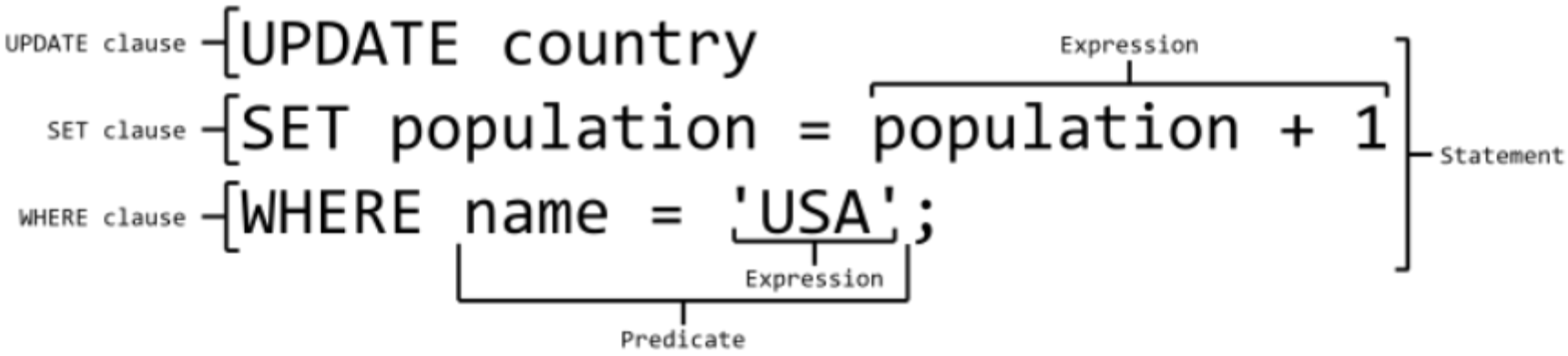
- many programs today are written to be used by other programs never to interface with a human user directly.
  - exposed as a service
  - as a programming library
- Users today write things such as
  - mail filters
  - spreadsheet macros
  - mailing lists

Does Turing completeness help a mail filter?

Reducing Accidental Complexity

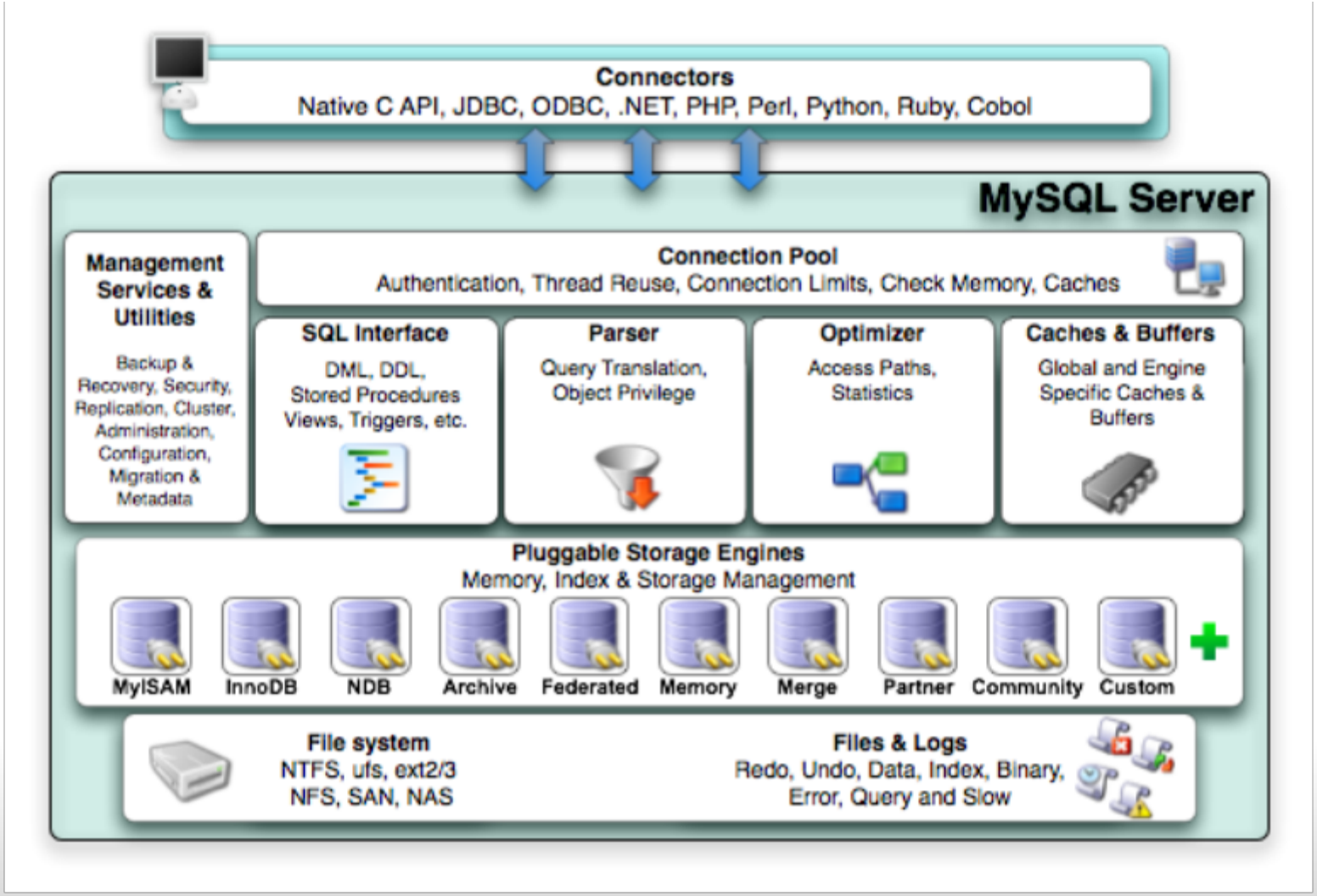
SQL

An example:
SQL (Structured Query Language)



What can't we do? What can we do? What don't we think about? Why is that important? What does the user/DBA care about?

An example: **MySQL**



The DBA *gets* to ignore the details "below" the DSL.

Who sees the SQL?

Who uses SQL

- a specialist tier of programmers called DBAs
- specialist analysts who deal with reporting on data
- tools like Hibernate that provide Object Relational Mapping.

And they lived happily ever after...

... not quite.

Accidental Complexity

... a failure in new light.

The Vietnam of Computer Science

"Object-Relational mapping is the Vietnam of our industry"
-- Ted Neward

ORM: a study of accidental complexity...

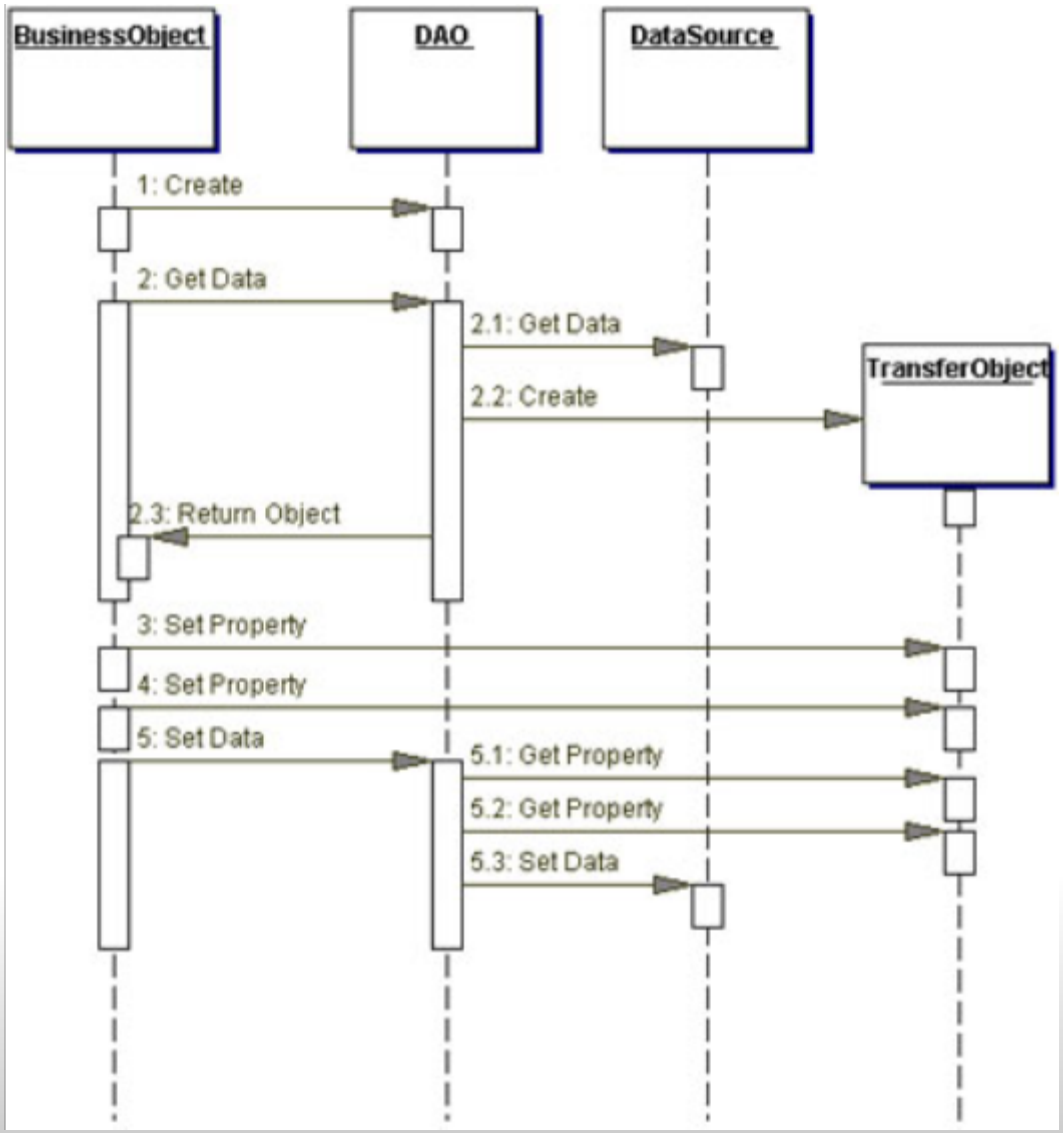
```
{
  {PERSON SSN='123-45-6789' Name='Catherine Kennedy' City='Seattle'},
  {PERSON SSN='321-54-9876' Name='Charlotte Neward' City='Redmond'},
  {PERSON SSN='213-45-6978' Name='Cathi Gero' City='Redmond'}
}
```

"...there are distinct differences between how the relational world and object world view the "proper" design of a system, and more will become apparent as time progresses."

ORM is accidental complexity

it has virtually nothing to do with
solving the problem an application
has been set to.

a GP talking to a DSL



- Static typing (not in diagram)
- create dependency
- Map Relational (SQL) to Object (Java class)
- Network serialization could be there so we have to implement it
- Honor the encapsulations so use these getters and setters.
- send back the changes
- map from object back to tables

"The cost of flexibility is complexity." - Martin Fowler

The DAO in Java

```
// Interface that all CustomerDAOs must support
public interface CustomerDAO {
public int insertCustomer(...);
public boolean deleteCustomer(...);
public Customer findCustomer(...);
public boolean updateCustomer(...);
public RowSet selectCustomersRS(...);
public Collection selectCustomersTO(...);
...
}
// code for CustomerDAOFactory
// and CustomerDAOImpl omitted for brevity.
```

...I only had room for the interface...

Domain Specific Language

GORM:

Grails' Object Relational Mapping

GORM: Domain Specific Language

```
class Customer {
  String firstName
  String lastName
  String email

  static constraints = {
     email(email:true, unique:true)
  }
  static mapping = {
    columns {
      firstName(column:'given_name')
    }
  }
} // remember the Vietnam of Computer Science? Where is it?
```

DSL != 4GL

not generations but layers of abstraction

* these architectural abstractions apply to NoSQL or any system with a complex API.

DSL Builders

- Builders create XML documents
  - For example: GORM's constraints & mapping
  - May include executable code
- Smart configurations adapt to code run-time
- Invoke API to configure systems
  - even when no configuration system existed previously
  - example: Swing Builder

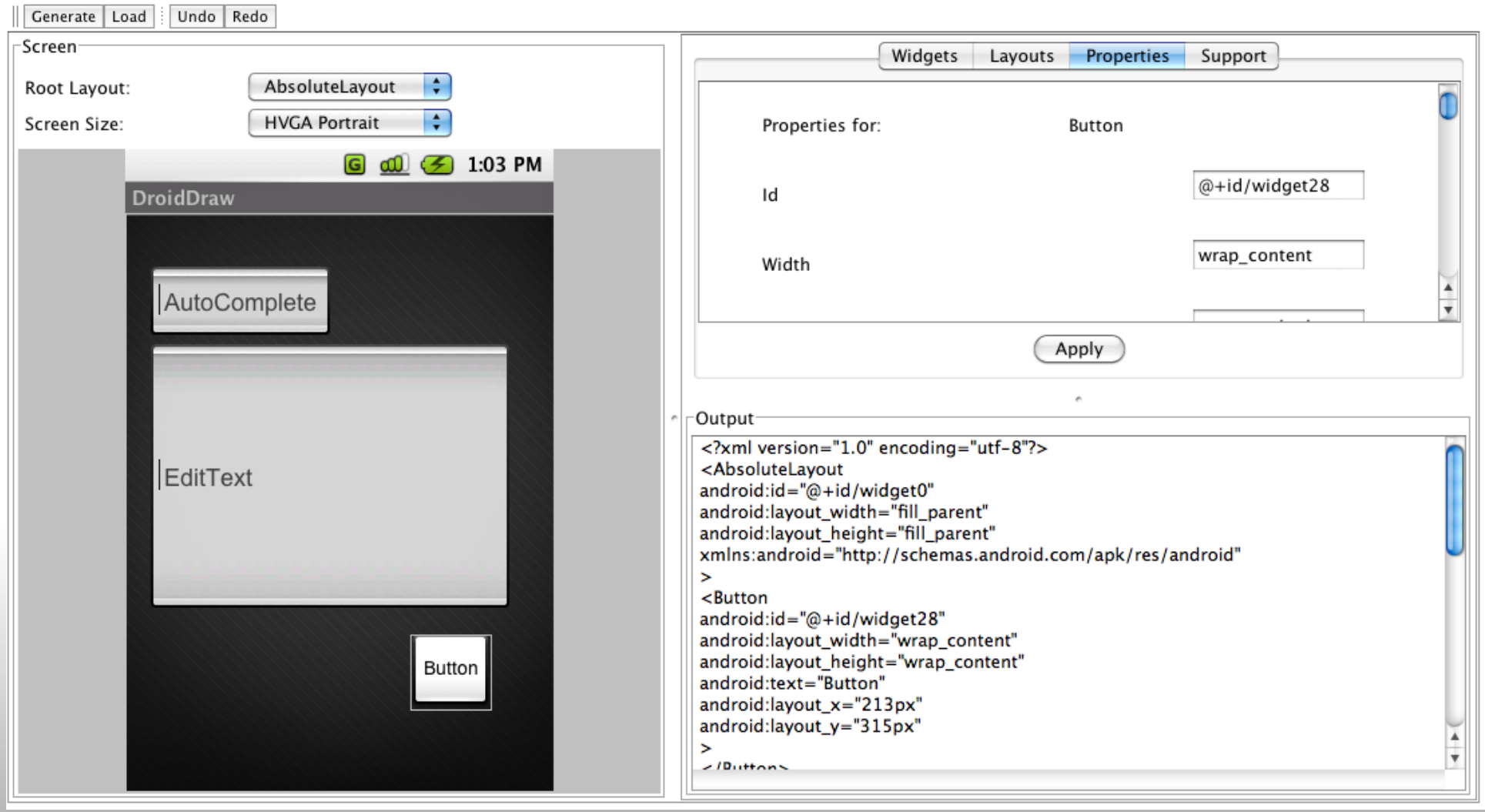
Groovy Swing Builder example

```
import groovy.swing.SwingBuilder
import java.awt.BorderLayout as BL

def swing = new SwingBuilder()
count = 0
def textlabel
def frame = swing.frame(title:'Frame', size:[300,300]) {
    borderLayout()
textlabel = label(text:"Click the button!", constraints: BL.NORTH)
button( text:'Click Me', actionPerformed: {
        count++;
textlabel.text = "Clicked ${count} time(s).";
        println "clicked"
      },constraints:BL.SOUTH)
}
frame.show()
```

DroidDraw

http://www.droiddraw.org/



Layers

Each of these Domain Specific Languages create a layer

SQL

hides the complexity of *how* data is stored

Groovy Swing Builder

hides the complexity of assembling swing objects

DroidDraw

hides some of the complexity of Android UI creation

GORM's Mapping DSL

hides the complexity of ORM

Other ideas

- web tests
- Complex service calls
- Message routing
- data transformations
- business rules

making our own?

- when you see the need for
  - fluent API
  - complex conditional building of objects
  - architectural layers

a GP dealing with time...

```
// every Wednesday in 2009
int year = 2009
Calendar cal = Calendar.getInstance();
int day = Calendar.WEDNESDAY;
cal.clear();
cal.set(year,Calendar.JANUARY,1);
List wednesdays = new ArrayList();
while(cal.get( Calendar.DAY_OF_WEEK ) != day) {
    cal.add(Calendar.DAY_OF_WEEK, 1);
}
wednesdays.add(cal.getTime());
while(cal.get(Calendar.YEAR) < year + 1) {
    cal.add(Calendar.DAY_OF_YEAR, 7)
    wednesdays.add(cal.getTime());
} // would you show this to a client?
```

A Groovy DSL for the same.

```
def wednesdays = dateDsl.every.wednesday.in(2009)
```

credit to Tim Yates

Making our own DSL

- Internal DSL or External DSL?
  - An internal DSL is just an API that follows certain rules
  - An external DSL is an API that can be interpreted from a source outside the program code...

The example I borrowed from Tim Yates is an Internal DSL. You invoke it from inside other Java/Groovy code.

Live coding example...

... creating the Groovy Date DSL step by step.

External DSL

- Coded or created outside the runtime of the main program
  - created in a file
  - edited in a form for submission
  - or using a Graphical User Interface of some kind
- In Groovy you set up a GroovyShell object and bind it to a context for execution...
  - bindings may include services or local values

Another live coding example...

External DSL

What does this do Architecturally?

- Natural Layers
  - the DSL implementation's specifics can be easily hidden and abstracted
  - expose only what's relevant in the context
- Reusable components
  - the DSL is an interface to a component that does a job
- Easily testable
  - Express use cases in DSL statements without needing an entire environment
  - Test cases become documentation of the DSL
- Simple in that it matches the domain being thought about
  - may not require a "programmer"

Everyone will Program

not everyone will be a programmer

Today, virtually everyone can read and write. Yet, not everyone is a "writer." This is how programming will be.

Books on Groovy DSLs

Books on the topic I have read and enjoyed...

"Scripting in Java"

by Dejan Bosanac
Publisher: Addison Wesley Professional
Pub Date: August 31, 2007
Print ISBN-10: 0-321-321-93-6
(introduced me to the Extension Point Pattern)

"Groovy in Action"

by Dierk Koenig with Andrew Glover, Paul King, Guillaume Laforge and Jon Skeet
January, 2007
ISBN: 1-932394-84-2
(not on DSL specifically but a vital book to own)

other books...

"Language Implementation Patterns"

by Terrance Parr
Publisher: The Pragmatic Programmers
Pub Date: Dec 2009
ISBN: 978-1-93435-645-6
(I have started reading this book but can't comment yet)

"Groovy for Domain-Specific Languages"

by Fergal Dearle
Pub Date: March 2010
ISBN: 184719690X
(I have not even looked at this book so I hope it's good)

References

- Core J2EE Design Patterns http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html
- "The Vietnam of Computer Science" Ted Neward http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx
- JN0545-Dates http://groovy.codehaus.org/JN0545-Dates
- Groovy Date DSL by Tim Yates: http://groovyconsole.appspot.com/view.groovy?id=27001
- GroovyMag http://groovymag.com/

And the entire Groovy Developer Community who contribute great examples, frameworks, and experiments for us all to tinker with.

Shawn Hartsock

@hartsock on twitter.com

shartsock@osintegrators.com

hartsock@acm.org

Senior Consultant with
Open Software Integrators