Textual DSLs
and
Code Generation
with
Eclipse and oAW
Context:
Architecture DSLs

As you understand and develop your architecture...
...develop a language to express it

The language resembles architectural concepts...
and we describe the application(s) with the language

Architecture DSL
Architecture DSL Example
component DelayCalculator {
    provides IDelayCalculator
    requires IInfoScreen
}
component InfoScreen {
    provides IInfoScreen
}
component AircraftModule {
    provides IAircraftModule
    requires IDelayCalculator
}

interface IDelayCalculator {}
interface IInfoScreen {}
interface IAircraftModule {}
namespace com.mycompany.test {
  system testSystem {
    instance dc: DelayCalculator
    instance screen1: InfoScreen
    instance screen2: InfoScreen
    connect dc.screens
      to (screen1.default, screen2.default)
  }
}

Data Replication
struct FlightInfo {
    from: Airport
    to: Airport
    scheduled: Time
    expected: Time
    ...
}

repeated singleton flights {
    flights: FlightInfo[]
}

component DelayCalculator {
    publishes flights
}

component InfoScreen {
    consumes flights
}

Pre- and Post-Conditions
interface IAircraftStatus {

  oneway message reportPosition(aircraft: ID, pos: Position) {
    pre aircraft != null: "aircraft not specified"
    pre pos != null: "position not specified"
  }

  request-reply message reportProblem {
    request (aircraft: ID, problem: Problem, comment: String)
    reply (repairProcedure: ID)
    pre aircraft != null: "aircraft not specified"
    pre problem != null: "problem not specified"
    post repairProcedure != null
  }

}

Message Sequences:
Protocol State Machines
interface IAircraftStatus {
    oneway message registerAircraft(aircraft: ID!)
    oneway message unregisterAircraft(aircraft: ID!)
    oneway message reportPosition(aircraft: ID!,
                                pos: Position!)
    request-reply message reportProblem {
        request (aircraft: ID!, problem: Problem!,
                 comment: String!)
        reply (repairProcedure: !ID)
    }
    protocol initial = new {
        state new {
            registerAircraft => registered
        }
        state registered {
            unregisterAircraft => new
            reportPosition
            reportProblem
        }
    }
}
Based on actual practical experience

Currently in use with four of my customers
Why Textual?

... or: why not graphical?
Languages and Editors are easier to build

Languages and Editors are easier to build

Evolve Language and simple editor as you understand and discuss the architecture, in real time!
Integrates easily with current infrastructure: CVS/SVN diff/merge

adapting existing models as the DSL evolves

Model evolution is trivial, you can always use grep.
Many developers prefer textual notations
Tooling

Eclipse TMF / Xtext openArchitectureWare
Specify Grammar

ANTLR Grammar and Parser is generated from this specification
Generated Metamodel

Specify Semantics
Specify Constraints

Generated Editor
Generated Editor

Code Completion

Generated Editor

Syntax Coloring
Custom Keyword Coloring
Realtime Constraint Validation

Customizable Outlines
Generated Editor

Code Folding

Generated Editor

Goto Definition
Find References
Cross-File References
Model as EMF
5

DEMO
Building a DSL
When a graphical notation is better, you can visualize.
Via M2M
Read-Only
Auto-Layout
Drill-Down

Textual DSLs
vs.
Graphical
vs.
Visualization
Graphviz

Prefuse
GrapViz Overview

GrapViz Trafo Code
Generating Code
Generate API
Maps Architectural Concepts to Implementation language (non-trivial!)

Implementation
Implementation only depends on the generated programming model API
Glue Code
Aka Technology Mapping Code
Maps API to selected platform

Implementation Code

Programming Model API

Glue Code

Runtime Infrastructure
(Platform/Middleware)

Templates

```java
public class MyClass() {  
  private List<String> myList = new ArrayList<>();  

  public void add(String item) {  
    myList.add(item);  
  }  

  public List<String> getMyList() {  
    return myList;  
  }  
}
```
Extensions

DEMO
A Code Generator
Tools Summary
EMF

Ecore meta meta model
+
Editing
Transactions
Validation
Query
Distribution/Persistence

GMF

Graphical Box/Line editors based on EMF
TMF

Building Textual Editors
currently being built from oAW Xtext

M2M

Model-to-Model Transformations
INRIA’s ATL
Several QVT implementations

M2T

Model-to-Text Transformations
JET: Java Emitter Templates
Xpand: oAW’s template engine
openArchitectureWare

One Stop Toolkit for DSLs + X

Version 4.3.1 is current
Lively ecosystem of tools and extensions
Proven track record in various domains & project contexts
Stable, productive and helpful developer, support and user communities

Integration with Eclipse:
Uses EMF as a basis
Graphical editors based on GMF
All editors and tooling based on Eclipse
oAW 5

To be released in June

Xpand, Xtend, Check, MWE migrated to Eclipse projects

Completely new, much more powerful Xtext/TMF
The End

THE END.
Thank you.
Questions?

The End

coordinates
web www.voelter.de
email voelter@acm.org
skype schoggiad
xing http://www.xing.com/profile/Markus_Voelter
linkedin http://www.linkedin.com/pub/0/377/a31