DSLs, Models and Architecture

Markus Voelter
www.voelter.de
voelter@acm.org

What is Architecture?
What is Architecting?
Domain Specific Languages
Examples
What did we do?
Why Textual?
DSL Tools
Validation
Generating Code
Analysis Tools
Standards and UML
Documentation
Summary
What is Architecture?
... the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships between them.

[Wikipedia]

... the set of design decisions which, if made incorrectly, may cause your project to be cancelled.

[Eoin Woods]
... its "style and method of design and construction".

Hayes-Roth

... comprises

- A collection of software and system components, connections, and constraints.
- A collection of system stakeholders' need statements.
- A rationale which demonstrates that the components, connections, and constraints define a system that, if implemented, would satisfy the collection of system stakeholders' need statements.

Barry Boehm
... everything that needs to be consistent throughout a software system

My Own
Modularize

Procedures, Classes, Components, Services

Encapsulate

Private Members
Frameworks
Facade Pattern
Components
Layers/Rings/Levels
Packed Data Wrapper
Contracts

Interfaces
Pre/Post Conditions
Protocol State Machines
Message Exchange Patterns
Published APIs

Decoupling

Message Queues
Deferred Consistency
Compensating TX
Handle Crosscuts
Aspect Orientation
Interceptors
Application Servers
Exception Handling

Isolate
Pure functional vs. Impure
Safety Critical Parts
OS Processes
Real-Time Kernel
Decentralization

The Internet
Emergent Behaviour
Service-Oriented Architecture

Orthogonality
Closures, Program As Data, Macros, Higher-Order Functions
Hierarchical Decomposition

Procedures/Methods
State Machines, Components

Protocols
Transactions
Locking/Synchronization
Resource Access
DOC Middleware
Orthogonal Persistence
(OR Mappers)

Make
Transparent

Make Explicit

- Dependencies
- SOA, Messaging
- Monads
- PLE Variabilities
- Persistence: Loading Data
What is Architecting?
describe
verify
communicate
implement
check

describe
unambiguously
formally
Modeling
Domain Specific Languages

A DSL is a **focussed, processable language** for describing a specific **concern** when building a system in a specific **domain**. The **abstractions** and **notations** used are natural/suitable for the **stakeholders** who specify that particular concern.
Example 1: Distributed System
Airport Management System

Monitors
Website
Aircraft-Module
Data Center
component DelayCalculator {}

component InfoScreen {}

component AircraftModule {}

component DelayCalculator {
    provides IDelayCalculator
    requires IInfoScreen
}

component InfoScreen {
    provides IInfoScreen
}

component AircraftModule {
    provides IAircraftModule
    requires IDelayCalculator
}

interface IDelayCalculator {}
interface IInfoScreen {}
interface IAircraftModule {
component InfoScreen {
    provides IIInfoScreen
}

instance screen1: InfoScreen
instance screen2: InfoScreen
...

component DelayCalculator {
    provides default: IDelayCalculator
    requires screens[0..n]: IIInfoScreen
}

component InfoScreen {
    provides default: IIInfoScreen
}

component AircraftModule {
    provides default: IAircraftModule
    requires calculator[1]: IDelayCalculator
}
component DelayCalculator {
  provides aircraft: IAircraftStatus
  provides managementConsole:
      IManagementConsole
  requires screens[0..n]: IInfoScreen
}

cOMPONENT Manager {
  requires backend[1]: IManagementConsole
}

cOMPONENT InfoScreen {
  provides default: IInfoScreen
}

cOMPONENT AircraftModule {
  requires calculator[1]: IAircraftStatus
}

component DelayCalculator {
  requires screens[0..n]: IInfoScreen
    ...
}

cOMPONENT InfoScreen {
  provides default: IInfoScreen
}

instance dc: DelayCalculator
instance screen1: InfoScreen
instance screen2: InfoScreen

connect dc.screens
    to (screen1.default, screen2.default)
namespace com.mycompany.production {
    instance dc: DelayCalculator

    // InfoScreen instances are created and
    // started in other configurations
    dynamic connect dc.screens every 60 query {
        type = IInfoScreen
        status = active
    }
}
```csharp
struct FlightInfo {
    from: Airport
to: Airport
scheduled: Time
expected: Time
...}

replicated singleton flights {
    flights: FlightInfo[]
}

component DelayCalculator {
    publishes flights
}

component InfoScreen {
    consumes flights
}

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
    }
}
```
Example 2: OSGi-based System

```java
subsystem the.root.mechaniece {

  public:

    immutable type FromimgObject {
      private: string
        memory: int
        emergency: bool
    }

    interface Request {
      report( M: FromimgObject ); void
    }

    interface Prefer {
      requestsensor( M: string ); string
    }

  private:

    component FuctionA {
      provides update( M: bool )
      requires Isssue: PRESS
    }

    component PrintingDevice {
      provides power: Press
    }

  }

subsystem the.wow.mechaniece {

  uses the.root.mechaniece

  provider:

    component Ameasury {
      task portNile scheduled oneByOneStartUp
      register begin: Submit([0,1])
    }

}}
```
Example 3: Embedded Protocol Handler
/*processing DigitalIn "B1" module*/

struct hal { DigitalInHAL; }

// datatype
{
    SinglePointIndicationWithoutTime;
    SinglePointIndicationWithTime;
    DoublePointIndicationWithoutTime;
    DoublePointIndicationWithTime;
    BitStringType30BitWithoutTime;
    BitStringType30BitWithTime;
};

// parameter types
{
    DataType default {
        subtract dso # interpretedDataType == yat SinglePointIndicationWithTime;
    }
    SubtractFilterTime default {
        attrFilterValues[0] == 0x00;
        subtract dsi # 0F == 0x00;
        subtract dsi # 1F == 0x00;
    }
    MaximumOscillatingFrequency;
}

function READDATA ||: ProceedData;
function WRITEDATA<input: ProceedData>;

struct ProceedData {
    int8 channel;
    int8 fixData[4];
};

struct Memory {
    int8 pres;
    ProceedData data;
};

instance memory Memory;

#hal DigitalInHAL memory DIB { command HighSpeedCounterInit; command HardwareInit; }

struct DIB { int16 chl_16; int32 ha_counter; };

halImpl DigitalInHALImpl for DigitalInHAL;
halImpl Win32HALImpl for DigitalInHAL;
procedure existsRequestedNumber2 requestCode 3x29 {
    request: struct request1 {
        int8 srcPattern; // 2:000;
        C:parentRequestCode;
    };
    int8 registerAddress;
}

reply: struct doesn'tReply {
    int8 statusByte patternedStatusByte;
    int8 dontCare patterned defaultReturn;
};

request: struct request2 {
    int8 registerType pattern (4:0:0:0);
    4:registerType;
    int8 registerAddress;
    int8 registerData [2];
};

system 0x29 Input target ainel extends ModuleType 0x29 rack 1 {
    disp: DigitalIn alot 1 with DigitalInAllInput channels 16
    using datatypes {
        SinglePointIndicationsWithoutTime;
        SinglePointIndicationsWithTime;
        BitStringType2BitWithoutTime;
        BitStringType2BitWithTime;
    };
    dispatcher {
        processingComponent DigitalIn {
            on normalizeModule;
            on readStandardStatus;
            on readModuleInService;
            on readRegisterOfChannel12;
            on requestLastReply;
            on requestObject;
            on writeRegisterNumberOf;
        };
    };
}
procedures writeRegisterNumber2 requestCode Ox19
    request: struct request I
    uint8 uin pattern (3:0);
    {params();
      uint8 registerAddress;
    };
    reply: struct don'tCareReply I
      uint8 statusType patterned:
        uint8 registerAddress;
        uint8 LogPattern:
          uint8 registerAddress;
        };
    request: struct request I
    uint8 registerType pattern (4:0):
      uint8 registerType:
        uint8 registerAddress;
        uint8 registerAddress;
      };
    };
}

register parameterInstruction address Ox37 struct (uint8 db1 pattern (3:0):
  {params();
    uint8 channelNumber;
  });

procedure writeRegisterNumber2 requestCode Ox19
    send request I
    attr registerAddress == registerAddress;
    {params();
      uint8 registerAddress;
      uint8 LogPattern:
        uint8 registerAddress;
        uint8 registerAddress;
      };
    send request I
    attr registerType == registerType:
      send parameterInstruction:
      {params();
        uint8 channelNumber;
      };

Example 4: 
Embedded Systems & Robots
This module represents the code for the line follower robot. It has a couple of main imports: ClockKernel, EState, BuildEnvironment.

```
constant int WHITE = 500;
constant int BLACK = 700;
constant int SLOW = 20;
constant int FAST = 40;
```

This state machine manages the line follower:

```
state machine linefollower {
    event initialized {
        initialized (true) -> runn
    }
    state running {
        ...
    }
}
```

This function resets the light sensor on the robot:

```
escribe_set_light_sensor:
    event linefollower:initialize {
        ...
    }
```

```
exported interface MotorControl {
    void stop( );
    void setLeftSpeed( int8 speed );
    void setRightSpeed( int8 speed );
}
```

```
exported component Motors {
    provides MotorControl : MotorControl;
    exported component implementation MotorNXT : Motors {
        procedure void motorControl::stop( ) {
            motor::set::speed(0, 0, 0);
        }
        procedure void motorControl::setLeftSpeed( int8 speed ) {
            motor::set::speed(speed, 0, 0);
        }
        procedure void motorControl::setRightSpeed( int8 speed ) {
            motor::set::speed(0, speed, 0);
        }
    }
}
```
module impl imports <<imports>> {

    int speed( int val ) {  
        return 2 * val;
    }

    robot script stopandrecoil 
        block main on bump block retreat on bump <no bumpReaction>
            step
                accelerate to 8 - 30 within 2000
                drive on for 2000
                decelerate to 0 within 3000
            step
                accelerate to speed(25) within 3000
                drive on for 2000
                turn left for 2000
                block driveMore on bump <no bumpReaction>
                    accelerate to 80 within 2000
                    turn right for 3000
                    decelerate to 0 within 3000
                step
        }
}
Example 5: „Business“ Architecture
### 3.3 Commutatiegetallen op 1 leven

\[ D = \sqrt{\frac{1}{100}} \times \frac{1}{x} \times \frac{1}{e} \times \frac{1}{365} \times \frac{1}{100} \times Dec(3) \]

\[ \sum_{x=0}^{\infty} D = 7 \times Dec(3) \]

### 3.6 Contantie waarde 1 leven/2 levens

\[ D = \frac{1}{(19 \times Dec(4))} \times \frac{1}{(18 \times Dec(3))} \times \frac{1}{(17 \times Dec(3))} \times \frac{1}{(16 \times Dec(3))} \]

\[ N = N \times D = 20 \times Dec(3) \]

\[ 4 \times Dec(3) = 4 \times Dec(3) \]

---

### Elements

#### Rules

- **Rule 1:** Score Trend Periode

  - **Formula:**
    - Multiperioden
    - Multiperiode
    - Multiperiode
    - Multiperiode
    - Multiperiode

  - **Trend:**
    - Trend 1
    - Trend 2
    - Trend 3
    - Trend 4
    - Trend 5

- **Tests:**
  - Test 1
  - Test 2
  - Test 3

- **Results:**
  - Result 1
  - Result 2
  - Result 3

- **Conditions:**
  - Condition 1
  - Condition 2
  - Condition 3

- **Variables:**
  - Variable 1
  - Variable 2
  - Variable 3

- **Calculations:**
  - Calculation 1
  - Calculation 2
  - Calculation 3

### Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Valid time</th>
<th>Transaction time</th>
<th>Pattern</th>
<th>Product</th>
<th>Expected value</th>
<th>Actual value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Column 1:** Name
- **Column 2:** Valid time
- **Column 3:** Transaction time
- **Column 4:** Pattern
- **Column 5:** Product
- **Column 6:** Expected value
- **Column 7:** Actual value

<table>
<thead>
<tr>
<th>Period</th>
<th>Valid time</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>36/5/2006</td>
</tr>
<tr>
<td>36</td>
<td>36/5/2006</td>
</tr>
<tr>
<td>36</td>
<td>36/5/2006</td>
</tr>
</tbody>
</table>
What did we do?
As you understand and develop your Architecture...

Develop a language to express it!
Language resembles architectural concepts

We express the application(s) with the language.
Architectures

Clear Understanding from building the language
Unambiguous Vocabulary

Concepts independent from Technology
Programming Model can be defined based on Conceptual Architecture

Architecture „executable“ (i.e. more than rules and docs)
Why textual?
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

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

Textual DSLs
vs.
Graphical
vs.
Visualization
Validation
Grammar is not expressive enough

More Validation Rules Required: Constraints
Simple Examples:
Name-Uniqueness
Type Checks
Non-Nullness

More Complex Examples:
Version Compatibility
Overall completeness
Quality of Service
context Interface ERROR "interface names must start " +
"with a capital I":
name.startsWith("I");

context Component ERROR "Qualified Name "+
qualifiedName()+" must be unique":
allComponents().
  select( c | c.qualifiedName() == qualifiedName() )
  .size == 1;

context Attribute ERROR "no type defined: " +
type.name:
visibleInstancesOfType(this, DataType)
  .contains(type);

context Connector ERROR "target must be provided port":
ProvidedPort.isInstance( u);
Precondition II:
All Data Available
Generating Code

Since we already have a formal model....
Generate API

Maps Architectural Concepts to Implementation language (non-trivial!)

Implementation

Implementation only depends on the generated programming model API

Runtime Infrastructure
(Platform/Middleware)
### Programming Model

*Generated API + Usage Idioms*

*Completely Technology-Independent*

- Implementation Code
- Programming Model API
- Glue Code
- Runtime Infrastructure (Platform/Middleware)

### Runtime Infrastructure

*Select based on fit wrt. to architectural concepts and non-functional requirements*

- Implementation Code
- Programming Model API
- Glue Code
- Runtime Infrastructure (Platform/Middleware)
Glue Code
Aka Technology Mapping Code
Maps API to selected platform

Implementation Code

Programming Model API

Glue Code

Runtime Infrastructure
(Platform/Middleware)

Glue Code
Contains Configuration Files for Platform
Might require „mix in models“

Implementation Code

Programming Model API

Glue Code

Runtime Infrastructure
(Platform/Middleware)
Several Platforms

Different Platforms, not Languages
Support for Scaling (non-functional req)

- Implementation Code
- Programming Model API
- Glue Code
- Glue Code 2
- Runtime Infrastructure (Platform/Middleware)
- Platform 2

Testing!
Architecture Analysis Tools

Software Tomography
SonarJ
Structure 101
... and the like
How do you describe Architecture?

Constraint-Driven:
- Assemble Packages into Components
- Define layers/columns
- Define and Check valid Dependencies
You do not get a way to define and express architectural abstractions beyond those!

Dependency Analysis
Anti Pattern Detection
Metrics
[Trend Analysis]
[Visualization]
Police Approach:
Detect Problems
After the fact!

Anti Pattern Detection Metrics

Useful for manually written parts of the system
Dependency Analysis

Useful for manually written parts of the system

Constraints generated from the model.
Standards & UML

Formal Architecture Description is not new:
ADLs, UML
But all of those use existing, generic languages!

This misses the point!
Trying to express your specific architecture with predefined abstractions is not useful!

You want to build a language to capture your own architectural abstractions as you learn things.
Where are standards useful?

People have to learn architectural concepts anyway.
Is UML with a profile still a standard language?

On which meta level do I want to standardize?
M2 (UML), M3 (MOF)?
Isn’t a **DSL** based on **MOF** as „standard“ as a profile based on **UML**?

**UML Profiles instead?**

You’ll think more about UML-ities than your own concepts
UML Profiles instead?
Tool integration issues (repository, diff/merge, versioning)

UML Profiles instead?
Tools are often complex, heavyweight, bloated. Acceptance limited.
UML Generally Useless?
No. UML can be used for documentation (sequence diagrams, eg)
Documentation
The DSL and the „programs“ are documentation.

So I don‘t have to write... ... more docs?
Not quite.
Grammar is a good formal definition, but not a teaching tool.
Tutorials:
Architectural Concepts (Meta Model)
How to use the language
How to use the programming model
(How to generate code)
(How to add manual code)

Example Driven!
Grammar describes what the architecture looks like.
Grammar describes what the architecture looks like.

But: Why is it that way?
Rationales
Conceptual Architecture
Relate to Grammar

Rationales
Conceptual Architecture
Relate to Grammar
Technological Decisions
Non-Func. Req.
Make Explicit

Abstraction
Isolate Technology

Limit Freedom
THE END.

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