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
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

describe

verify
describe
verify
communicate
implement

describe
verify
communicate
describe
verify
communicate
implement
check

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.
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.
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.
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.
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 naturally suitable for the **stakeholders** who represent 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 IInfoScreen
}

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:
    IMManagementConsole
  requires screens[0..n]: IIInfoScreen
}

component Manager {
  requires backend[1]: IMManagementConsole
}

component InfoScreen {
  provides default: IIInfoScreen
}

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
    }
}

interface IAircraftStatus {

  oneway message reportPosition
      (aircraft: ID, pos: Position )

  request-reply message reportProblem {
      request (aircraft: ID, problem: Problem,
               comment: String)
      reply (repairProcedure: ID)
  }
}

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
Example 3: Embedded Protocol Handler

```plaintext
#define DigIn "31" moduletype DigitalInML 
{ 
  datatypes { 
    SinglePointIndicationWithoutTime; 
    SinglePointIndicationWithTime; 
    DoublePointIndicationWithoutTime; 
    DoublePointIndicationWithTime; 
    BitStringType208BitWithoutTime; 
    BitStringType208BitWithTime; 
  } 
  parameters { 
    DataIType default { 
      subattr db0 # intendedDataType => sd1 SinglePointIndicationWithTime; 
    } 
    @bouncedFilterTime default { 
      subattr db0 # EP => bx00; 
      subattr db1 # TN => bx01; 
    } 
    MaximumOscillatingFrequency; 
  } 
  function READDATA [ ] : ProcesData; 
  function WRITEDATA [input : ProcesData]; 
  struct ProcesData { 
    int8 channel; 
    int8 fInData[4]; 
  } 
  struct Memory { 
    int8 state; 
    ProcesData data; 
  } 
  instance memory Memory ; 
}
```
```c
hal DigitalInHAL memory DID {
    command HighSpeedCounterInit;
    command HardwareInit;
}

struct DID {
    int16 ch1_t16;
    int32 ha_counts;
}

halImpl DigitalInHALImpl for DigitalInHAL;
halImpl WinSiHALImpl for DigitalInHAL;
```

```c
procedure writeRegisterSmack2 requestCode 3x29 {
    request: struct request1 {
        int8 arc pattern;
        2: u32;
        CiparentRequestCode;
    };
    int8 (registerAddress);
} ;
reply: struct doneCommandReply {
    int8 statusByte patterned statusByte;
    int8 doneCommand patterned defaultReturn;
} ;
request: struct request2 {
    int8 registerType pattern (4:1 0:0);
    4: (registerType);
} ;
int8 registerAddress;
int8 registerData[2];
} ;
} ;
```
system DemoInput target #1 extend ModuleType OnOff rack 1 {
  digi: DigitaleInList #1 with DigitaleInModule type channels 4
  using datatypes {
    SinglePointIndicationWithoutTime;
    SinglePointIndicationWithTime;
    BitStringType2BitWithoutTime;
    BitStringType2BitWithTime;
  };
  dispatch {
    processComponent DigitaleIn {
      on normaliseModule:
      on readStandardStatus:
      on readModuleInError:
      on readDataDigitaleInputChannels:
      on request(LastReply):
      on resetBoiler:
      on writeRegisterNumValue;
    }
    }
  }
}

procedure writeRegisterNumValue (requestId 0x13) {
  request: direct request1: {
    int & pattern (2:000);
    int & registerAddress;
  };
  reply: direct donCaret(0);
  int statusByte patterned statusByte:
  int donCaret patterned (statusByte :
    int registerAddress patterned (donCaret)
  );
  request: direct request2: {
    int registerType pattern (4:0000);
    int registerType patterned
      registerType (4:0000) :
      int registerAddress patterned
        (registerType [2][2])
      int registerData [2][2];
      };
}

register parameterInstruction address 0x3F struct {
  int8 db1 pattern (2:000);
  int8 channelNumber:
};
Example 4: Embedded Systems & Robots
This module represents the code for the line follower lego robot. It has a couple of
module main imports LegoKernel, ECAST, ELevelUtils;
constant int WHITE = 500;
constant int BLACK = 700;
constant int SLOW = 20;
constant int FAST = 40;

-- State machine to manage the
stateMachine linefollower {
    event initilized;
    initial state initializing {
        initialized [true] -> runn
    }
    state running {
        ...
    }
    ...
}

-- Initialize
initialise {
    script.on_close()
    event linefollower:initializ
}
exported interface MotorControl {
    void stop();
    void setLeftSpeed(int8 speed);
    void setRightSpeed(int8 speed);
}

exported component Motors {
    provides MotorControl : MotorControl;
}

exported component implementation MotorsNXT : Motors {

    procedure void motorControl.stop() {
        nxt_motor_set_speed(MOTOR_PORT_T1::NXT_PORT_B, 0, 1);
        nxt_motor_set_speed(MOTOR_PORT_T1::NXT_PORT_C, 0, 1);
    }

    procedure void motorControl.setLeftSpeed(int8 speed) {
        nxt_motor_set_speed(MOTOR_PORT_T1::NXT_PORT_B, speed, 1);
    }

    procedure void motorControl.setRightSpeed(int8 speed) {
        nxt_motor_set_speed(MOTOR_PORT_T1::NXT_PORT_C, speed, 1);
    }

}

module impl imports <<imports>> {

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

    robot script stopAndGo
        block main on bump block retreat on bump <no bumpReaction>
            stop
            accelerate to 0 - 30 within 2000
            drive on for 2000
            decelerate to 0 within 1000
            step
            accelerate to speed(25) within 3000
            drive on for 2000
            turn left for 2000
            block driveNext on bump <no bumpReaction>
                accelerate to 80 within 2000
                turn right for 3000
                decelerate to 0 within 3000
                step
        
}
Example 5: „Business“ Architecture
1.12.2010

---

<table>
<thead>
<tr>
<th>3.3 Commutatiegetallen op 1 leven</th>
</tr>
</thead>
</table>

\[
D = N e^{D(3)}
\]

\[
N = \sum_{j=0}^{\infty} e^{jT (3)}
\]

---

<table>
<thead>
<tr>
<th>3.6 Contant waarde 1 leven/2 leven</th>
</tr>
</thead>
</table>

\[
E = \frac{1}{D}\sum_{j=0}^{\infty} e^{jT (4)}
\]

\[
s - \delta = 0.5 e^{jT (3)}
\]

\[
N = \sum_{j=0}^{\infty} e^{jT (3)}
\]

---

<table>
<thead>
<tr>
<th>4 BNC (sic) koopsommen</th>
</tr>
</thead>
</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.

Architecture

DSL
Clear Understanding from building the language

Unambigous 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
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
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 Visualization

Graphviz
Tools
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");

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

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

current Connector ERROR "target must be provided port":
ProvidedPort.isInstance( u);
Precondition I: Algorithm for Constraint

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

Programming Model
Generated API + Usage Idioms
Completely Technology-Independent
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“

Several Platforms
Different Platforms, not Languages
Support for Scaling (non-functional req)
Architecture Analysis Tools
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, e.g.)
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.
Grammar is a good formal definition, but not a teaching tool.

→ Tutorials

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)
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.

But: *Why is it that way?*
Rationales

Conceptual Architecture
Relate to Grammar
Technological Decisions
Non-Func. Req.
Summary

Make Explicit
Abstraction

Viewpoints
Notation

Isolate Technology
Limit
Freedom

Translate
Automate
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
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
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