Architecture Rules Enforcement and Governance Using Aspects

Srini Penchikala

SATURN 2009

About the Speaker

- Enterprise Architect
- Writer, Speaker, Editor (InfoQ)
- Detroit Java User Group Leader
- Working with Java since 1996, JEE (2000), SOA (2006), & PowerPoint since Sep 2008
- Current: Agile Architectures, Domain-Driven Design, Architecture Enforcement, Model Driven Development
- Future: Role of DSL's in Architecture Enforcement

Goals for this Presentation

- Overview of Reference Architecture (RA) and its significance in EA
- How Aspects and AOP can help to enforce RA and manage Architecture Governance

Format

- Interactive
- Demos
- Duration: ~60 minutes
- Q & A
- Prerequisite: Familiarity with AOP and Aspects

Before we start…

- How many currently have some kind of Reference Architecture in place?
- How many actually use RA to enforce architecture?

Agenda

- Reference Architecture & Enforcement
- Architecture Rules Categories
- Architecture Enforcement Approaches
- Aspect-oriented Programming
- Rules Demo(s)
- CI Process Changes
- Case Study
- Architecture Rules Aspects - Open Source Project

Agenda

- Reference Architecture & Enforcement
- Architecture Rules Categories
- Architecture Enforcement Approaches
- Aspect-oriented Programming
- Rules Demo(s)
- CI Process Changes
- Case Study
- Architecture Rules Aspects - Open Source Project

Reference Architecture

- A high-level system design which consists of:
  - description of system components
  - definitions of relationships between components
  - definitions of relationships between system components & elements external to the system

Architecture Enforcement

- Architecture Rules:
  - Rules (compile time as well as run time) used to assert architecture.
- Architecture Governance:
  - Construct an Architecture Contract to govern overall implementation & deployment process.
  - Perform appropriate governance functions while system is being implemented and deployed.
  - Ensure conformance with defined architecture by implementation projects

Architecture Enforcement Maturity Model

- Level 0: No RA
- Level 1: RA is described in a Word/HTML/PDF document
- Level 2: Semi-automated checking mechanism for architecture enforcement
- Level 3: Fully automated checking mechanism

Why Architecture Enforcement?

- Checking mechanism to enforce Reference Architecture
- Match Requirements (Architecture) to Implementation (Code)
- Detect and prevent structural complexity
- Promotes consistency and modularity in the System
- Aids in Architecture, Design, and Code refactoring efforts

Architecture Rules

- Business v. Architecture Rules
  - Architecture rules are as important as business rules
- Code Quality By Design
  - Unit tests, TDD & BDD only help with code quality from functional requirements stand-point
  - Need for a process to ensure code quality from design stand-point
- Technical Credit
  - Good Design, Testable and Integrated Code
  - Code that complies with Architecture/Design Policies

Types of Enforcement

- Layered architecture and application module dependencies
- API usage (internal and 3rd party)
- Coding Policies
- Code-Naming Conventions
- Design Patterns and Best Practices

Setup Types

- Static analysis
  - Allows for easy visual representation
  - Techniques
    - AOP, AspectJ
  - Tools
    - Structure 101
    - SonarJ
    - Lattix
- Dynamic analysis
  - Based on the program flow
  - Rule-set usable at runtime
  - Techniques: AspectJ/Spring AOP

Agenda

- Reference Architecture & Enforcement
- Architecture Rules Categories
- Architecture Enforcement Approaches
- Aspect-oriented Programming
- Rules Demo(s)
- CI Process Changes
- Case Study
- Architecture Rules Aspects - Open Source Project

Rule Categories

- Layered Architecture
- Separation Of Concerns
- Domain-Driven Design
- Infrastructure
- Miscellaneous

Rules - Layered Architecture

- Presentation layer should not use DAO classes directly
- Service layer never calls web layer
- DAO (Persistence) layer depends on no other layer except domain

Rules - Separation of Concerns

- No transaction management in DAO classes
- Only DAO's should reference javax.sql classes
- Only Controller classes are aware of Web/HTTP objects
- Service layer should be transactional
- External systems can not access internal implementation classes (*Impl classes)

Rules - Domain-Driven Design

- Service object creation via a Factory (i.e. no “new Service()” code)
- No direct access to DAO’s except from Domain classes
- Business service that fails with a concurrency related failure can be retried

Rules - Infrastructure

- No direct use of API:
  - Log4J or Commons Logging (use Framework Logger)
  - JavaMail (use MailService component)
  - FTP (use FileTransferService component)
- No System.out.println statement anywhere in the code

Rules - Miscellaneous

- All primary key classes (*PK.java) should implement Serializable interface
- All test classes should be named with the suffix “Test”

Persistence

- Hibernate session close only allowed in EJB's and Servlet Filters
- DAO's accessing DB Views use JDBC (no Hibernate)
- Use SpringJdbcTemplate for JDBC data access

Other Domain Concerns

- Auditing
  - Domain state change tracking
- Monitoring
  - Aspects to implement MonitorMBean interface
  - Expose Results as an MBean
  - Publish results using RSS feeds

Agenda

- Reference Architecture & Enforcement
- Architecture Rules Categories
- Architecture Enforcement Approaches
- Aspect-oriented Programming
- Rules Demo(s)
- CI Process Changes
- Case Study
- Architecture Rules Aspects - Open Source Project

Approaches

- Code Inspections
- Architecture Reconstruction
- Model Driven Architecture (MDA)
- Code Generation
- Byte Code Instrumentation
- Enforcement Tools
- Aspect-oriented Programming (AOP)

Architecture Analysis Tools

- Macker
- ArchitectureRules
- Classycle
- PatternTesting
- Contract4J
- Structure 101, Lattix, SonarJ

Architecture Analysis Tools

- Structure 101
  - Slicing / Rules
- SonarJ
  - Logical architecture definition & Physical mapping of architecture to Java code
- Lattix
  - Dependency Structure Matrix showing the Desired vs. the Realized architecture implementation

Agenda

- Reference Architecture & Enforcement
- Architecture Rules Categories
- Architecture Enforcement Approaches
- Aspect-oriented Programming
- Rules Demo(s)
- CI Process Changes
- Case Study
- Architecture Rules Aspects - Open Source Project

Aspect-oriented Programming

- Add behavior to objects in a non-obtrusive manner through static and dynamic crosscutting
- Code cross-cutting concerns in separate modules and apply them in a declarative way

AOP Use Cases

- Spring Framework Built-In Aspects:
  - Transaction Management
  - Security
- Custom Aspects:
  - Profiling
  - Caching
  - Architecture Rules
  - Contract Enforcement

AOP in Architecture Enforcement

- Provides the necessary abstraction to enforce rules
- Customized compile-time error or warning messages
- Less intrusive
- Target application code is not modified in any form
- Rules can be turned on or off any time

Enforcement: Aspects or Tools?

- Question:
  - AOP or Tools?

Enforcement: Aspects or Tools?

- Tools:
  - +Better enforcement options
  - -Licensing costs
- Aspects
  - +More flexibility for customization
  - -Relatively intrusive
  - -Not backed by an architecture meta model

Enforcement: Aspects or Tools?

- Question:
  - AOP or Tools?
- Answer:
  - It depends..
  - Choose the right tool to do the right job

Agenda

- Reference Architecture & Enforcement
- Architecture Rules Categories
- Architecture Enforcement Approaches
- Aspect-oriented Programming
- Rules Demo(s)
- CI Process Changes
- Case Study
- Architecture Rules Aspects - Open Source Project

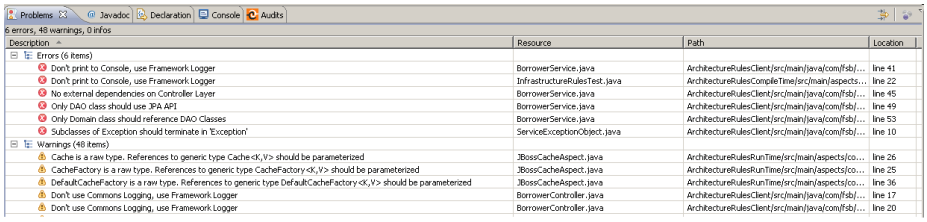
Architecture Rules Enforcement

- IDE (Eclipse, AJDT)
- Build Process Changes (Ant, CruiseControl)
- RSS Feeds (Rome)

Sample Application

- Tools:
  - Eclipse
  - AspectJ
  - AJDT
  - Spring AOP
  - Ant

Sample Application - Eclipse



Layered Architecture Rules

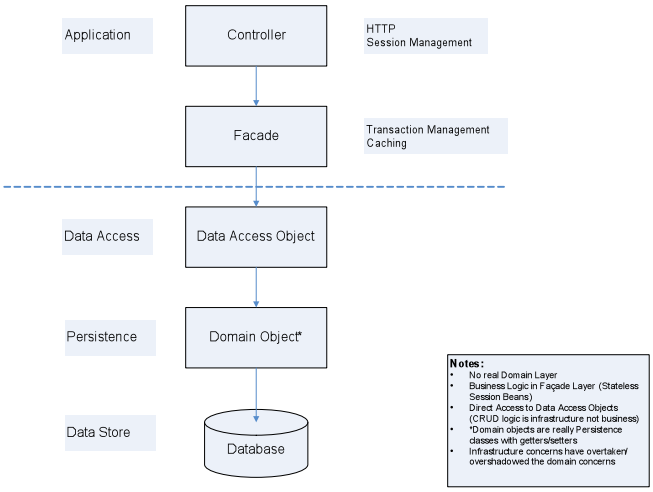
- Service classes should not depend on Application layer classes

- DEMO

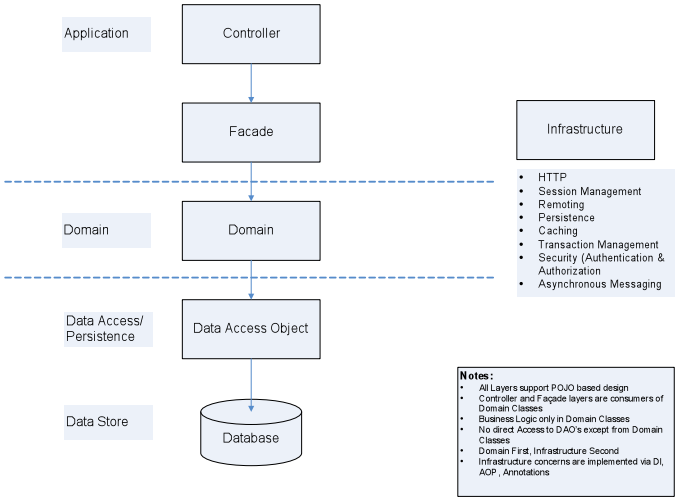
Domain-Driven Design Rules

- No direct access to DAO's except from Domain classes (go through Domain or Repository objects)

Typical J2EE Architecture Model



Domain Driven Architecture



- DEMO

DDD Rules Demo 2

- Business service that fails with a concurrency related failure can be retried
- Caching example
  - Spring AOP and EHCache
  - Requirement: To cache specific data (objects) using a custom Annotation
  - Annotation: @CacheEntity

- DEMO

Infrastructure Rules

- No direct use of API:
  - Log4J or Commons Logging (use Framework Logger)

- DEMO

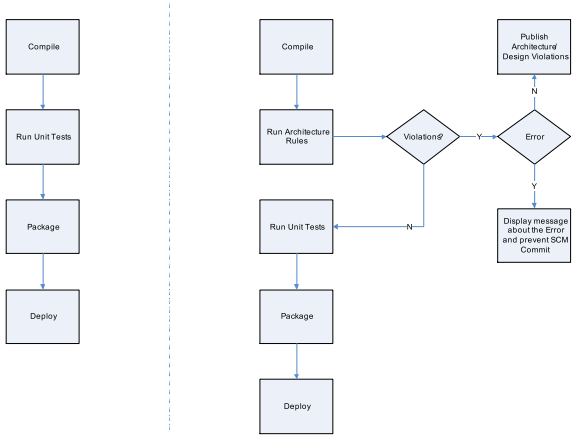
Agenda

- Reference Architecture & Enforcement
- Architecture Rules Categories
- Architecture Enforcement Approaches
- Aspect-oriented Programming
- Rules Demo(s)
- CI Process Changes
- Case Study
- Architecture Rules Aspects - Open Source Project

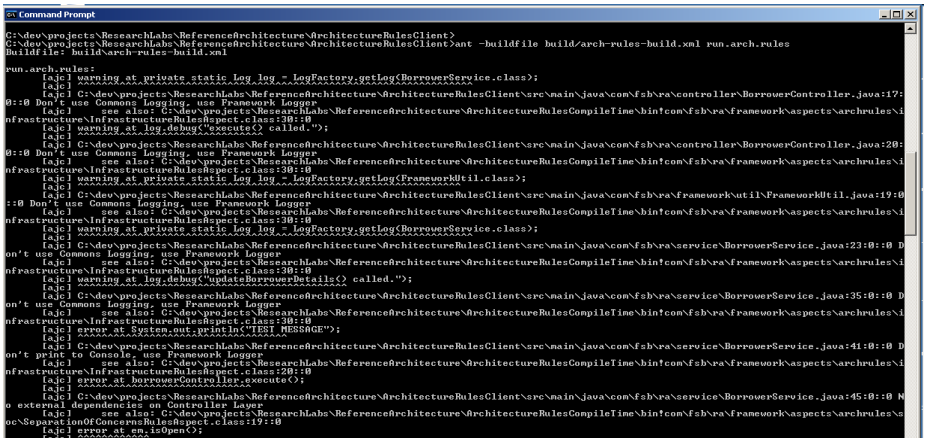
CI Process Changes

- Build process changes to include Architecture rules task
- Part of build process (in Local and Integration environments)
- Deviations are published nightly as RSS feeds
- Identified deviations are used in conducting design and code reviews

Build Process Changes



Sample Application - Ant Build



Agenda

- Reference Architecture & Enforcement
- Architecture Rules Categories
- Architecture Enforcement Approaches
- Aspect-oriented Programming
- Rules Demo(s)
- CI Process Changes
- Case Study
- Architecture Rules Aspects - Open Source Project

Case Study

- Implementation Plan:
  - Phase 1 (Day 1): Enable rules but not enforce them (yet)
  - Phase 2 (Day 16): Enable rules and enforce only critical Errors (No Warnings)
  - Phase 3: Enable rules and enforce all deviations (Errors & Warnings)
  - Phase 4: Enable rules and enforce all Errors and Warnings in all applications in the enterprise

Architecture Rules Aspects

- Open source Google project
  - architecture-rules-aspects (http://code.google.com/p/architecture-rules-aspects/)
- Future road-map:
  - DSL to define architecture rules (e.g. Groovy)
  - Define rules in a Rules Engine (JBossRules)

Implementation Road Map

- Publish rules along with Reference Architecture (RA) and Standard Technology/Framework Stack
- Pick a pilot application to enforce architecture rules as part of the build process (and Eclipse IDE)
- Incorporate architecture enforcement into other applications
- Training/Mentoring as part of an on-going Architecture Governance effort

Conclusions

- Make architecture rules part of build process to detect deviations earlier in SDLC
- Use enforcement check results to improve architecture/design/code (via refactoring efforts)
- Capture design patterns and best practices as policy enforcement Aspects
- Refine & refactor enforcement rules

AOP Tools

- AJDT
- Spring AOP
- MaintainJ: Sequence Diagram Using AspectJ
- PointcutDoctor
- Jexin

References (1/2)

- Architecture Rules Aspects Google Project (http://code.google.com/p/architecture-rules-aspects/)
- AspectJ In Action (2nd Edition)
- Building Software As Solid As The Pyramids (Ramnivas Laddad & Alef Arendsen)
- Using Aspect-Oriented Programming to Enforce Architecture (Paulo Merson)
- AOP In the Enterprise (Adrian Colyer)
- Aspects & policy injection - clean up your code
- Policy Injection Application Block

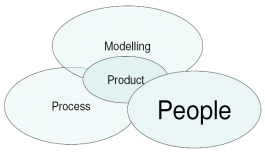
References (2/2)

- Structure 101
- SonarJ (http://www.hello2morrow.com)
- Lattix
- Contract4J (http://www.contract4j.org/contract4j)
- Macker (http://innig.net/macker/)
- Rome - RSS Feeds Framework (https:// rome.dev.java.net/)

Contact Information

- Domain-Driven Design and Enterprise Architecture articles on InfoQ
- InfoQ website (http://www.infoq.com)
- E-Mail: srinipenchikala@gmail.com
- Blog: http://srinip2007.blogspot.com

- Q & A



Non-Linear First-Order Components
Alistair Cockburn

Thank You

- Thank you for your attention
- Feedback survey