Plugin Architecture

Institute of Systems Biology
Seattle, WA

Introduction

Hector Rovira
Senior Software Developer and Technical Architect
MITS, Inc - Advanced Reporting and Business Intelligence

Current Project: MITS Report 2.0
- what is MITS Report?
- asynchronous extractions from relational databases
- connection plugins to access different database vendors
- UI plugins enable separate releases of new features

Software Toolkit: Java, Spring, Maven2, Ant, Web2.0, AOP, Asynchronous Messaging, Relational Databases, Hibernate, SQL, JDBC, JSP/Servlets, XML, Web Services, JUnit, etc

Goals

- Extension framework for core application functionality

- Improve speed of feature development and delivery

- Ability to grow the core application as popular plugins are generalized for community use

- Improved software quality as smaller components can be tested in isolation

- Enhance popularity of the overall application (renewed excitement as plugins come online)

- Involve community in the development effort

Challenges

- Clear Upgrade Path
  allow plugins to work seamlessly with any version of the core application within a major release

- Ease of Integration
  developers should be able to understand the Core Plugin APIs, provide implementations to satisfy their needs, integrate and test against the core application quickly and intuitively

- Avoid Conflicts Between Plugins
  plugins should be able to coexist, cooperate and work in the same transaction without affecting each other's processes

- Resource Management
  plugins should delegate to the core the responsibility of managing threads, connections, pools, files, etc...

Critical Design Principles

- Separation Of Concerns
- Inversion of Control
  (and Dependency Inversion Principle)
- Dependency Injection Pattern
- Composition vs Inheritance

Separation of Concerns

Goal is to design systems so that functions can be optimized independently of other functions, so that failure of one function does not cause other functions to fail, and in general to make it easier to understand, design and manage complex interdependent systems

A concern is a single piece of interest or focus in a program

AOP addresses cross-cutting concerns:

- transactions
- access control
- monitoring (performance, activity, audits)
- context control (hibernate)

Some implementation guidelines:

- consider variety of imports in the class
- how difficult is it to name the class
- does the class implement a functional interface

Inversion of Control

One important characteristic of a framework is that the methods defined by the user to tailor the framework will often be called from within the framework itself, rather than from the user's application code. The framework often plays the role of the main program in coordinating and sequencing application activity. **This inversion of control gives frameworks the power to serve as extensible skeletons**. The methods supplied by the user tailor the generic algorithms defined in the framework for a particular application.
--Ralph Johnson and Brian Foote

Dependency Inversion Principle

The Dependency Inversion Principle has been proposed by Robert C. Martin. It states that:

High level modules should not depend upon low level modules. Both should depend upon abstractions. Abstractions should not depend upon details. Details should depend upon abstractions.

This principle seeks to "invert" the conventional notion that high level modules in software should depend upon the lower level modules. The principle states that high level or low level modules should not depend upon each other, instead they should depend upon abstractions.

Dependency Injection Pattern

- Declare dependencies on interfaces

- Promotes loosely coupled and testable objects

- Facilitates separation of concerns

- Easier to introduce aspects

- Individual components are easily testable, as mock objects can be injected as dependencies

- Variety of frameworks to instantiate and inject dependencies

- Injection Styles: Constructor, Setter and Interface

Composition vs Inheritance

Composition
- Functionality is provided at runtime
- Outsourcing concerns
- Disparate types can reuse logic
- Stronger encapsulation
- Easier to test and replace code
- Easier to find Generic cases

- Components may be too granular
- Code is not as easy to understand or navigate

Inheritance
- Easy to reuse functionality
- Abstracts caller from subclass
- Logic is less abstract

- Breaks encapsulation, child must understand inner workings of parent
- Superclass interface is fragile
- Tends to promote rigid functional hierarchies
- Couples data and functionality

Samples

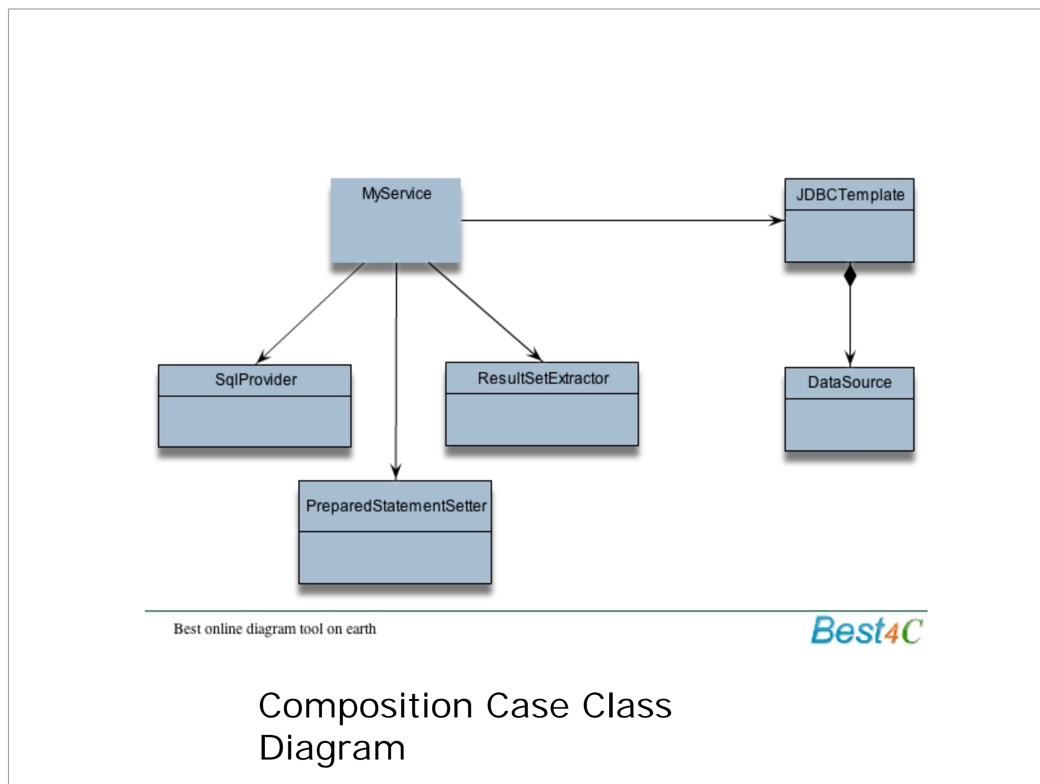
MyService1 - example of a class with many concerns

Inheritance Solution

MyAbstractService - refactors common functionality
MyService2 - extends from MyAbstractService and processes ResultSet

Composition Solution

MyService3 - injected with JdbcTemplate (contains DataSource, handles all database connectivity), instantiates ResultSetExtractor and owns sql statement
MyService4 - delegates instantiation and sql statement (may be too granular)
MyService5 - more practical implementation
isbSamples.xml - contains dependency injection samples
All sample code found at http://isbsamples.googlecode.com/svn/trunk



Composition Case Class Diagram

Integration Point Design

- Integration Points

- Plugin Design

- Core API Dependencies

- Testing
  - Core should offer testing modules, frameworks and guidance
  - Plugin classes can be injected with mock objects for unit tests
  - Plugin Certification

Integration Points

Core application can define a variety of integration points in which plugins can participate, such as:

- Menu Items (offered in view, coupled with actions or events)
- Service Registry (plugins can register their impls)
- Event Listeners
- View Extensions (plugins can decorate UI elements)
- Workflow Actions (asynchronous)
- State Transitions

Sample Plugin Integration

- menu item is registered in object selection context menu
- action and selected objects forwarded to registered service
- service analyses objects and annotates page context
- view decorator translates annotations into display options
- application renders page according to instructions

Plugin Design Guidelines

A Plugin should be an artifact containing

- a number of classes
- resources - property files, mappings, content
- assembly and deployment instructions
- versioned dependencies on components and APIs

Plugin classes should:

- only address a single concern (react to an action or event, provide view instructions, implement a service)
- implement a single interface defined by an Integration Point in the core application
- define their required dependencies and assume that these will be satisfied at runtime
- allow core to handle all cross-cutting concerns

Core API Dependencies

Plugins should depend on core API to extract information from the context of a request or event

```
public class DeleteObjectAction<T> implements RequestAction
{
   private PersistenceService<T> persistenceService;
   private ObjectProvider<HttpServletRequest, T>
objectProvider;

   public void processRequest(HttpServletRequest request) {
         T object =
      objectProvider.getObjectFromRequest(request);
         persistenceService.delete(object);
   }
}
```

Design and Development Processes

Long Term

- Project Plan - major release schedules, resource allocation
- Estimates - high level view of effort (days, months, weeks)
- Requirements Gathering
  - gain a clear view of use cases
  - write and organize feature cards
- Distributed Collaborations - plan communication frequency
- System and Acceptance Testing

Short Term

- Iteration Plan - several minor releases planned in advance
- Estimates effort in 1/2 day increments summarizing effort in each layer of the application, including unit tests
- Requirements - detail use case and identify corner cases
- Enforce Build Discipline
- Unit, Integration and Performance Testing

Recommended Technologies

Maven 2.0

Provides a standard way to build projects, a clear definition of what the project consisted of, an easy way to publish project information and a way to share artifacts across projects.

- makes the build process easy
- coherent site of project information
- consistent usage across all projects means no ramp up time for new developers coming onto a project
- superior dependency management
- a large and growing repository of libraries and metadata to use out of the box, and arrangements in place with the largest Open Source projects for real-time availability of their latest releases
- release management, distribution publication and source control integration

Spring Framework

Wide ranging framework for enterprise Java development. Includes abstraction layers for transactions, persistence frameworks, web application development and JDBC.

OSGi

Allow applications to be constructed from small, reusable and collaborative components... can be composed into an application and deployed. Provides the functions to change the composition dynamically on the device of a variety of networks, without requiring restarts. To minimize and manage coupling, provides a service-oriented architecture that enables components to dynamically discover each other for collaboration.

References and Useful Links

Martin Fowler Article on Dependency Injection

Dependency Inversion Principle

Sample Code

OSGI Technology