Frameworks and Components

Amit Shabtay

Frameworks

- "A reusable, semi-complete application that can be specialized to produce a custom application"
- "A set of cooperating abstract and concrete classes that makes a reusable design for a specific class of software"
- An Object-Oriented Reuse Technique
  - Design Reuse + Code Reuse

Comparison of Reuse Techniques

- Components are less abstract than frameworks
  - Frameworks are incomplete applications: They compile, but they don't run
  - Components are usually framework-specific
  - Framework generates an application while components comprise the application.
  - Framework encapsulate all data to generate the application.

Comparison of Reuse Techniques II

- Frameworks are less abstract than patterns
  - Include actual code
  - Specific to programming language
  - Specific to one application domain
  - Many patterns came from successful FWs
- Patterns are described as micro-architecture

A Tiny Example: Calculators

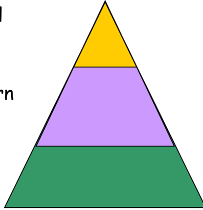
- interface Calculator
  - getValue(), compute(Operator o), clear(), undo()
  - Uses Command pattern, optionally Singleton
  - Remembers parentheses, can iterate on their tree
- interface Operator
  - Descendants: UnaryOperator, BinaryOperator
  - Concrete classes: Plus, Minus, Power, …
  - Acts as Command class, supports Composites
- interface VisualCalculator
  - Observer on Calculator, can display operators on buttons, can display current computation tree
- All are extendible, "Main" receives interfaces

Designing an OO Framework

1. Domain Knowledge
   - What applications is the framework for?
   - What is common to all of them?
2. Architecture
   - Biggest, most critical technical decisions
   - What is required besides classes?
3. Object-oriented design
   - Design Reuse: Patterns
   - Inversion of Control + Find right hooks

Framework as a Pyramid

- Architectural level
- Design pattern level
- Component repository



Domain Knowledge

- a.k.a. Analysis or Modeling
- Common "significant" decisions:
  - Major concepts of the modeled domain
  - Major operations
  - Use cases: How users do common tasks
- For example, a calculator
  - Concepts: unary operator, binary operator, current value, in-memory value, shift key
  - Operations: Clear, use operator, compute
  - Use case: Computing an average of *n* numbers

Architecture

- The set of significant decisions about the structure of software, the division to components and subsystems and their interfaces, and guidelines to composing them
- Common "significant" decisions:
  - Programming language, operating system, hardware
  - Use of major external libraries or applications
  - Physical Distribution, processes and threads
  - Main Concepts: Kinds of modules and interfaces
  - Communication and synchronization between modules
  - Security Model
  - Performance and scalability

For Example: JCA

- Java Cryptography Architecture (JCA, JCE)
  - Encryption, Digital Signatures, Key Management
  - Open for new algorithms, new implementations
- Main Concepts
  - Provider: provides implementations for a subset of the Java Security API, identified by name
  - Engine Classes: functionality for a type of crypto behavior, such as *Signature* and *KeyPairGenerator*
  - Factory Methods: static methods in engine classes that return instances of them for a given algorithm
  - Key Store: System identity scope

JCA II

- Generating a public/private key pair:

```
KeyPairGenerator keygen =
    KeyPairGenerator.getInstance("DSA", "MY_PROVIDER");
keygen.initialize(keySize, new SecureRandom(userSeed));
KeyPair pair = keygen.generateKeyPair();
```

  - Cast to *DSAKeyPairGenerator* is required to initialize it with algorithm-specific parameters (p,q,g)
- Generating a signature:

```
Signature sha = Signature.getInstance("SHA-1");
PrivateKey priv = pair.getPrivate();
sha.initSign(priv);
byte[] sig = sha.sign();
```

  - Provider is optional in *getInstance()*

JCA III

- Although implementations will usually be non-Java, they must be wrapped in Java classes
- Statically, add lines to java.security text file
  - Security.providerName.n = com.acme.providerPackage
  - n is the preference order of the provider, 1 is highest
- Providers can be managed dynamically too:
  - Class Security has addProvider(), getProvider()
  - Class Provider has getName(), getVersion(), getInfo()
- Providers must write a "Master class"
  - Specifies which implementations are offered by it
  - There are standard names for known algorithms

JCA IV: Summary

- So what does the architecture answer?
  - Domain Knowledge: What behavior (engine classes) should be supported at all?
  - How are different algorithms and different implementations defined and selected?
  - How should non-Java implementations be used?
  - How can an administrator configure a key store and a trusted set of providers and implementations?
  - How can commercial companies sell Java-compatible closed-source implementations of security features
- Not only classes and interfaces
  - Persistent key store, config files, non-Java code
  - Practical, management and economic considerations

Hooks

- Hook = Hotspot = Plug-point
  - Points where the FW can be customized
- Design issues requiring domain knowledge
  - How to find the right hooks?
  - Few or many hooks?
  - What should be the default behavior?
- Implementation alternatives
  - Template Method
  - Strategy or Prototype
  - Observer

Framework Colors

- White-Box Frameworks
  - Extended by inheritance from framework classes
  - Template Method, Builder, Bridge, Abstract Factory
  - Require intimate knowledge of framework structure
- Black-Box Frameworks
  - Extended by composition with framework classes
  - Strategy, State, Visitor, Prototype, Observer
  - More flexible, slightly less efficient
- Gray-box Frameworks
  - What usually happens in real life…

Framework Colors II

- Frameworks tend to evolve to being black-box
- AWT 1.0 had a white-box event model
  - Each visual component had an *handleEvent()* method
  - Each frame inherited and overrode it
  - The method was a long *switch* statement
- AWT 1.1 and Swing are black-box
  - Observer pattern: UI components publish events to registered listeners
- Why is black-box better?
  - Separation of concerns: better abstractions
  - Important for (automatic) code generation

Designing an OO Framework

1. Domain Knowledge
   - What applications is the framework for?
   - What is common to all of them?
2. Architecture
   - Biggest, most critical technical decisions
   - What is required besides classes?
3. Object-oriented design
   - Design Reuse: Patterns
   - Inversion of Control + Find right hooks

Application Domains

- System Infrastructure
  - Operating System Wrappers: MFC, MacApp
  - Communication Protocols: RMI
  - Database Access: ADO, JDO
  - Security: JCA, JSA
- User Interfaces
  - SmallTalk-80 is the first widely used OOFW
  - Swing, Delphi, MFC, COM…
  - Integrated with development environments

Application Domains II

- Middleware / Object Request Brokers
  - Object Request Brokers: CORBA, COM+, EJB
  - Web Services: .NET, Sun One
- Enterprise Applications
  - Enterprise = Critical to day-to-day work
  - Usually developed inside organizations
  - Notable Exception: IBM's San-Francisco
  - Telecomm, Manufacturing, Avionics, Finance, Insurance, Healthcare, Warehouses, Billing…

Framework Strengths

- Reuse, Reuse, Reuse!
  - Design + Code
- Extensibility
  - Enables the creation of reusable Components
- Enforced Design Reuse
  - An "Educational" Tool
- Partitioning of Knowledge & Training
  - Technical vs. Applicative Specialization

Framework Weaknesses

- Development effort
  - Generic frameworks are harder to design and build
  - They are also hard to validate and debug
- Maintenance
  - Does the FW or the app need to change?
  - Interface changes requires updating all apps
- Learning Curve
  - Unlike class libraries, you can't learn one class at a time
- Integratibility of multiple frameworks
- Efficiency
- Lack of standards

There's Big Money Involved

- All "big players" develop and sell FWs
  - So you must use our language (Swing)
  - So you must use our operating system (MFC)
  - So you must use our development tool (Delphi)
  - So you must use our database (Oracle)
- There's a component industry too
  - Companies that write and sell components
- Frameworks are an economic necessity
  - Unwise to develop UI, DB, ORB alone today