GoF Design patterns III

Week 10 Lecture
October 4, 2006

Agenda

- Architectural Analysis
- More GoF patterns
  - Failover to a local service when a remote service fails
  - Adapter + Proxy
Iteration 3 requirements

- **NextGen POS**
  - Provide failover to local services when the remote services cannot be accessed
  - Provide support for POS device handling
  - Handle credit payment authorization
  - Support for persistent objects
- **Monopoly Game**
  - Handle more types of squares: Lot, Railroad and Utility
    - Player may buy the square if it is not owned
    - Player may land on its own square, nothing happened
    - Player may pay a rent if it is owned by other players

Architecture Analysis

- A specialization of requirements analysis, with a focus on requirements that strongly influence the “architecture”
- Identify factors that should influence the architecture, and resolve them
- When do we start architecture analysis
  - Before the first development iteration
Variation and Evolution points

- Variation point – variations in the existing current system or requirements
  - Eg. Support multiple tax calculator interface
- Evolution point – speculative points of variation that may arise in the future, but which are not present in the existing requirements

Architectural analysis: important issues

- How do reliability and fault-tolerance requirements affect the design
- How do the licensing costs of purchased subcomponents affect profitability?
- How do the adaptability and configurability requirements affect the design?
- How does brand name and branding affect the architecture?
Common steps in architectural analysis

- Identify and analyze the non-functional requirements that have an impact on the architecture
  - Architectural factors
- For those requirements with a significant architectural impact, analyze alternatives and create solutions that resolve the impact
  - Architectural decisions

The science: identification and analysis of architectural factors

- Quality scenarios
  - Describe non-functional architectural factor in a measurable form
    - Eg. When the completed sale is sent to the remote tax calculator to add the taxes, the result is returned within 2 seconds “most” of the time, measured in a production environment under “average” load conditions
  - Avoid describing scenarios that will never be tested.
The science: identification and analysis of architectural factors

- Describing factors
  - In addition to quality scenario, identify the variation point as well as the evolution point

<table>
<thead>
<tr>
<th>Factor</th>
<th>Measures and quality scenarios</th>
<th>variability</th>
<th>Impact of factor</th>
<th>Priority</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recovery from remote service failure</td>
<td>When a remote service fails, reestablish connectivity with it within 1 minute of its detected re-availability, under normal store load in a production environment</td>
<td>Current flexibility- local client simplified services are acceptable until reconnection is possible Evolution – within 2 years, some retailers may be willing to pay for full local replication of remote services (high probability)</td>
<td>High on the large scale design</td>
<td>H</td>
<td>M</td>
</tr>
</tbody>
</table>

The art: resolution of architectural factors

- Recording architectural alternatives, decisions and motivation
  - Technical Memo of each architectural factors
- Priorities of decision making
  - Inflexible constraints, including safety and legal compliance
  - Business goals
    - Eg. Has qualities and features attractive to department stores in Europe
  - All other goals
Technical Memo Sample

**Solution Summary:** Location transparency using service lookup, failover from remote to local, and local service partial replication.

**Factors:**
- Robust recovery from remote service failure (e.g., tax calculation interrupts)
- Robust recovery from remote-product-specific database failures

**Solution:**
Achieve protected variation with respect to location of services using an adapter created in a localized context, where possibly failure is handled by a localized service layer. The adapter is responsible for ensuring that the local service is reachable if the remote service not is. It also provides a means to failover to a local service if the remote service is not reachable.

See also the reliability—Main/Service technical memo for the reliability aspects of this solution, because remote service implementation will vary at each installation.

To satisfy the quality contract of remote services, the remote services layer must support service-level agreements (SLAs) for the services, such as those expected for remote service operations, and route them to the local service where possible.

**Motivation:**
To ensure that the system remains available and delivers high-quality services to users, remote services must be able to deliver services reliably and efficiently. This requires careful design and implementation of the adapter layer to ensure that it can handle failures and maintain service availability.

**Unresolved Issues:**
- cost
- complexity

**Alternatives Considered:**
A "gold level" quality of service agreement with remote service providers is considered acceptable for the higher-level requirements. This design is expected to reduce maintenance costs and simplify the implementation of new services.

Technical Memo Sample (II)

**Solution Summary:** Purchase a tax calculator component.

**Factors:**
- Current tax rules must be applied, by law.

**Solution:**
Purchase a tax calculator with a licensing agreement to receive ongoing tax rule updates. Note that different calculators may be used at different installations.

**Motivation:**
Time-to-market, correctness, low maintenance requirements, and happy developers (see alternatives). These products are costly, which affects our cost-containment and product pricing business goals, but the alternative is considered unacceptable.

**Unresolved Issues:**
What are the leading products and their qualities?

**Alternatives Considered:**
Build one by the NextGen team? It is estimated to take too long, be error prone, and create an ongoing costly and uninteresting (to the company's developers) maintenance responsibility, which affects the goal of "happy developers" (purely, the most important goal of all).
The art: resolution of architectural factors

- Under and Over-engineering
  - Fail to resolve important future “evolution points” might result in expensive re-engineering
  - Handling too much distant future “evolution points” is costly and can make the architecture too complex
  - “Prediction is very difficult, especially if it is about the future!”

- Basic architectural design principles
  - Low coupling
  - High cohesion
  - Protected variation (interfaces, indirection, service lookup, and so forth)
Separation of Concerns and Localization of Impact

- Cross-cutting concerns
  - Mixing application logic with database logic
  - Factors out persistence support and security support into separate “things”
- Large-scale techniques to achieve separation of concerns
  - Modularize the concern into a separate component
  - User Decorators
  - Use post-compilers

Themes in architectural analysis

- “architectural” concerns are especially related to non-functional requirements
- Architectural concerns involve system-level, large-scale and broad problems that may affect large-scale or fundamental design decisions
- There are interdependencies and trade-offs in addressing architectural concerns
- Architectural analysis is useful in generating and evaluating alternative solutions
More object design with GoF patterns

- NextGen POS requirements
  - Local caching
  - Failover to a local service when a remote service fails

Failover to Local Services

- Architectural factors
  - Robust recovery from remote service failure (e.g., tax calculator, inventory)
  - Robust recovery from remote product (e.g., descriptions and prices) database failure

- Solutions
  - Cache Proxy + Adapter + ServicesFactory to handle remote service failure
  - Redirection Proxy to handle reconnection with the remote service
Recap: current design involving product catalogue

1. desc = getProductDesc(id)

2. makeLineItem(desc, qty)

1.1: desc = get(id)

2.1: create(desc, qty)

2.2: add(sl)

1.1. desc = get(id)

2.1.1: create(id, price, description)

2.1.2*: put(id, pd)

1.2: loadProdSpecs()

1.2.1*: create(id, price, description)

1.2.2*: put(id, pd)

pass a reference to the ProductCatalog to the Register, so that it has permanent visibility to it

create an empty collection object

create an empty collection object

the * in sequence number indicates the message occurs in a repeating section

add the newly created SalesLineItem instance to the List

by Creator

by Controller

by Creator

by Expert

by Creator

by Expert

by Creator
Recap: current design involving product catalogue

- First iteration ProductCatalog code <page 377>

```java
public class ProductCatalog {
    private Map<ItemID, ProductDescription> description = new HashMap();

    public ProductCatalog() {
        // sample data
        ItemID id1 = new ItemID(100);
        ItemID id2 = new ItemID(200);
        Money price = new Money(3);

        ProductDescription desc;
        desc = new ProductDescription(id1, price, "product 1");
        description.put(id1, desc);
        desc = new ProductDescription(id2, price, "product 2");
        descriptions.put(id2, desc);
    }
}
```

Recap: current design to handle external service

- CATaxAdapter
- MATaxAdapter
- ITaxCalculatorAdapter
- SAPAccountingAdapter
- GreatNorthernAccountingAdapter
- IInventoryAdapter
- ICreditAuthorizationService Adapter
Recap: current design to handle external service

```java
ServicesFactory
accountingAdapter : IAccountingAdapter
inventoryAdapter : IInventoryAdapter
taxCalculatorAdapter : ITaxCalculatorAdapter

getAccountingAdapter() : IAccountingAdapter
getInventoryAdapter() : IInventoryAdapter
getTaxCalculatorAdapter() : ITaxCalculatorAdapter
```

note that the factory methods return objects typed to an interface rather than a class, so that the factory can return any implementation of the interface.

```java
if ( taxCalculatorAdapter == null )
{
    // a reflective or data-driven approach to finding the right class: read it from an external property
    String className = System.getProperty( "taxcalculator.class.name" );
    taxCalculatorAdapter = (ITaxCalculatorAdapter) Class.forName( className ).newInstance();
}
return taxCalculatorAdapter;
```

Communication with remote product database

- **Issues:**
  - We can reuse the current adapter and factory design
    - We use adapters to hide the underlying various DBMS used by various store
  - We need to have a local cache of product data for performance and reliability consideration
    - **Cache Proxy** provides temporary storage of the results of expensive target operations so that multiple clients can share the results.
Communication with remote product database

Solution

- The ServicesFactory will always return a proxy to a local product information service.
- The local cache proxy will implement the responsibilities of the local services.
- The local service is initialized to a reference to an adapter to the remote product service.
- If the local service finds the data in its cache, it returns it; otherwise, forwards the request to the adapter for the external service.

Two levels of client-side cache

- The in-memory ProductCatalog object will maintain an in-memory collection (Hashtable) of some (1000) productDescription objects that have been retrieved from the product information service.
- The local products service will maintain a larger persistent cache that maintains some quantity of product information (1 or 100MB of file space).
Proxy for product information

Implements the adapter interface, but is not really an adapter for a second component. Rather, it itself implements the local service function.

Initialization of the product information service

The local service gets a reference to the adapter for the external service.
Starting the collaboration with the products service

1: ps = getDescription(id)
2: makeLineItem(ps, qty)
3: put(id, ps)

Descriptions: Map<ProductDescription>

Remote service: BigWebServiceProductsAdapter

The design can handle various Product services. E.g. the external Product service can be changed from a database to some other web services.
Collaboration with the persistence subsystem

Collaboration with the O/R mapping persistence subsystem requires indicating the type of object to retrieve, and its ID. This subsystem is relatively generic—it is not especially designed just for the NextGen POS application.

“ProductDescription.class” is Java to specify the object type.

Proxy (structural pattern)

- Name: Proxy
- Problem: direct access to a real object is not desired or possible.
- Solution: add a level of indirection with a surrogate proxy object that implement the same interface as the subject object, and is responsible for controlling or enhancing access to it.
Proxy pattern: structure

```
Proxy
realSubject : ISubjectInterface
foo()
{
... whatever
realSubject.foo()
... whatever
}

subject actually references an instance of Proxy, not RealSubject

subject : ISubjectInterface
foo()
{
... pre-processing
realSubject.foo()
... post-processing
}

client actually references an instance of RealSubject
```

Various Proxy Patterns

- **Cache Proxy**
  - provides temporary storage of the results of expensive target operations so that multiple clients can share the results
  - Most widely used case is Remote Proxy in distributed application
    - If a client need to access a remote service, it called upon a local client-side object (“stub”) to communicate with the remote service.
      - This client-side stub is a local proxy, or representative for a remote object
      - It hides all network communication details from the client, the client can access remote services in the same way as it accesses a local service
    - Most Web-services implementations use this pattern
Caching strategies

- Lazy initialization
  - The cache fills slowly as external product information is retrieved
- Eager initialization
  - The cache are loaded during the \texttt{StartUp} use case
- Can use strategy pattern to experiment with alternatives.

Threads in the UML

- Stale cache problem
  - \texttt{LocalProduct} objects should communicate regularly with the remote services to update its cached local copy
  - Add a thread to \texttt{LocalProduct} object, this thread will sleep for some time, wake up, query the remote product services to update its local replication, then go back to sleep again
Handling failure

- Local persistent cache is usually a partial copy of the whole product catalog
- How to handle the situation where there isn’t a local cache hit and access to the external product services fails
  - Alternative business solutions: cashier manually enter the price and description; OR cancel line item entry.

Failover to local services with a Proxy

- Failover to a local service for the product information was achieved by inserting the local service in front of the external service
  - The local service (local copy) is always tried first
  - Sometimes it is not desirable
    - We should try remote service, if it is not available, try a local service (always as a backup)
      - E.g., Posting of sales to the accounting service
Redirection Proxy variation

- Also know as **Failover Proxy**
- Application in POS case study regarding external accounting service
  - A **postSale** message is sent to redirection proxy, as if it was the actual external accounting service
  - If the redirection proxy fails to make contact with the external service, it then redirect the **postSale** to a local service, which locally stores the sales for forwarding to the accounting service when it is active

Redirection Proxy in POS
Proxy vs. Adapter

- Both provide delegation services to the real object
- Proxy has the same interface with the object being represented
- Adapter has different interface with the object being represented