Ownership in Design Patterns

Master's Thesis
Final Presentation
Stefan Nägeli
07.02.06
Overview

- Status Quo
- Pattern Overview
- Encountered Problems applying UTS
- Pros and Cons compared to other systems
- UTS Feature Requests
- Conclusion
- Remaining Work
Status Quo

- Finished analyzing all patterns in the GoF Design Patterns book.
- Ownership in a Swing GUI application. Core design encapsulates 7 patterns.
- Pattern usage and ownership in the Swing Framework.
## Overview - Creational

<table>
<thead>
<tr>
<th></th>
<th>UTS</th>
<th>Ownership Types</th>
<th>Ownership Domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract Factory</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Builder</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Factory Method</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Prototype</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Singleton</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

- **✗** = applying ownership enhances pattern and works fine
- **✗** = ownership typing leads to problems or has few benefits
- **✗** = ownership typing is not possible or has no benefits
# Overview - Structural

<table>
<thead>
<tr>
<th></th>
<th>UTS</th>
<th>Ownership Types</th>
<th>Ownership Domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adapter</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Bridge</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Composite</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Decorator</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Facade</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Flyweight</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Proxy</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

- ✗ = applying ownership enhances pattern and works fine
- ✗ = ownership typing leads to problems or has few benefits
- ✗ = ownership typing is not possible or has no benefits
### Overview - Behavioral

<table>
<thead>
<tr>
<th></th>
<th>UTS</th>
<th>Ownership Types</th>
<th>Ownership Domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chain of Resp.</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Command</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Interpreter</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Iterator</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Mediator</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Memento</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Observer</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>State</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Strategy</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Templ. Method</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Visitor</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

**x** = applying ownership enhances pattern and works fine  
**x** = ownership typing leads to problems or has few benefits  
**x** = ownership typing is not possible or has no benefits
### Implementation enhancements

<table>
<thead>
<tr>
<th>Enhances implementation</th>
<th>No enhancements</th>
<th>Strongly depends</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract Factory</td>
<td>Singleton</td>
<td>Adapter</td>
</tr>
<tr>
<td>Builder</td>
<td>Decorator</td>
<td>Composite</td>
</tr>
<tr>
<td>Factory Method</td>
<td>Observer</td>
<td>Facade</td>
</tr>
<tr>
<td>Prototype</td>
<td>Visitor</td>
<td>Strategy</td>
</tr>
<tr>
<td>Bridge</td>
<td>Command</td>
<td>Interpreter</td>
</tr>
<tr>
<td>Flyweight</td>
<td>Chain of Resp.</td>
<td></td>
</tr>
<tr>
<td>Iterator</td>
<td>Template Method</td>
<td></td>
</tr>
<tr>
<td>Mediator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memento</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Encountered Problems

using the Universe type system

- No ownership transfer
- No concept of friends
- No multiple contexts
- Readonly reference leaking
1. No ownership transfer

Object creation determines the ownership context for life-time.

During an object's life-time it may need to reside in different contexts, but only in one at a time.

- Delegating object creation.
- The desired ownership structure is not in accordance with the creation order.
- Need to change context upon performing an operation
Problem occurrences

- Abstract Factory
- Factory Method
- Prototype
- Adapter
- Composite
- Visitor
Example: Abstract Factory

- Garage
  - +createTire()

- Car
  - createTire

- Tire
  - transfer context
Example: Visitor

[Diagram of a visitor pattern with Client, Element, and Visitor classes and their interactions.

- Client
- Element
- Visitor

Arrows indicate
- accept
- visit
- query

Transfer context arrow from Visitor to Element]
2. No concept of friends

Only peer objects may alias each other in a read/write manner and there are no exceptions.

Some objects reside in a single context for life-time but need to be referenced not only by peer but also by a carefully chosen group of other objects.
2. No concept of friends (2)

- Tightly coupled objects in terms of usage, but not related in terms of ownership.
- No notion of a global context.
- No possibility to share objects among “friends”.
- No r/w reference to owner
Problem occurrences

- Singleton
- Builder
- Observer
- State
- Strategy
- Command
- Chain of Responsibility
- Composite
Example: Singleton

Diagram showing the Singleton pattern with a root node, a global node containing a Singleton, and two other nodes labeled A and B.
Example: Observer

Application

AccountManager

Account
-amount : Double

MainWindow

LabelAmount
-text : String

attach

amountChanged
3. No multiple contexts

All objects owned by the same owner are automatically located in the same context.

*There is no way to declare multiple ownership contexts for the same owner.*

- No explicit enforcement of architectural constraints.
Problem occurrences

- Mediator
Example: Mediator
4. Readonly reference leaking

Each object may at least have a readonly reference to every other object in the system.

*Leaking of readonly references is possible.*

- Certain patterns would benefit from also preventing readonly references.
- \( x \ast \text{rep} \rightarrow \text{readonly} \)
  \( x \ast \text{rep\_strict} \rightarrow \text{forbidden} \)
Problem occurrences

- Proxy
- (Facade)
- (Bridge)
Example: Proxy

Guard
- secret : Secret
+ askForSecret()
+ leak() : Secret

Secret
+ getSecret()

Intruder
leak

leaking readonly reference
Pros and Cons of the UTS

compared to other ownership systems
Pros

- Simplicity and intuitive usage
- Clear, well-defined concept
- Notion of readonly references
Cons

- No global accessible context
- No support for declaring multiple contexts
- No aliasing exceptions (e.g. links)
- No read/write access to owner
- No mechanism to prevent readonly aliasing
Feature Requests for the UTS

- Ownership transfer
- *Global* contexts
- Possibility to declare objects as *friends*
- Possibility to suppress readonly references
- Possibility to declare multiple contexts
Conclusion

- Ownership succeeds to enhance certain pattern implementations.
- Ownership still lacks the necessary flexibility to cope with many common application designs / scenarios.
- To support most SE best-practices it might get necessary to give up static type checking in favor of being able to type more designs.
Remaining Work

- Finish review of ownership feasibility in the Swing demo application and framework.
- Correct and improve the report
Swing Demo Application

• Simple accounting application

• Core design encapsulates 7 patterns
  - Composite
  - Command
  - Visitor
  - Observer
  - Singleton
  - Strategy
  - Decorator

• Domain: Journal, Booking, Account, Group
Questions?

Comments?