

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

	UTS	Ownership Types	Ownership Domains
Abstract Factory	x	x	x
Builder	x	x	x
Factory Method	x	x	x
Prototype	x	x	x
Singleton	x	x	x

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

Overview - Structural

	UTS	Ownership Types	Ownership Domains
Adapter	x	x	x
Bridge	x	x	x
Composite	x	x	x
Decorator	x	x	x
Facade	x	x	x
Flyweight	x	x	x
Proxy	x	x	x

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

Overview - Behavioral

	UTS	Ownership Types	Ownership Domains
Chain of Resp.	x	x	x
Command	x	x	x
Interpreter	x	x	x
Iterator	x	x	x
Mediator	x	x	x
Memento	x	x	x
Observer	x	x	x
State	x	x	x
Strategy	x	x	x
Templ. Method	-	-	-
Visitor	x	x	x

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

Enhances implementation	No enhancements	Strongly depends
Abstract Factory	Singleton	Adapter
Builder	Decorator	Composite
Factory Method	Observer	Facade
Prototype	Visitor	Strategy
Bridge	Command	Interpreter
Flyweight	Chain of Resp.	
Iterator	Template Method	
Mediator		
State		
Memento		

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.

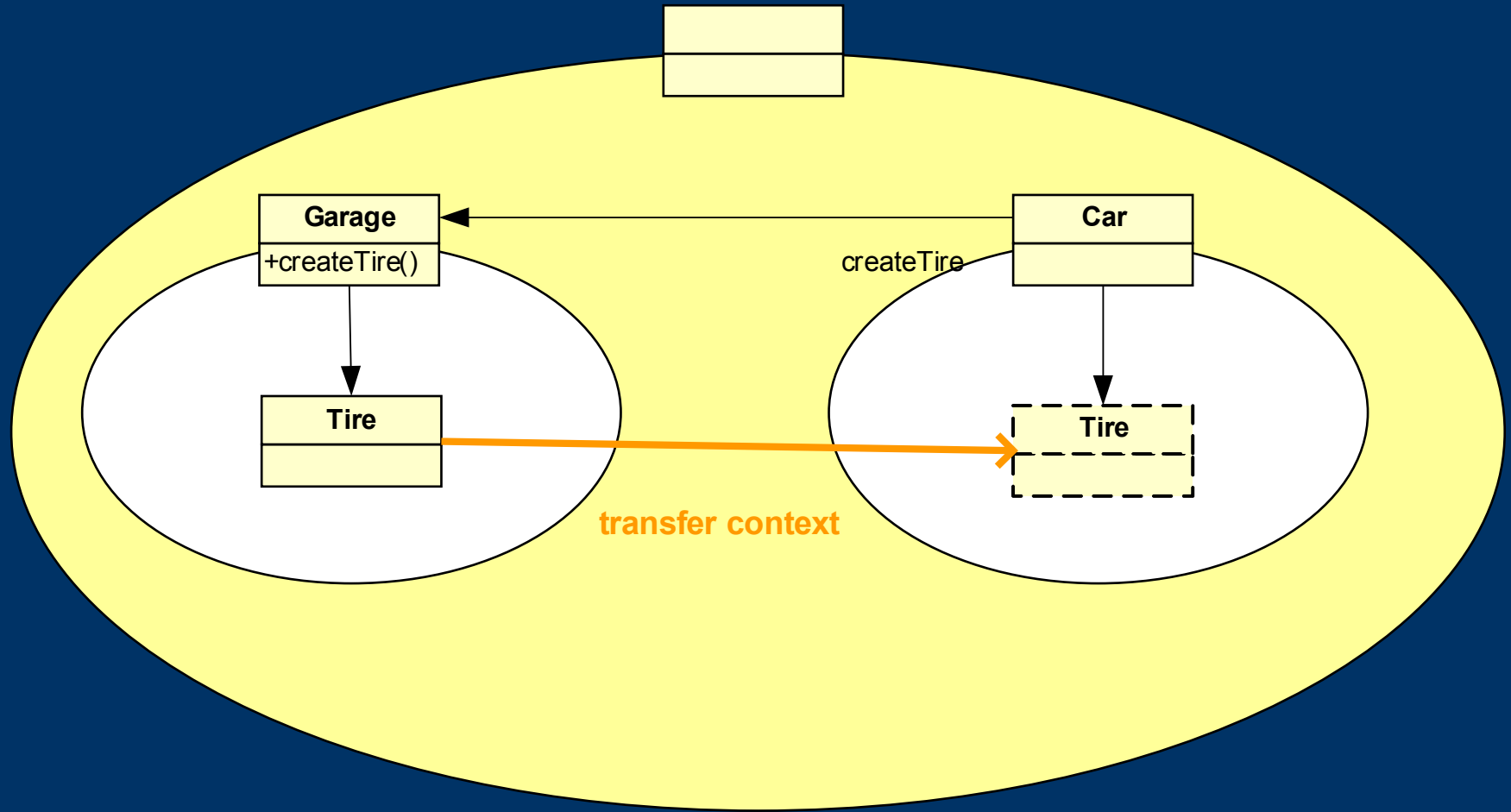
During an object's life-time it may need to reside in different contexts, but only in one at a time.

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

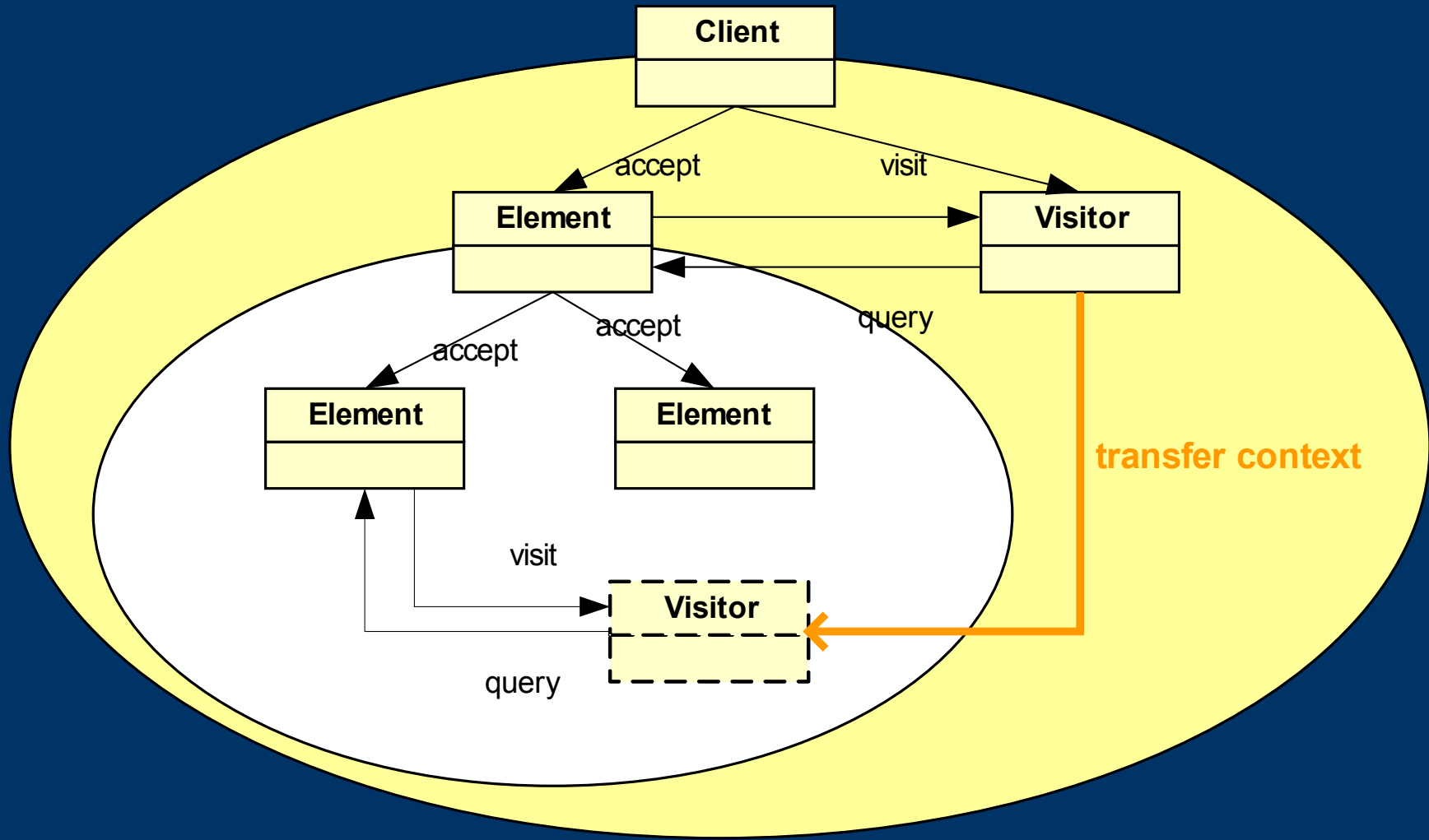
Problem occurrences

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

Example: Abstract Factory



Example: Visitor



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.

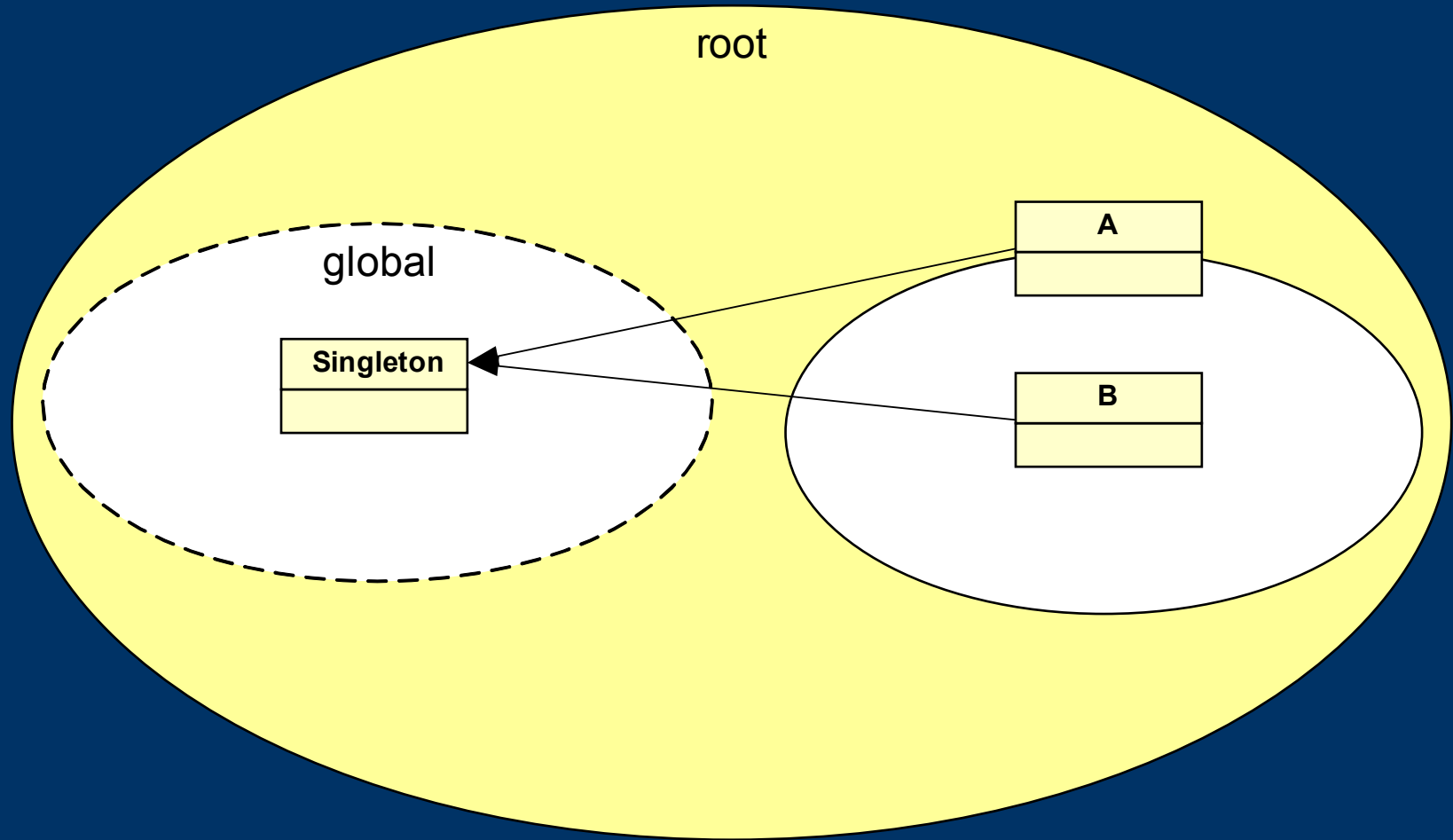
2. No concept of friends (2)

- 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
-
-

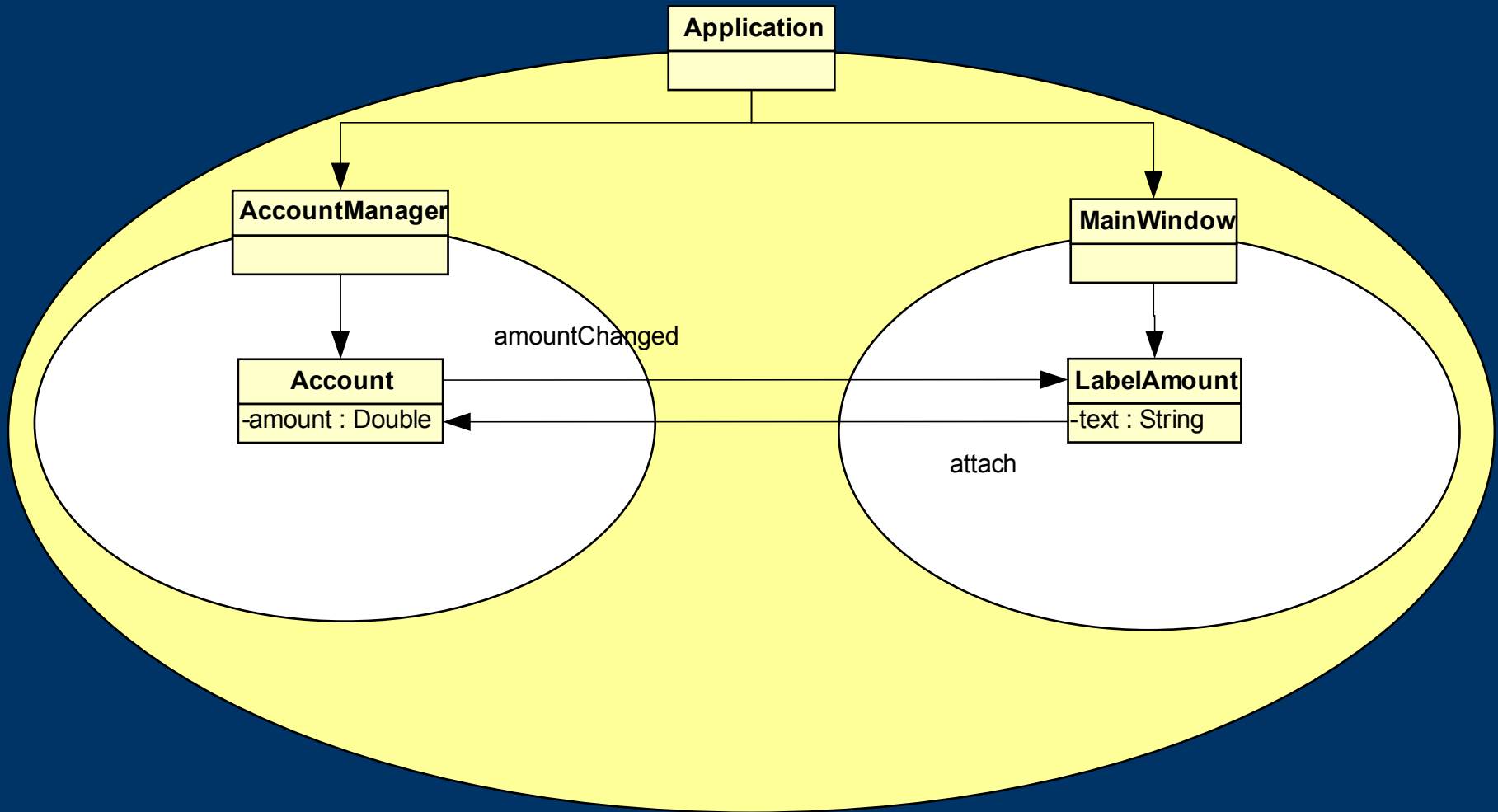
Problem occurrences

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

Example: Singleton



Example: Observer



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.

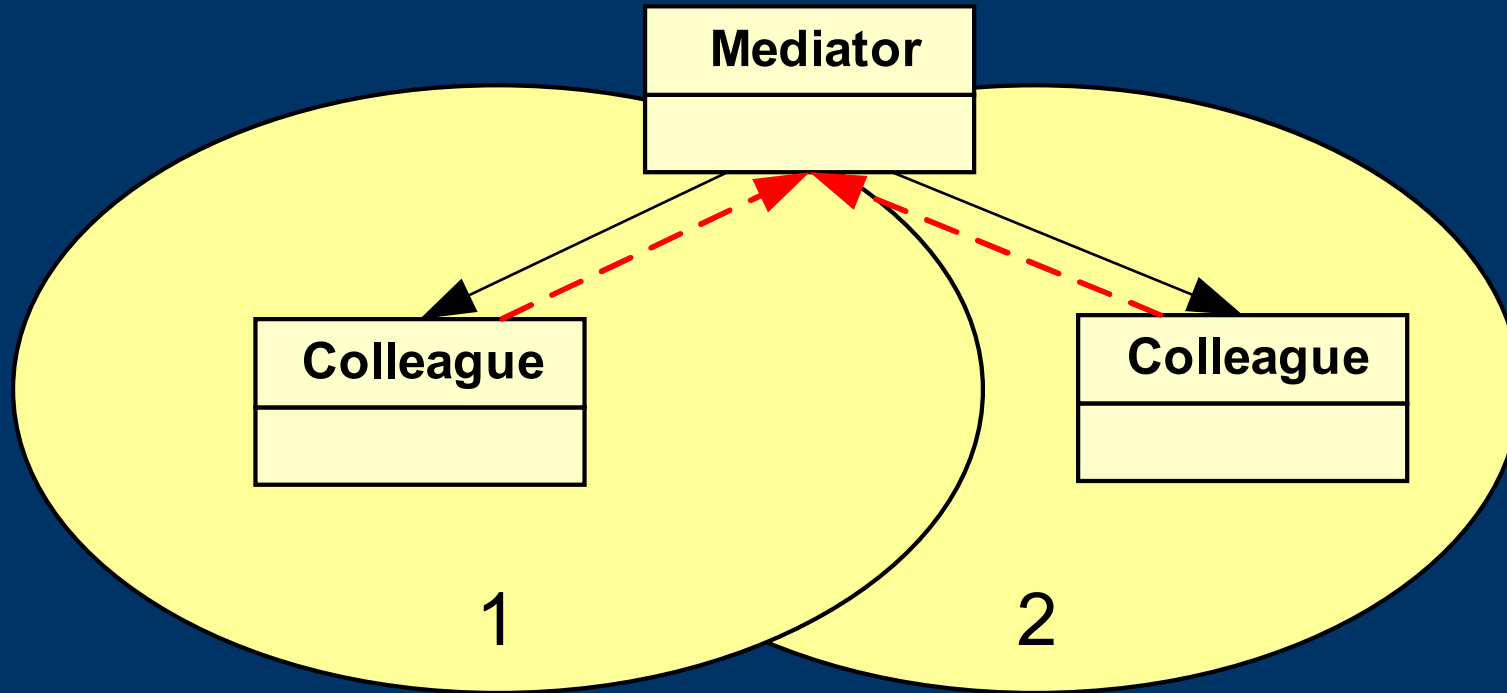
- No explicit enforcement of architectural constraints.
-
-

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.

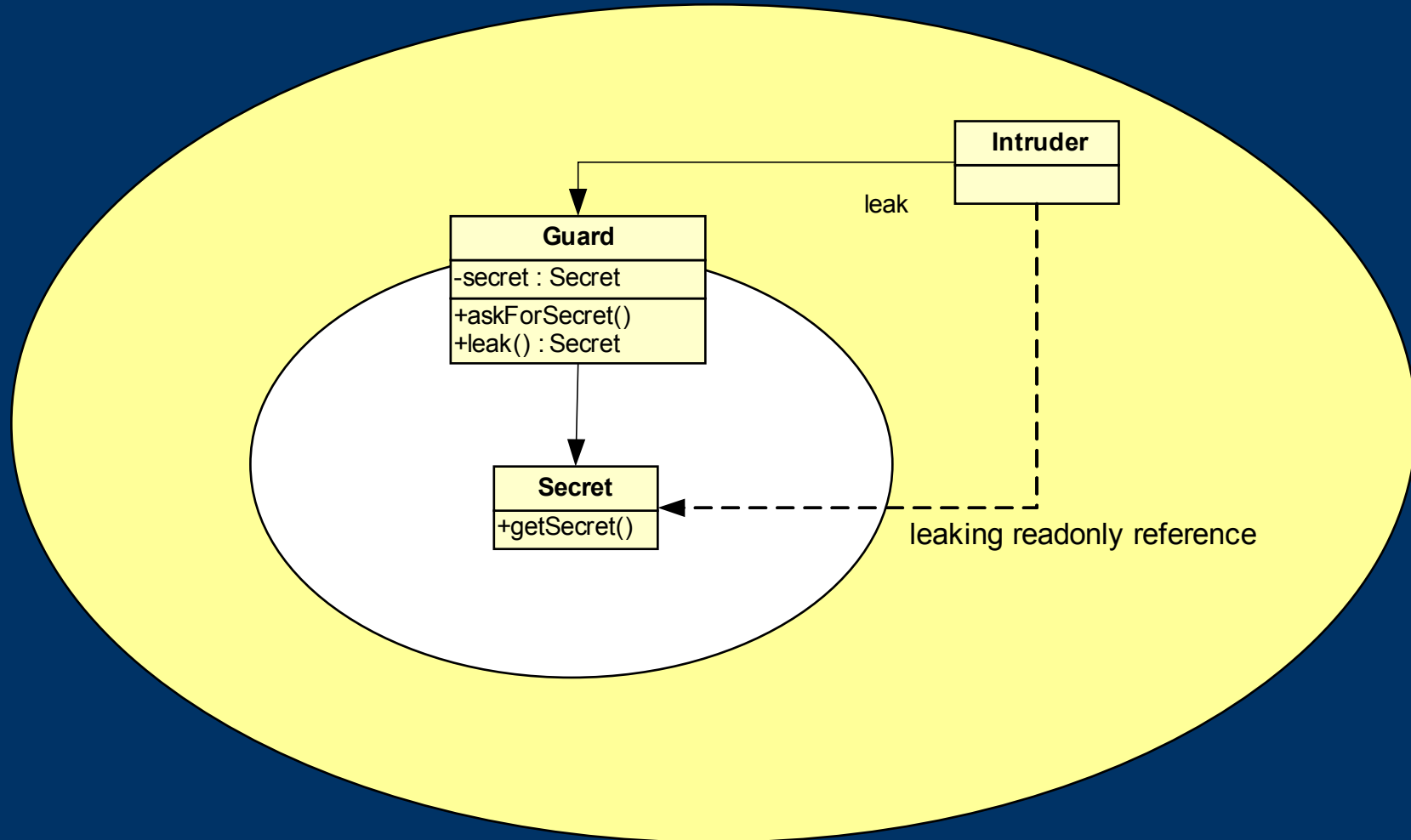
Leaking of readonly references is possible.

- Certain patterns would benefit from also preventing readonly references.
 - `x * rep -> readonly`
`x * rep_strict -> forbidden`
-
-

Problem occurrences

- Proxy
 - (Facade)
 - (Bridge)
-
-

Example: Proxy



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`
-
-

Screenshot

Application

Commands

Assets

- Assets 1555.0
 - Fluessige Mittel 175.0
 - Kassa 175.0
 - Konti 150.0
 - Post 25.0
 - Bank 125.0
 - Mobilien 425.0
 - Debitoren 5.0
 - Immobilien 800.0

Liabilities

- Liabilities 1555.0
 - FK 1055.0
 - Hypothek 600.0
 - Kredit 400.0
 - Darlehen 50.0
 - Kurzfrist. Kredit 5.0
 - EK 500.0
 - Eigenkapital 500.0

Journal

Date	From	To	Amount
6.1.2006	Post	Kurzfrist. K...	123
6.1.2006	Post	Mobilien	25
6.1.2006	Kassa	Bank	25

Date: From: To: Amount:

Questions?

Comments?



