Architectural choices when building COM(+) Applications

Astrid Hackenberg
class-a
astrid.hackenberg@class-a.nl

Agenda

- Application problems
- Design Patterns
- Scalability
- Performance
- Integration
- Availability

Application Problems

1. My application does not scale
2. My application does not perform
3. My application does not cooperate with my existing investments
4. My application is not always available

Design Patterns

"Patterns complement general but problem-independent analysis and design methods with guidelines for solving specific and concrete problems."

What is a Pattern ?

- Description of a design structure
  - Components & Responsibilities & Relationships & Collaborations
- Problem – Solution pair
  - for a recurring problem
  - solution flexible for problem variants

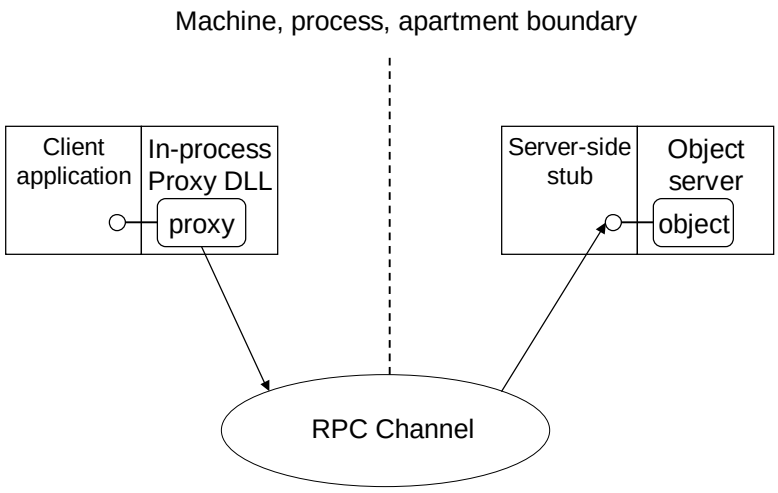
pat·tern (păt′ərn)
n.

1. a. A model or an original used as an archetype.
   2. A person or thing considered worthy of imitation.
2. A plan, diagram, or model to be followed in making things.
3. A representative sample; a specimen.

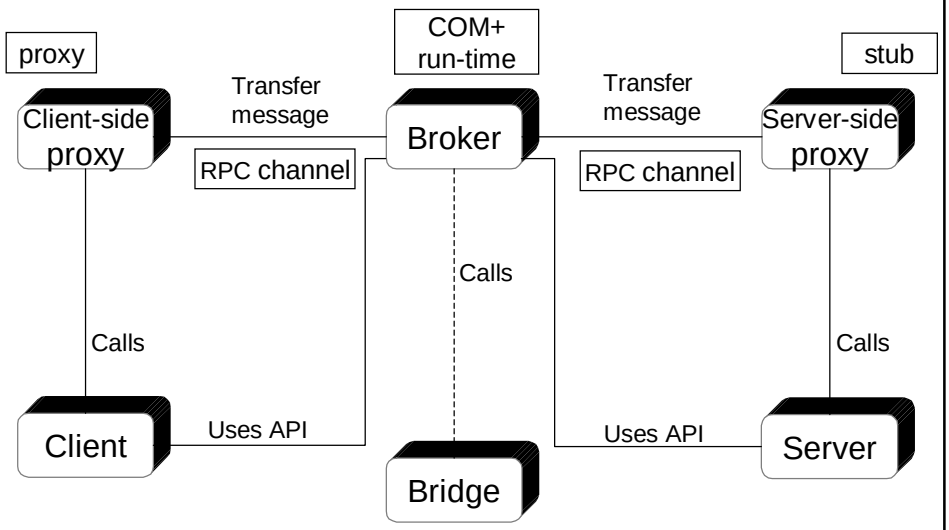
Patterns Applied

- Many Patterns in Microsoft Platform Architecture
  - Broker
  - Observer
  - Model-View-Controller
  - Counted Body
  - Proxy
  - Adapter
  - Etc.

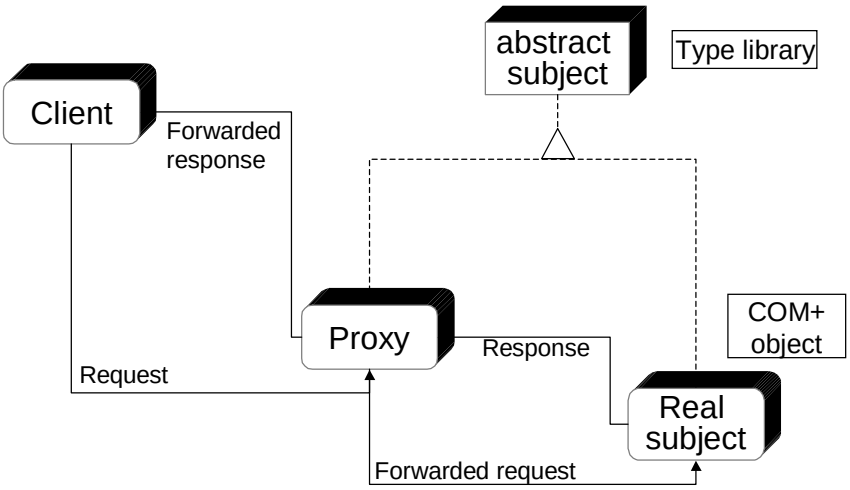
COM+ Remoting Architecture



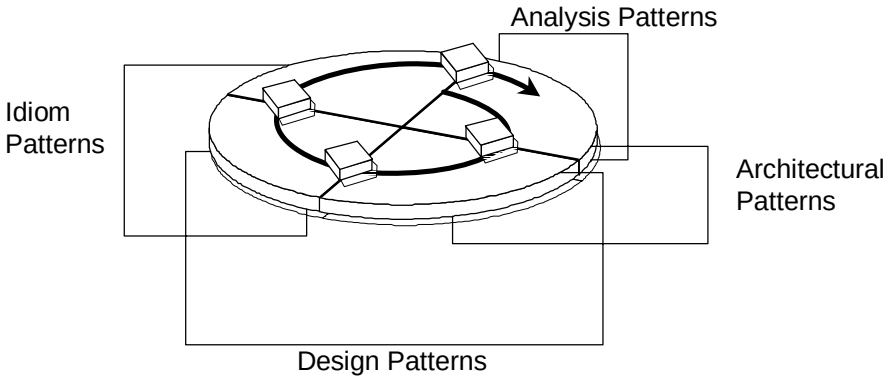
Broker: Architectural Pattern Applied



Proxy: Design Pattern Applied



Types of Patterns



Pattern Elements

Name	Handle to the pattern
Context	All (known) design situations where a problem can occur
Problem	Set of forces (system aspects) repeatedly arising in the context, e.g. - requirements - constraints - desirable properties
Solution	How to balance the forces by - static aspects (= system architecture): components, relationships, responsibilities - dynamic aspects (= run-time behavior): collaboration, communication
Consequences	Results of applying the pattern - benefits - trade-offs

Describing Patterns

- Consistent format
- Description template
  - GoF (Gang of Four)
    Name; Intent; AKA; Motivation; Applicability; Structure; Participants; Collaborations; Consequences; Implementation;Sample Code; Known Uses; Related Patterns
  - Your Template
- Diagrams
  - CRC-cards (Component,Responsibilities,Collaboration)
  - OMT-Object Model (Components,Interfaces,Relationships)
  - Object Interaction (Object Message Sequence Chart)

Why use Patterns ?

- Reuse experience
- Reuse successful designs
- Common language to communicate about software design (decisions)
- Document software architecture

Steps to selecting a Pattern

1. Specify the problem (sub-problems)
2. Select the pattern category
3. Select the problem category
4. Compare pattern problem descriptions
5. Compare pattern consequences
   (most benefits, least trade-offs)
6. Select the best pattern variant

Application Problems

1. My application does not scale
2. My application does not perform
3. My application does not cooperate with my existing investments
4. My application is not always available

My Problem Pattern template

- Name
- Also Known As
- Background
- Symptoms
- Solution(s)
- Usable Patterns

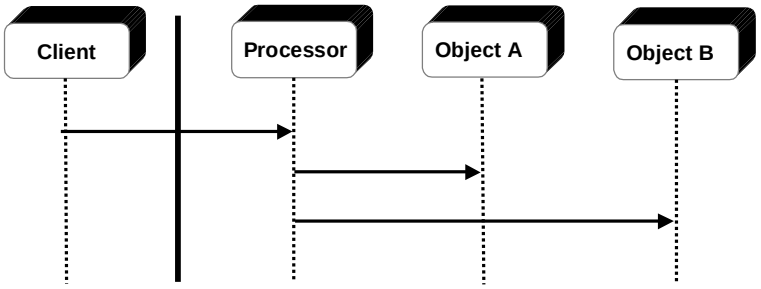
1. My Application does not Scale

- *a.k.a. "MS technology does not scale"*
- Background
  - By increasing the number of transactions, users,… my application behaves differently.
- Symptoms
  - Many network roundtrips
  - Many databases calls
  - Heavy memory load
  - Many object instances alive during a transaction

1. My Application does not Scale

- Solution
  - Revisited Scalability:
    - By adding more resources (memory, disk, network) the application should scale granular.
  - Use a N-tier application architecture
  - Minimize network roundtrips (connection-less programming)
  - Minimize database roundtrips (stateless programming)
  - Shorten transactions!
- Usable Patterns
  - Processor (Façade)
  - Flyweight

Processor pattern



Client makes single method call over network

Processor object on server makes method calls to other objects. It's main purpose is reducing the number of expensive calls the client need to make over the network.

2. My application does not perform

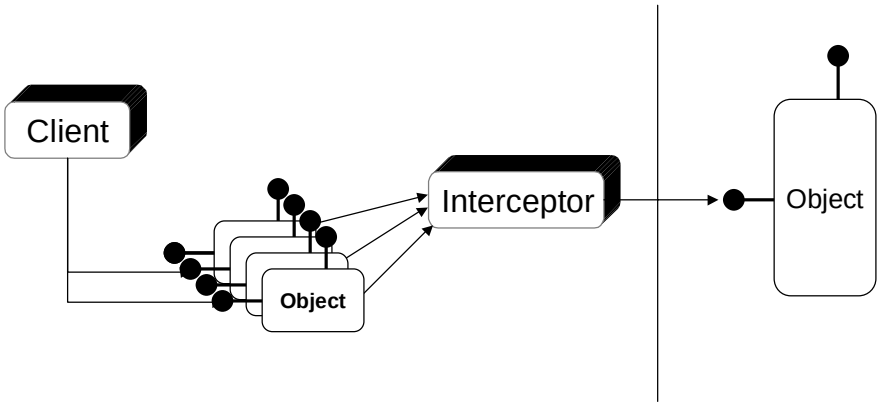
- *a.k.a. "Visual XXX is a toy"*
- Background
  - The application does not meet the expected end-to-end performance figures as defined.
- Symptoms
  - Object Oriented design
  - VARIANT
  - Database locks and blocks
  - State only available at database level
  - Discussions about implementation language

2. My application does not perform

- Solution
  - Define realistic performance goals
  - Think about Interface Design. Do not go into a religious OO war…
  - Use interception at client level
  - Provide state management
  - Shorten Transactions to avoid locking and blocking at database level
- Usable Patterns
  - Interceptor
  - Wrapper (Adapter)

Interceptor pattern

- Pre- and post processing of calls
- Interceptor must know about your interfaces
- Fundament of COM+ Runtime



Acquire late – Release early

- Acquire database connection late, let the pool do his work!
- Release as soon as possible!
- Do work always in the same sequence to avoid deadlocks
- Example

```
do work
OpenConnection
      do database action…(select, insert, etc)
Close Connection
handle results
```

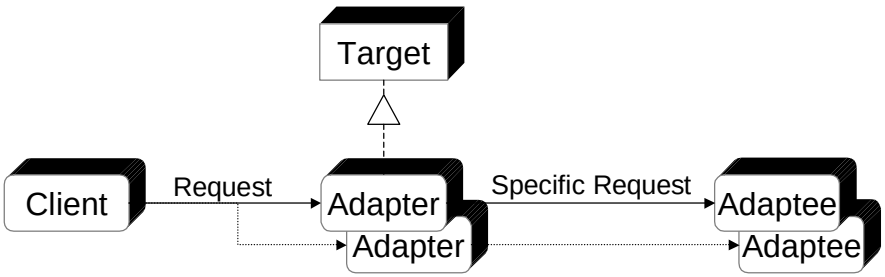
3. My application does not cooperate with my existing investments

- *a.k.a. "MS is proprietary"*
- Background
  - Existing applications should work with currently working applications. Integration is not a standard feature.
- Symptoms
  - Application islands
  - Handwork
  - Partly manually batch processes

3. My application does not cooperate with my existing investments

- Solution
  - Integrate at data and transaction level
  - Design for integration; it is not a separate task after the project
  - Use interfaces
  - Key technologies : Message Queuing and XML
  - Microsoft technologies: OLE-DB, XML (BizTalk Server), MSMQ, COM-TI
- Usable Patterns
  - Adapter
  - Bridge
  - Façade

Adapter pattern



- Makes components with incompatible interfaces cooperate.
- Adapter converts from the interface that the client uses to the interface that the existing application has and passes on the request.
- BizTalk

4. My application is not always available

- *a.k.a. "Microsoft technology is not enterprise ready"*
- Background
  - Application should be available 24hours/7 days week. An up-time of 99.9999% is required…
- Symptoms
  - System crashes
  - Expensive hardware purchases
  - High deployment costs

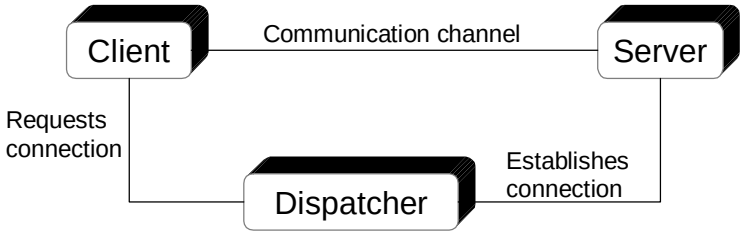
4. My application is not always available

- Solution
  - Revisit “Availability”. What do we mean with availability?
  - Again, availability is an design issue. Not something during deployment!
  - Use Components (hardware industry)
  - Design for machine independency
  - Use Message Queuing
  - Use Load Balancing when appropriate
- Usable Patterns
  - Client – Dispatcher – Server
  - Mediator

Availability – Load Balancing

- Load Balancing
  - Increases Availability
  - Increases Scalability
- Design components for Load Balancing
- No hardware/ system dependencies
- Think about WLBS vs. CLBS

Client-Dispatcher-Server pattern



- Provides location transparency
  - Server registers itself with dispatcher.
  - Client asks dispatcher for a communication channel to server.
  - Dispatcher establishes communication link and returns channel to client.

Take-a-ways…

- Manage expectations
- Use design patterns where appropriate
- Design for Scalability, Performance, Integration and Availability
- Focus on design, instead of technology

Some Books…

- Design Patterns (GoF)
  - Addison-Wesley, ISBN 0-201-63361-2
- A System of Patterns
  - Wiley, ISBN 0-471-95869-7
- Pattern Hatching
  - Addison-Wesley, ISBN 0-201-43293-5