Programação de Sistemas Distribuidos

Paulo Gandra de Sousa

psousa@dei.isep.ipp.pt

Mestrado em Engenharia Informática

DEI/ISEP

Disclaimer

- Parts of this presentation are from:
  - Paulo Sousa (PARS)
  - Ron Jacobs (ARC01)

Today's lesson

- Design Patterns
- Patterns for distributed Systems
- Service Orientation

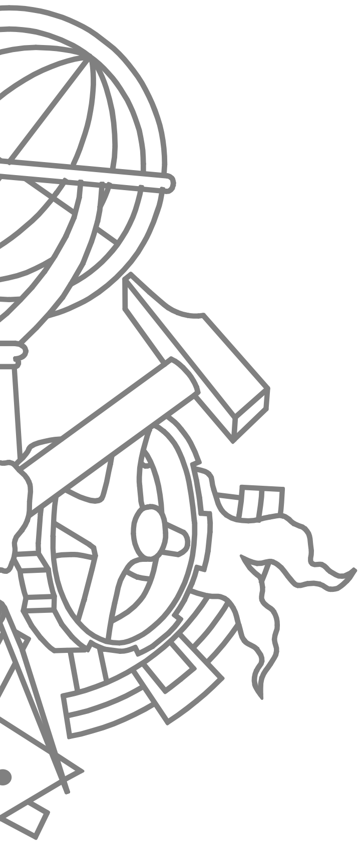
DESIGN PATTERNS

What is a Pattern?

Each pattern describes a **problem** that **occurs over and over** again in our environment and then describes the **core of the solution** to that problem in such a way that you can **use this solution a million times** over **without ever doing it the same way twice**.

Christopher Alexander

What is a design Pattern?

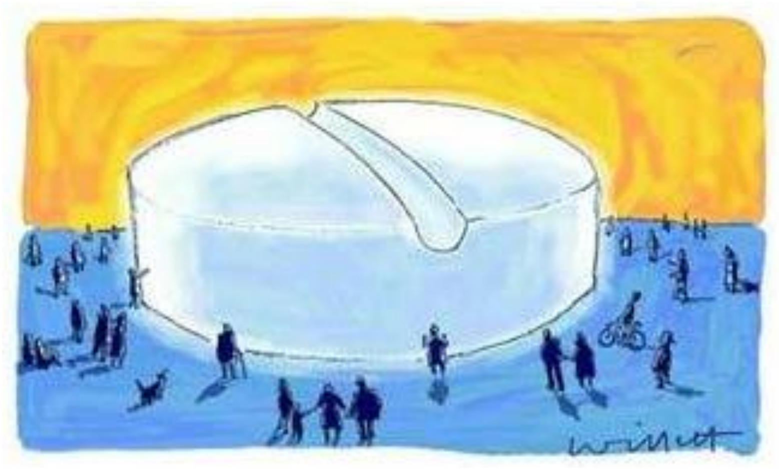


A design pattern names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a **reusable object-oriented design**.

Design Patterns-Elements of Reusable Object-oriented Software, Gamma et al. (Gang of Four)

What a pattern is not

- A miracleous receipt



source: British Medical Journal

What is a pattern

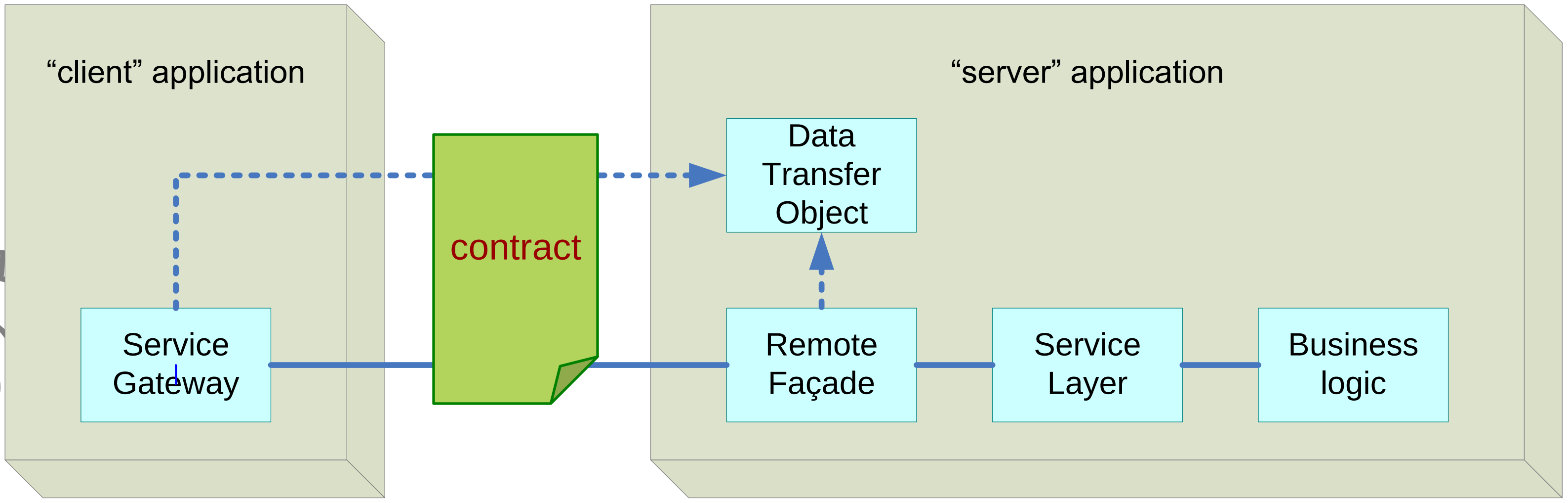
- A set of best-practices
- A typified solution for a common problem in a giving context
- Creates a common vocabulary
- **Patterns are discovered not invented**
- **“Patterns are half-baked”**
  - Martin Fowler

Anti-pattern

- Na example of what not to do
- Proven techniques that have shown bad results

PATTERNS FOR DISTRIBUTED APPLICATIONS

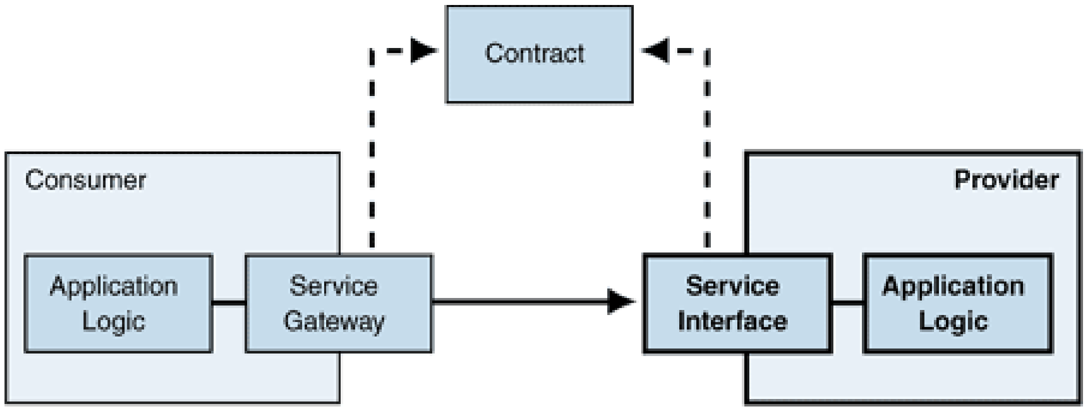
Architecture



Pattern

Service Gateway

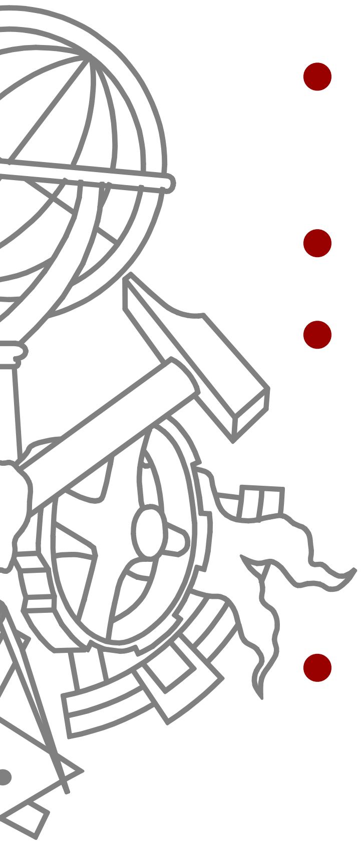
- An object that encapsulate the code that implements the consumer portion of a contract. They act as proxies to other services, encapsulating the details of connecting to the source and performing any necessary translation.



fonte: Enterprise Solution Patterns Using .NET

Pattern

Service Gateway



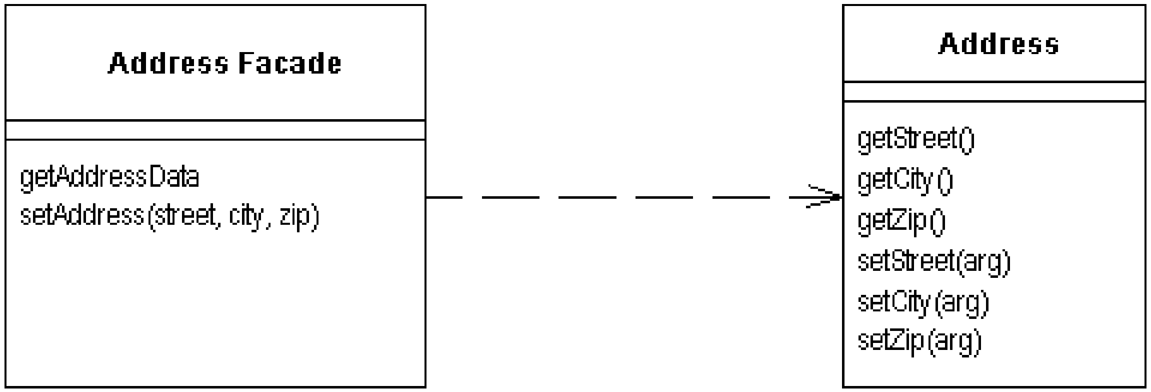
- Hides the details of accessing the service (ex., network protocol)
- May be considered a data access component
- Native support from most tools (e.g., Visual Studio, Netbeans, Rational software Architect) by web service proxies

- *See also Proxy and Broker pattern*

Pattern

Remote Façade

- Provides a coarse-grained façade on fine-grained objects to improve efficiency over a network

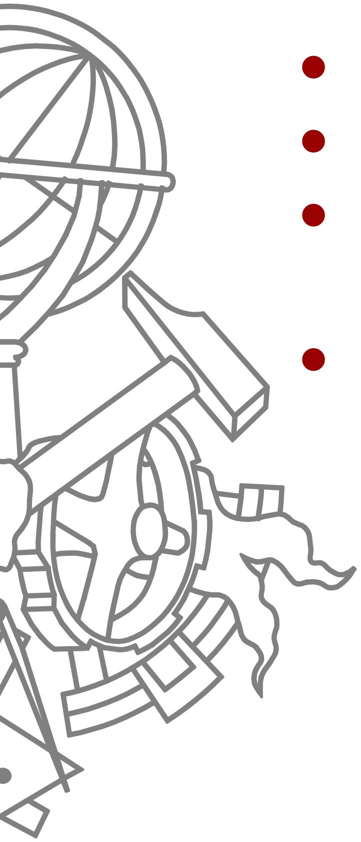


fonte: Patterns of Enterprise Application Architecture

Pattern

Remote Facade

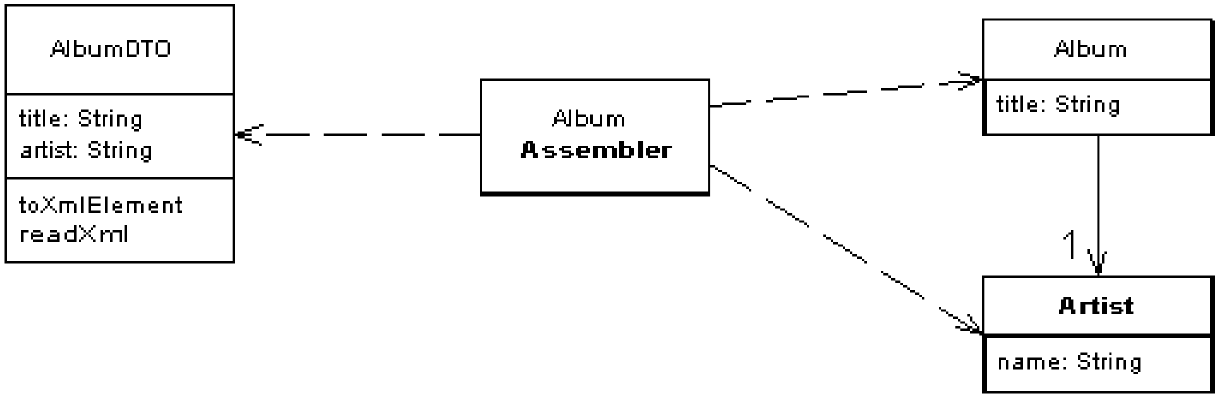
- Domain object interfaces are tipically fine grained
- Inadequeate for remote operations
- Create a surronding layer above domain objects
  - Local clients use the local interface
- The facade may encapsulate the interface of one or more business objects
  - Domain objects:
    - Address.New
    - Address.Set
    - Person.AddAddress
    - Person.Update
  - Remote Facade:
    - AddressFacade.AddNewAddressToPerson



Pattern

Data Transport Object

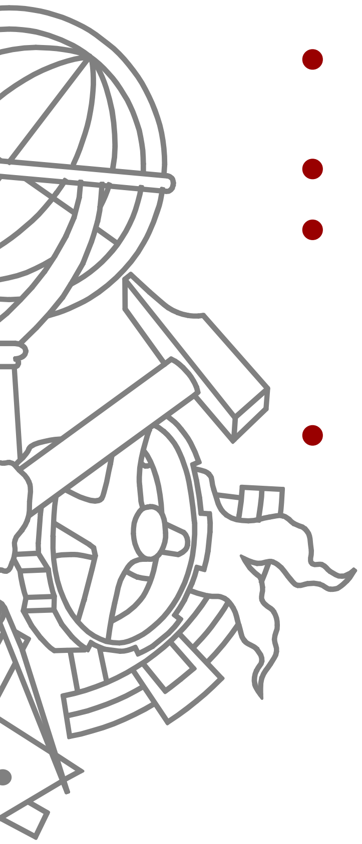
- An object that carries data between processes in order to reduce the number of method calls.



fonte: Patterns of Enterprise Application Architecture

Pattern

Data Transport Object

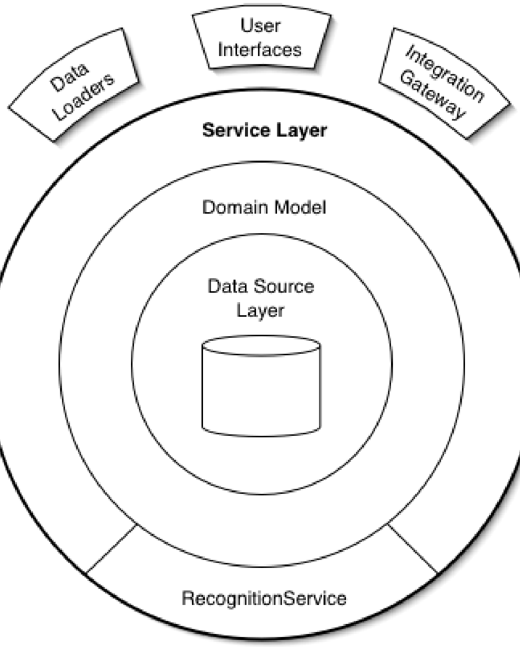


- Since XML is the *de facto* standard DTO should support serialization to/from XML
- Should be independent of the underlying domain object
- Should be implemented in accordance with the requiremnts of the remote application
  - CompleteCustomerInfoDTO
  - BasicCustomerInfoDTO
- Should be independent of the underlying platform (e.g., programming language)
  - DataSet/DataTable .net
  - ResultSet JDBC
  - DateTime .net

Pattern

Service Layer

- Defines an application's boundary with a layer of services that establishes a set of available operations and coordinates the application's response in each operation



fonte: Patterns of Enterprise Application Architecture

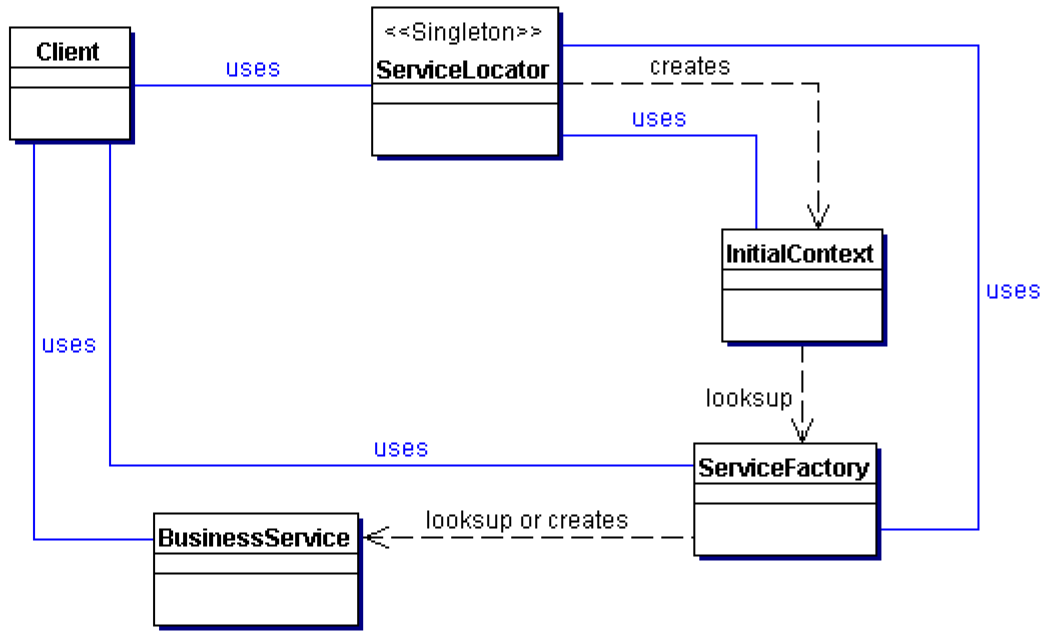
Pattern

Service Layer

- Domain logic pattern in the context of service orientation
- May be implemented as a Remote Facade or may be called by a Remote Facade

Service locator

- Hides the complexity of finding and creating service gateways



fonte: Core J2EE Patterns

Business Logic

- Outside of the scope
- Excellent reference: Patterns of Enterprise Application Architecture
  - Table Module
  - Table Data Gateway
  - Domain Model
  - Active Record
  - Data Mapper
  - Optimistic Offline Lock
  - …

SERVICE ORIENTATION

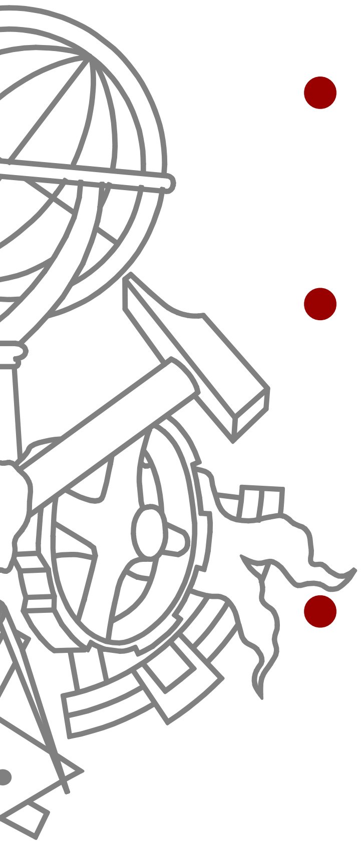
Definitions

- Contract
  - A funcionality provided by a party
- Service
  - An endpoint that fullfills one or more contracts
- Service Orientation
  - An architectural paradigm that employs the four tennets

The four tennets of SO

- Boundaries are explicit
- Share schema and contract not types
- Policy define service compatibility
- Services are autonomous

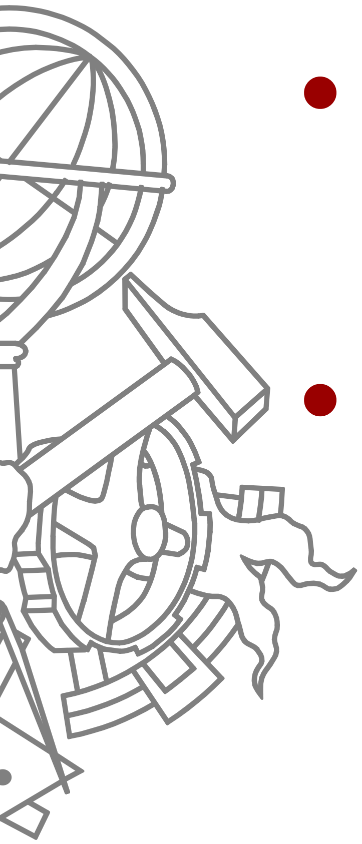
Boundaries are explicit



- Service boundaries are explicit and the cost of crossing a boundary is “known”
- A boundary is the border between the service public interface and its internal implementation
- Services interact intentionaly and explicitly by exchanging messages

Share schema and contract not types

- Services expose schemas defining data structures and contracts defining available operations
- Contracts and schema may be independently versioned over time



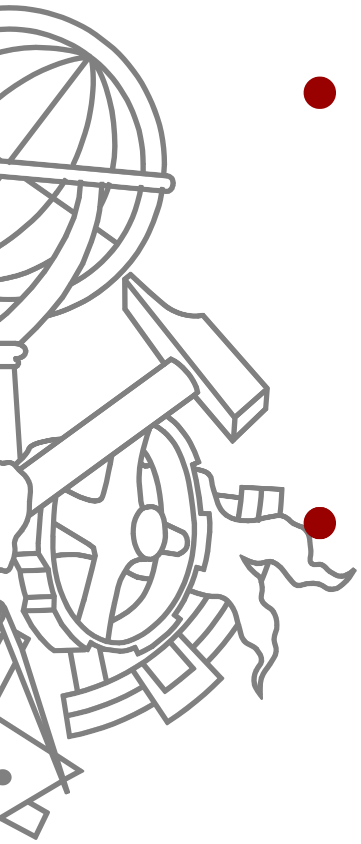
Policy define service compatibility

- Policy is the statement of communication requirements necessary for service interaction
- Service capabilities and requirements are expressed in terms of a policy expression
- A policy can contain multiple assertions

Services are autonomous

- Services are independently deployed, versioned and managed
- Autonomy ≠ Independence
- Topology of a system evolves over time
- Unlike OO, services do not share behavior
- Services gracefully handle failure

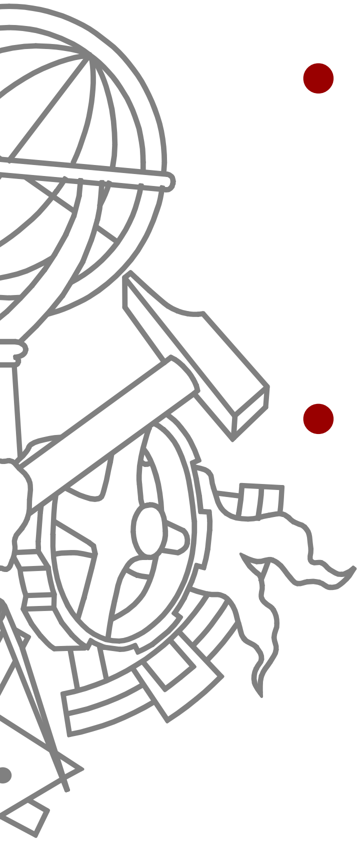
Service Anti-patterns



- CRUDy interface
  - Design the same old CRUD interface
  - verbose
- Loosey-Goosey
  - Design highly flexible interface
    - E.g., Expose direct SQL access
  - In the intent to provide flexibility, there is no service contract

Service Patterns

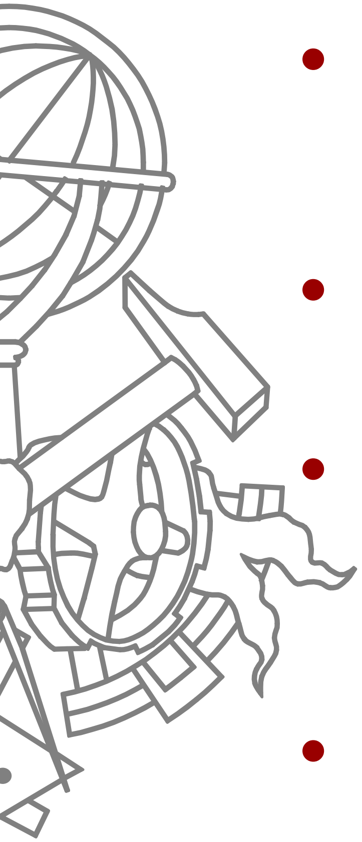
- Document Processor
  - Provide a document centric contract, not an RPC-like contract
- Reservation
  - Allow for long running transactions without locking
  - Must have compensation procedure



Exercise

- Remember the example DS you provided in the last session.
- Define an hypothetical SOA for that system
  - Define contract
  - Identify where you would use the presented patterns

Bibliography



- Buschmann, F.; Henney, K. And Schmidt, D. (2007) *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing, Volume 4.* Willey.
- *Patterns of Enterprise Application Architecture*. Martin Fowler. Adisson-Wesley.
- *Core J2EE Patterns: Best Practices and Design Strategies*. Deepak Alur, John Crupi and Dan Malks. Prentice Hall / Sun Microsystems Press. http://java.sun.com/blueprints/corej2eepatterns/index.html
- *Enterprise Solution Patterns Using Microsoft .NET*. Microsoft Press. http://msdn.microsoft.com/architecture/patterns/default.aspx?pull=/library/en-us/dnpatterns/html/Esp.asp

Suggested readings

- *Design patterns : elements of reusable object-oriented software*. Erich Gamma, Richard Helm, Ralph Johnson, John Vissides.
- *Pattern-oriented Software Architecture: System of Patterns*. Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal
- *Designing Data Tier Components and Passing Data Through Tiers*. Microsoft Patterns & Practices. http://msdn.microsoft.com/library/?url=/library/en-us/dnbda/html/BOAGag.asp?frame=true