AARHUS UNIVERSITET

Software Architecture in Practice

Role Based Design

Henrik Bærbak Christensen

AARHUS UNIVERSITET

Motivation

- The ‘decomposition’ step is probably the most devilish step in (software) engineering

  - Pattern: Divide it into ‘client’ and ‘server’ Fine!
    - Next: How to divide ‘server’ into, into, into… what? Tricky!
- Early object-orientation
  - ‘Model domain into objects, add methods’
  - Lead often to ‘big ball of mud’ – *the blob* or the *god class*
- *I found the **responsibility-centric** paradigm a big eye opener*
  - Smaller, well-focused objects
  - Basically a *grouping functionality oriented way of decomposition!*

What do people say about objects?

AARHUS UNIVERSITET

(l) A class definition encapsulates its objects' data and actions. [Morelli, 2000, p. 61]

(l) An object is a program construction that has data (that is, information) associated with it and that can perform certain actions. [Savitch, 2001, p. 17]

(l) A class definition describes the behavior and attributes of typical instances of that class. [Barnes, 2000, p. 36]

(l) An object is defined by a class, which can be thought of as the data type of the object. [Lewis and Loftus, 2003, p. 62]

(l) An object is characterized by its state, behavior, and identity. [Hortsmann, 2004, p. 39]

(m) Model elements in Java programs are called objects. [Arnow and Weiss, 2000, p. 4]

(m) Java objects model objects from a problem domain. [Barnes and Kolling, 2005, p. 3]

(r) The best way to think about what an object is, is to think of it as something with responsibilities. [Shalloway and Trott, 2004, p. 16]

(r) An object-oriented program is structured as a community of interacting agents called objects. Each object has a role to play. Each object provides a service or performs an action that is used by other members of the community. [Budd, 2002, p. 9]

AARHUS UNIVERSITET

Perspectives

- ***Language centric* perspective:**
  - Object = Data + Actions

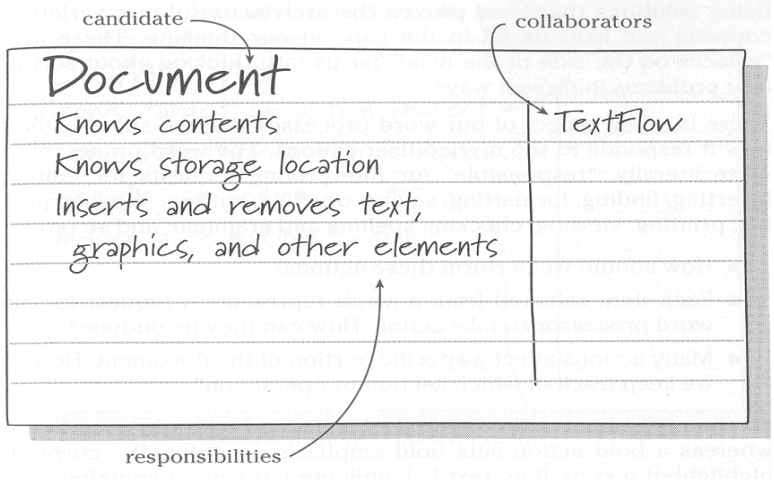
- ***Model centric* perspective:**
  - Object = Model element in domain

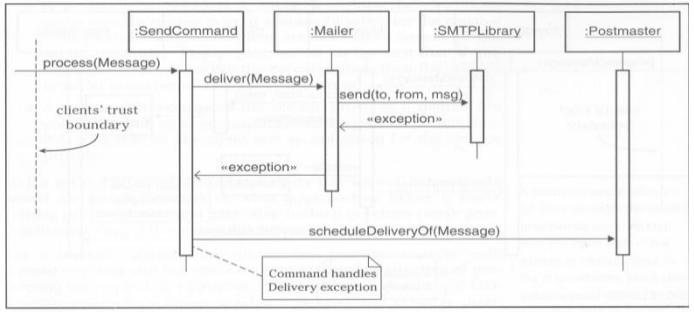
- ***Responsibility centric* perspective:**
  - Object = Responsible for providing service in community of interacting objects

Responsibility-centric

Focus

- Responsibility centric focus
  - Role, responsibility, and collaboration
  - Object = provider of service in community
  - Leads to strong *behavioral* focus
  - CRC cards (Beck, Wirfs-Brock)





AARHUS UNIVERSITET

Another Definition

- Another definition:
  - *An object-oriented program is structured as a **community** of **interacting agents** called objects. Each object has a **role** to play. Each object **provides a service** or performs an action that is used by other members of the community.*
  - *Budd 2002*

- Shifting focus
  - away from “model of real world”
  - towards “community”, “interaction”, and “service”

Service

- Budd's definition is more skewed towards the functionality of the system.
  - **At the end of the day, software pays the bill by providing *functionality* that the users need, not by being a nice model of the world!**
- Services are what developers get paid to create!

Service-oriented?
MicroServices?

What/Who

- Timothy Budd:
  - *"Why begin the design process with an analysis of behaviour? The simple answer is that the behaviour of a system is usually understood long before any other aspects."*

- *What / Who cycle*
  - **What:** identify behaviour / responsibility ⇨ roles
  - **Who:** identify objects that may play the roles
    - or even invent objects to serve roles only
      - Larman "Pure fabrication";

Implications

- Responsibility perspective:
  - A) Analyze behavior (what?)
  - B) Assign objects (who?)
- Guidelines:
  - A) Behavior abstracted ⇨ landscape of *responsibilities*
  - B) Implement responsibilities in objects
- Analysis
  - Resemble human organizations – often roles are invented
  - Still need to define the objects ☺
    - That is, the person(s) to fill the role

AARHUS UNIVERSITET

The Central Concepts

A strong mind-set for
designing flexible software

“Theory of Flexible Designs”

AARHUS UNIVERSITET

How people organize work!

- The central concepts:
  - **Behaviour:** *What actually is being done*
    - ”Henrik sits Sunday morning and writes these slides”
  - **Responsibility:** *Being accountable for answering request*
    - ”Henrik is responsible for teaching responsibility-centric design”
  - **Role:** *A function/part performed in particular process*
    - ”Henrik is the course teacher”
  - **Protocol:** *Convention detailing the expected sequence of interactions by a set of roles*
    - ”Teacher: ‘Welcome’ => Students: stops talking and starts listening”

AARHUS UNIVERSITET

It is all Roles and Protocol

- Any complex human organization relies completely on each person understanding roles and protocols
  - If I get hospitalized, I understand the roles of patient, nurses, and physicians
  - CEOs, managers, software developers, architects, testers, sales people, …
  - Hardship of marriage: finding the proper roles and protocols ☺
  - Cultural clashes: Hindu programmer and lunch story

AARHUS UNIVERSITET

Roles decouples

- The primary point of roles:
  - ***It provides a higher abstraction than that of the individual person***

- I know my responsibilities and the protocol once I am assigned a known role

- I can collaborate efficiently with others once I know their roles

AARHUS UNIVERSITET

Many-to-many relation

- Big company
  - One person is manager, one software architect, two lead developers, and ten software developers
- Small company
  - Same person is manager, software architect, lead and software programmer ☺
- That is: **One individual may server many roles**
- *Henrik: Teacher, researcher, tax payer, company owner, tourist, father, husband, …*

AARHUS UNIVERSITET

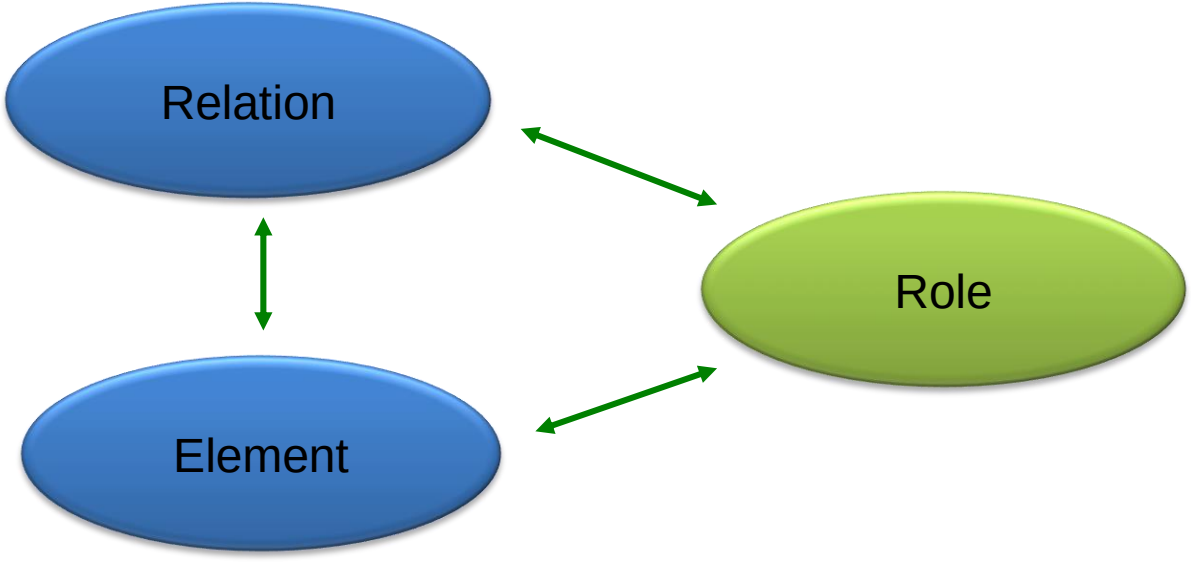
Many-to-many relation

- Hospital
  - Nurses attend the patients
  - And different persons serve the role during shifts

- That is: **One role may be served by many persons**

Role concept

- The role concept allows us to use *either* approach (who/what or what/who) because “what” can be expressed as roles.



Role makes service a *first-class citizen* of our design vocabulary

AARHUS UNIVERSITET

Roles are invented

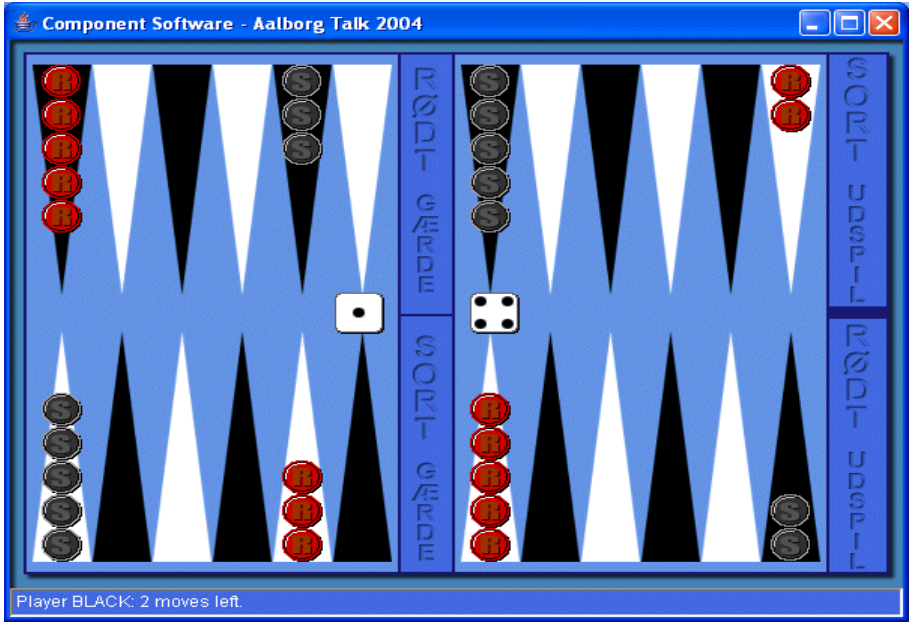
- Roles are invented by need.

- A pre-school kindergarten invented a ***Flyer*** role whose responsibility it was to ‘catch’ all interruptions to make the daily work more fluent for the ‘non-flyer’ pedagogues.

Example

A Case Study

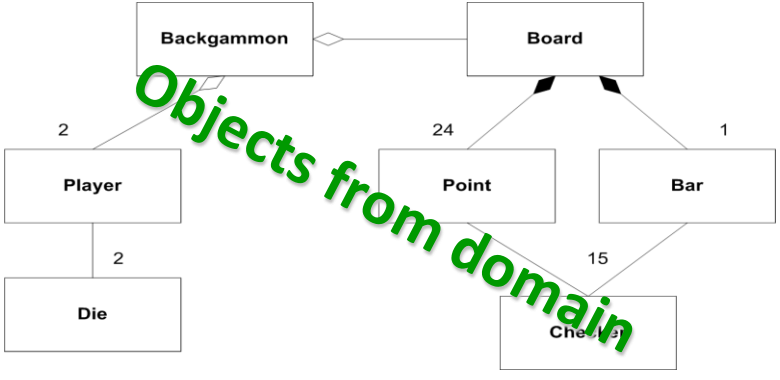
- Backgammon requirements:
  - Offer GUI for two players
  - Guaranty proper play
- Variants
  - *new rules* for which moves are legal
  - how many moves you can make per turn
  - how the board is initially set up



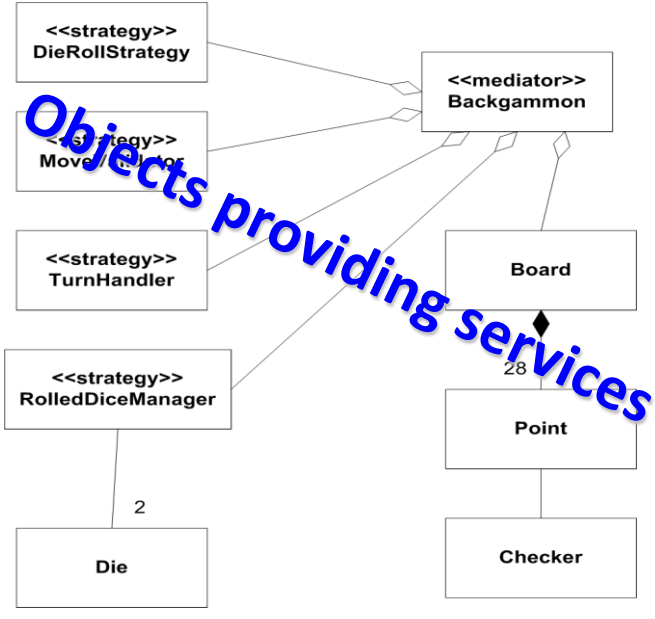
Same challenge – different designs

AARHUS UNIVERSITET

Model perspective:

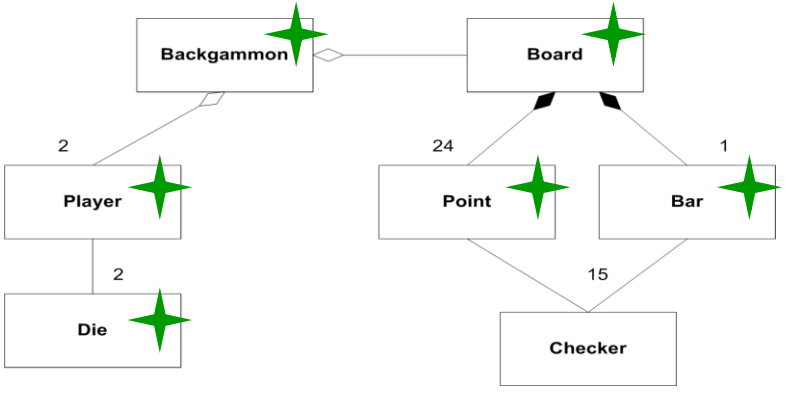


Responsibility perspective:

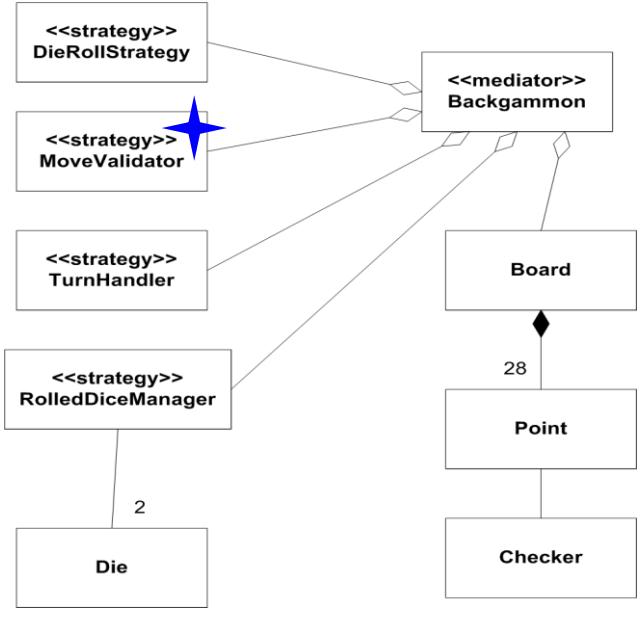


Who is responsible for validating moves?

Model perspective:



Responsibility perspective:



What is the cost of altering validation strategy?
How to change it at run-time?

AARHUS UNIVERSITET

And – architecture?

Architectural roles…

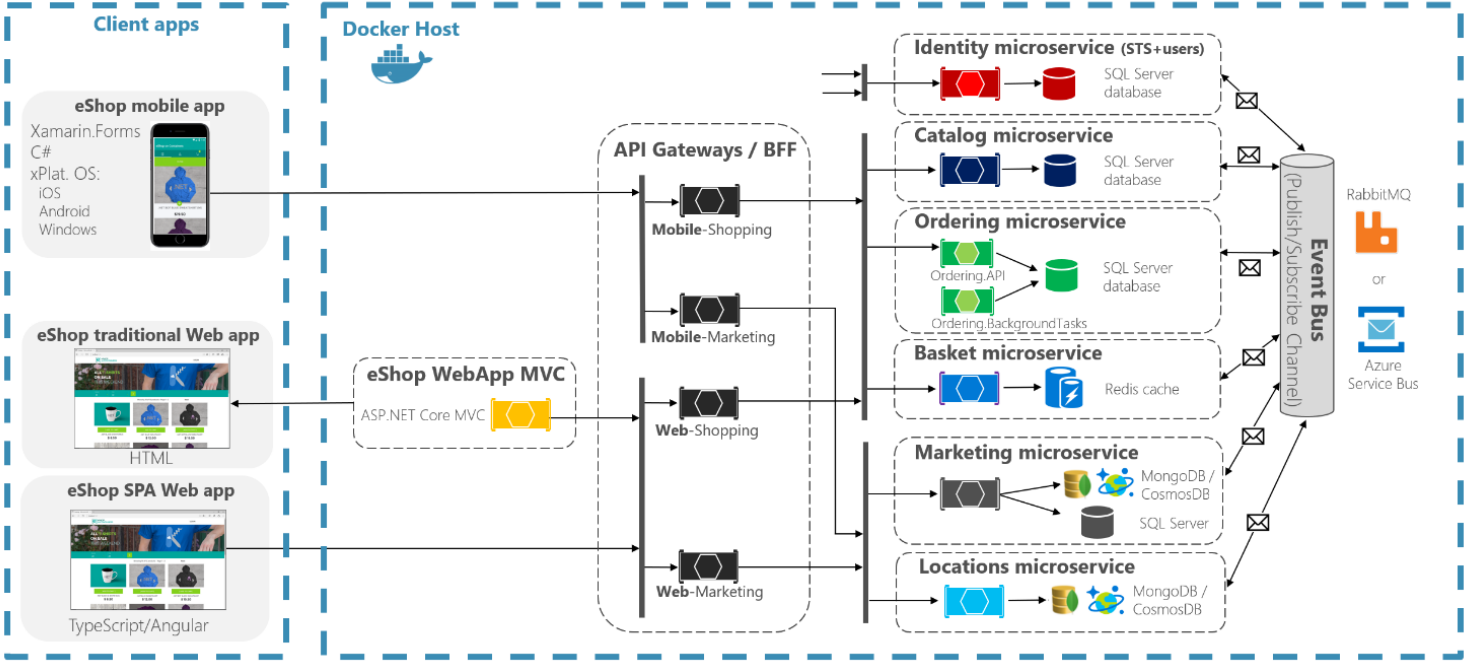
AARHUS UNIVERSITET

Architectural Roles

- Often architectural requirements can require us to create *architectural roles*
  - The Cache role
    - Responsibility: Store (key,value) pairs across horizontal scaled servers, with ability to set expiration time for each
  - The Storage role
    - Resp: Provide system wide (or ‘microservice wide’) persistence of domain objects
  - The Subscription role
    - Resp: To know all users, to grant authority to access resources, to …
  - The Catalog role
    - Resp: To know all data about purchasable item: name, image, price,

Example

- MicroSoft: *'eShopOnContainers'* on github
  - Source: https://aka.ms/microservicesebook



Architectural Constraints on Roles

AARHUS UNIVERSITET

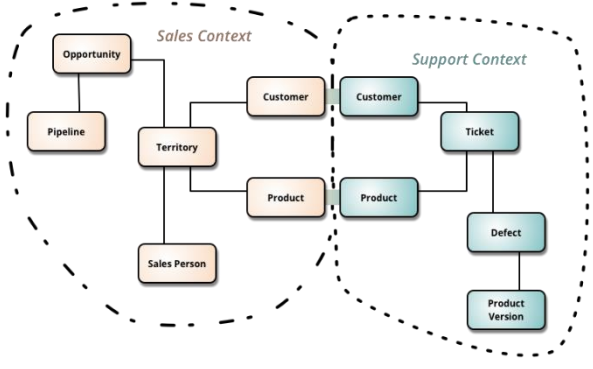
- Often constraints enforce specific requirements on roles
  - Statelessness (Increase scaling, decrease performance)
    - Objects implementing this role **may never** cache state locally
      - Must always fetch it from Storage or Cache
  - Statefullness
    - Well – the opposite ☺
  - Performance – ‘avoid chatty interface’
    - Client-objects cache data locally, bulk-tranfers all state periodically

AARHUS UNIVERSITET

Bounded Contexts

- Eric Evans : Domain-Driven Design
  - Hm, has this strong OOA and OOD flavor which I found lead to blob designs…
- Bounded Contexts
  - Instead of *one giant domain model* you cut that into subdomains. Each subdomain share concepts *but implement/view them seperately*

DDD calls the scope of a domain model a *bounded context.*



Summary

Keypoints

- Make them highly cohesive
  - **One** abstraction of **behavior** or **domain concept**
  - Few high-cohesive methods expressing a few responsibilities for a naturally-nameable role
    - Not ‘HardwareInteractionAnd10MinuteMeanValueComputingRole’
- Make them small
  - Avoid ‘god class/the blob’, split’em before it is too late
- Split along architectural boundaries
  - Statefull (db/cache) and stateless (service)
- And probably much more…

AARHUS UNIVERSITET

Ex: Uber Architecture

- Continuous Deployment puts architectural constraints to the ‘roles’ that services play. They divide them into
  - Stateless services: application servers, can be reintroduced
  - Statefull services: ‘databases with REST interface’
  - Batch services: long running analysis tasks, datamining