Domain Driven Design (DDD)

About me

Trayan Iliev

– CEO of **IPT – Intellectual Products & Technologies**
http://www.iproduct.org

– Oracle® certified programmer 15+ Y

– end-to-end reactive fullstack apps with Java, ES6+, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer: Spring, Java EE, Node.js, Express, GraphQL, SOA, REST, DDD & Reactive Microservices

– Voxxed Days, jPrime, Java2Days, jProfessionals, BGOUG, BGJUG, DEV.BG speaker

– Organizer RoboLearn hackathons and IoT enthusiast

Where to Find The Code and Materials?

https://github.com/iproduct/react-typescript-academy-2022

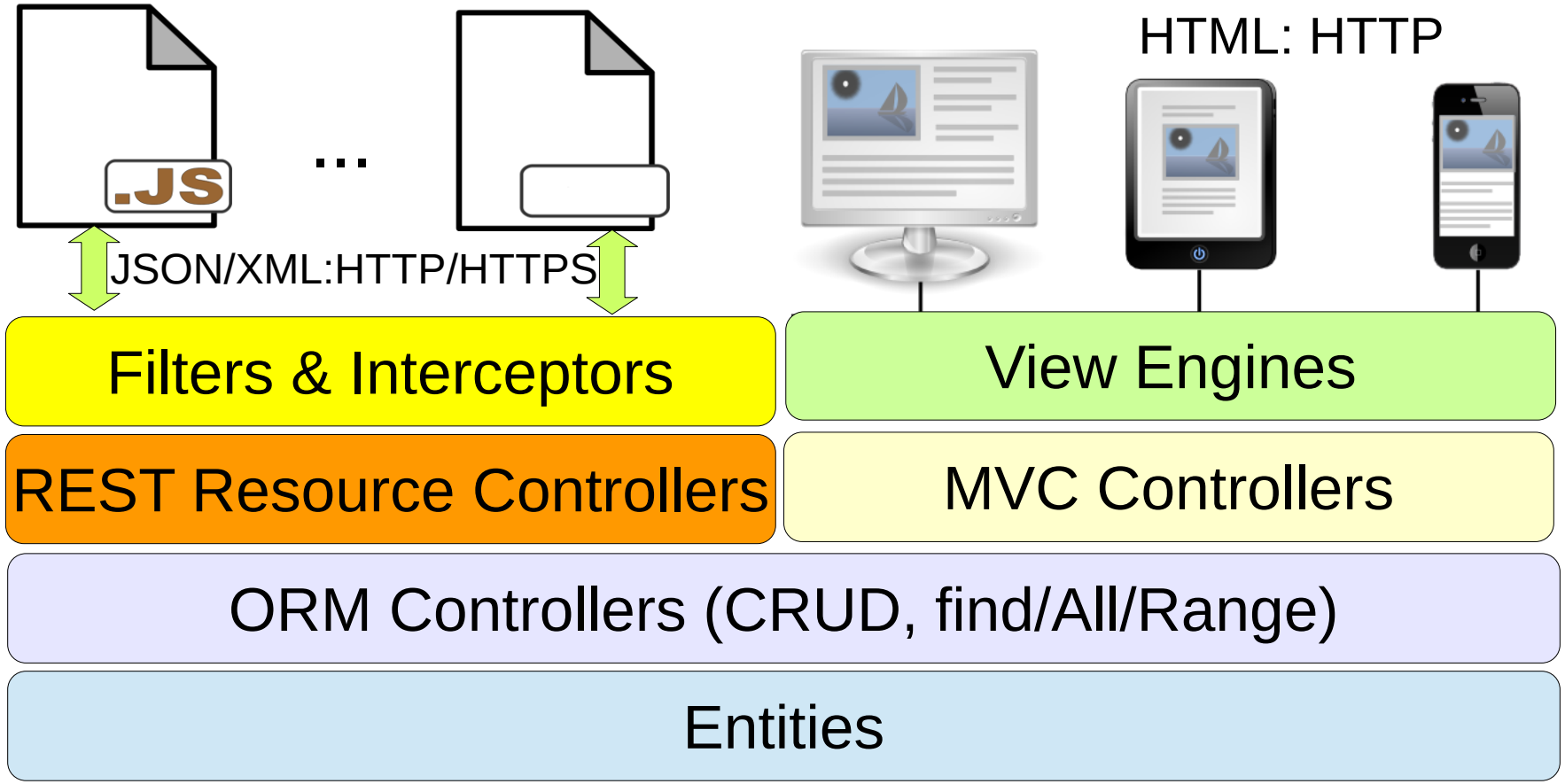
SOLID Design Principles of OOP

- Single responsibility principle - a class should only have a single responsibility, that is, only changes to one part of the software's specification should be able to affect the specification of the class.
- Open–closed principle - software entities should be open for extension, but closed for modification.
- Liskov substitution principle - Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.
- Interface segregation principle - Many client-specific interfaces are better than one general-purpose interface.
- Dependency inversion principle - depend upon abstractions, not concretions.

Types of Web Applications

- Web Sites – presenting interactive UI
  - Static Web sites – show the same information for all visitors – can include hypertext, images, videos, navigation menus, etc.
  - Dynamic Web sites – change and tune the content according to the specific visitor
    - server-site – use server technologies for dynamic web content (page) generation (data comes drom DB)
    - client-side – use JavaScript and asynchronous data updates
- Web Services – managing (CRUD) data resources
  - Classical – SOAP + WSDL
  - RESTful – distributed hypermedia

N-Tier Web Architectures



Domain Driven Design (DDD)

We need tools to cope with all that complexity inherent in robotics and IoT domains.

Simple solutions are needed – cope with problems through divide and concur on different levels of abstraction:

Domain Driven Design (DDD) – back to basics:
domain objects, data and logic.

Described by Eric Evans in his book:
Domain Driven Design: Tackling Complexity in the Heart of Software, 2004

7

Domain Driven Design (DDD)

Main concepts:

- ❖ Entities, value objects and modules
- ❖ Aggregates and Aggregate Roots [Haywood]:

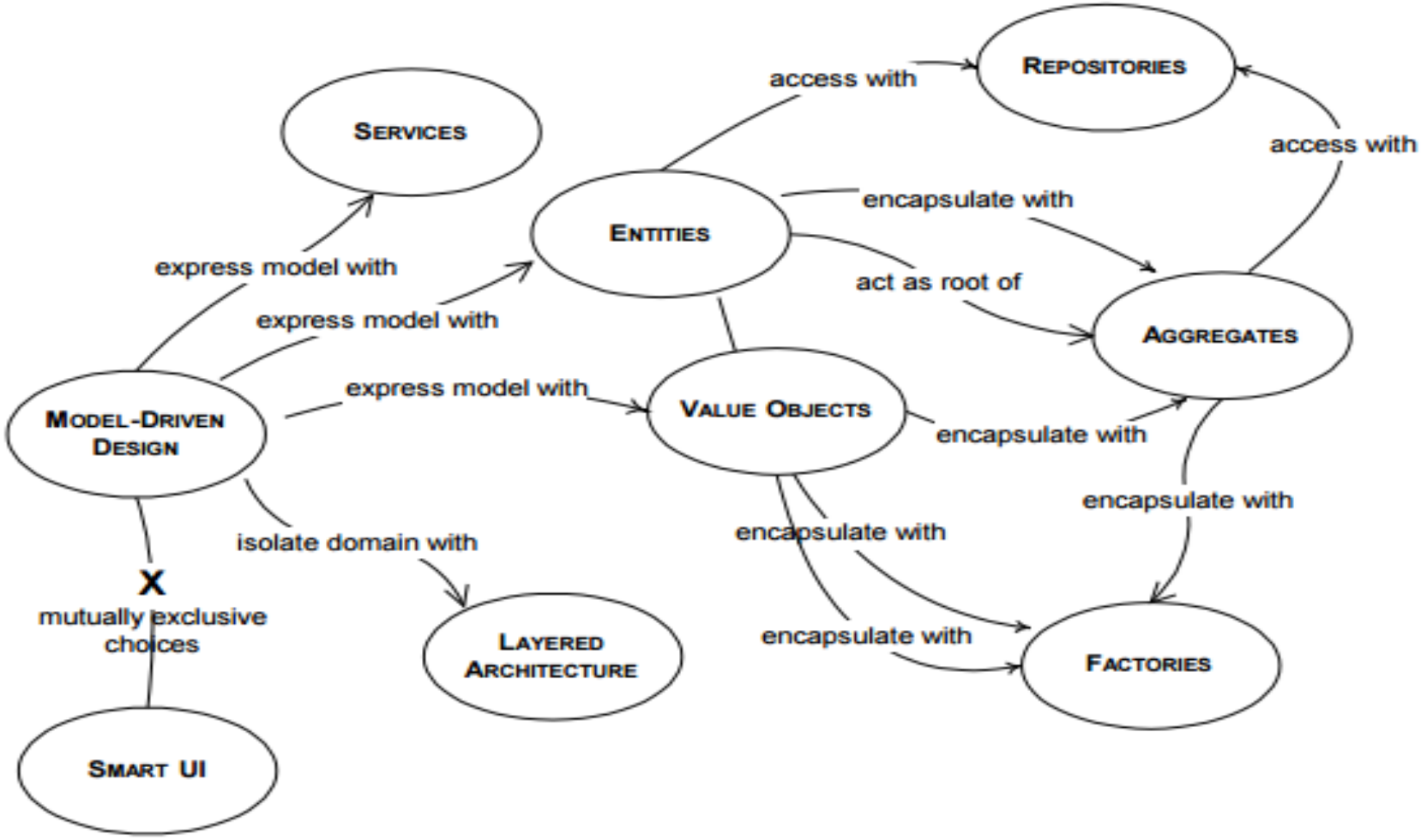
value < entity < aggregate < module < BC

- ❖ Repositories, Factories and Services:

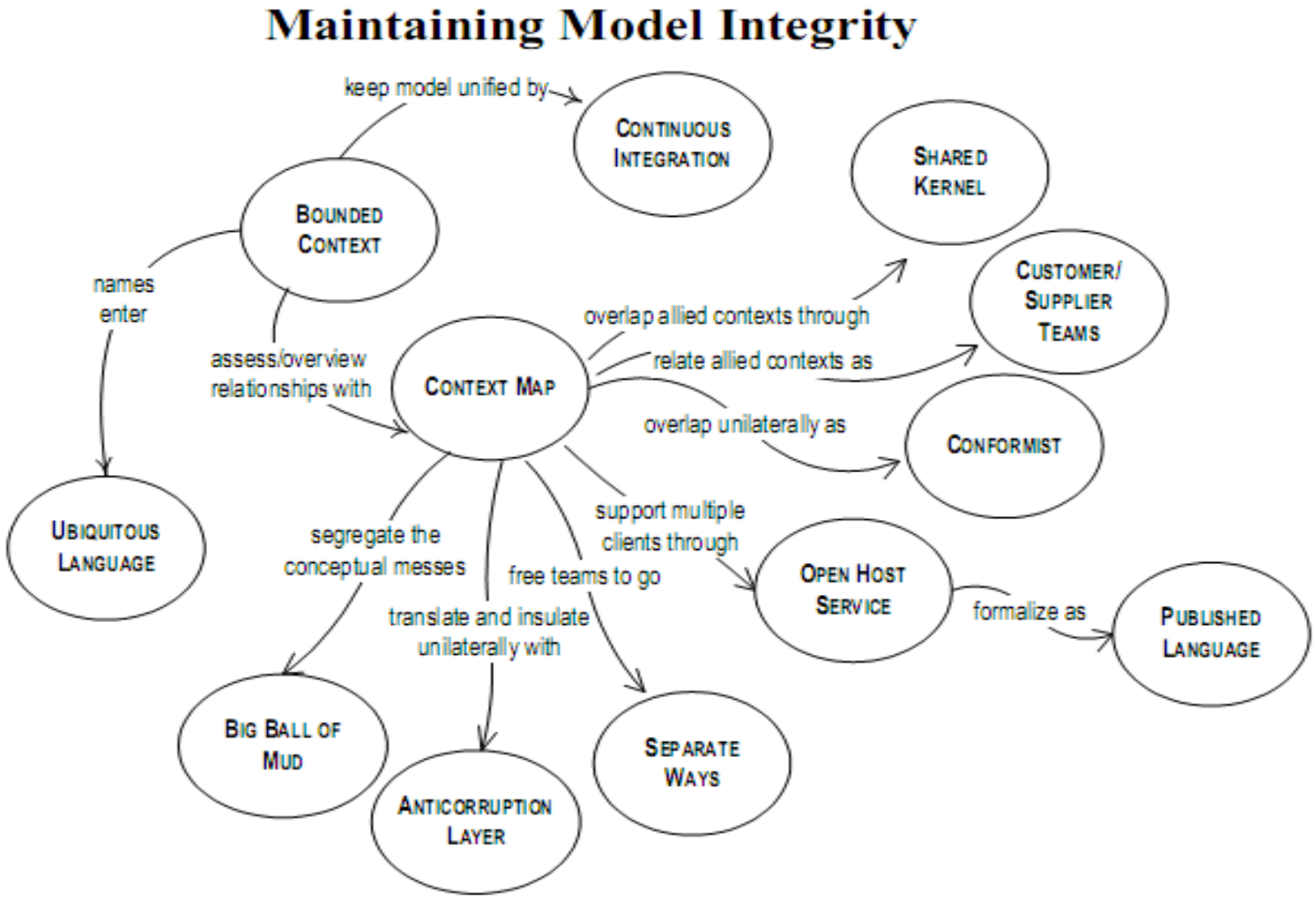
application services <-> **domain services**

- ❖ Separating interface from implementation

Domain Driven Design (DDD)



Domain Driven Design (DDD)



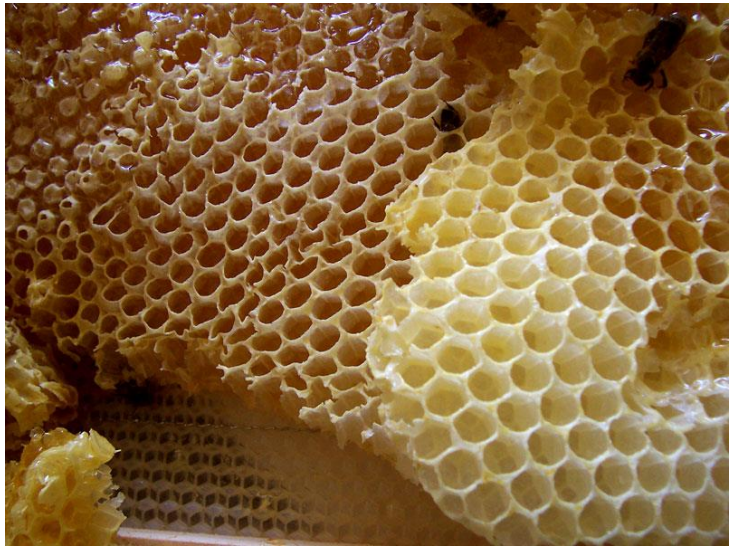
Domain Driven Design (DDD)

- Ubiquitous language and Bounded Contexts
- DDD Application Layers:
- Infrastructure, Domain, Application, Presentation
- Hexagonal architecture :

OUTSIDE <-> transformer <->

(application <-> domain)

[A. Cockburn]



11

Hexagonal Architecture Principles

- Allows an application to equally be driven by users, programs, automated test or batch scripts, and to be developed and tested in isolation from its eventual run-time devices and databases.
- As events arrive from the outside world at a port, a technology-specific adapter converts it into a procedure call or message and passes it to the application
- Application sends messages through ports to adapters, which signal data to the receiver (human or automated)
- The application has a semantically sound interaction with all the adapters, without actually knowing the nature of the things on the other side of the adapters

Source: http://alistair.cockburn.us/Hexagonal+architecture

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD