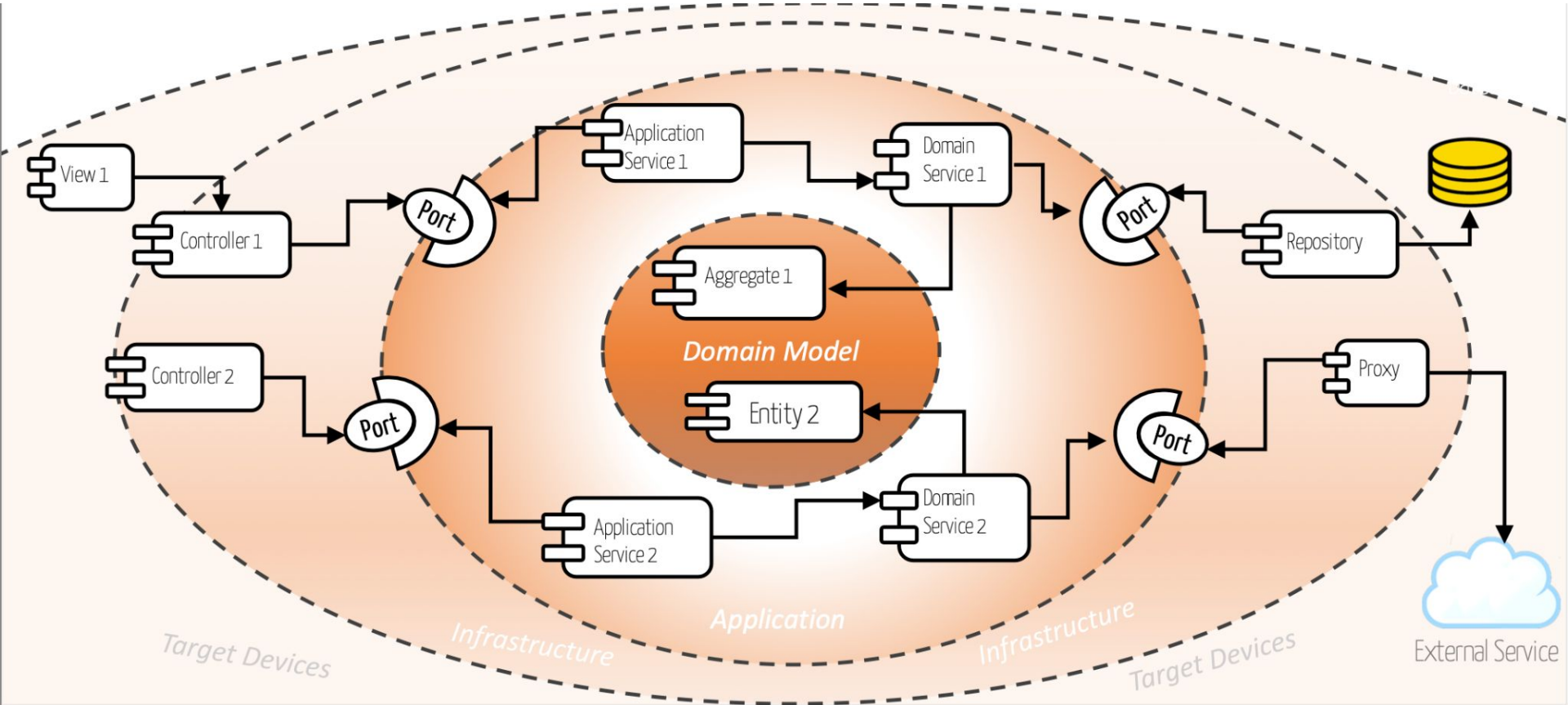
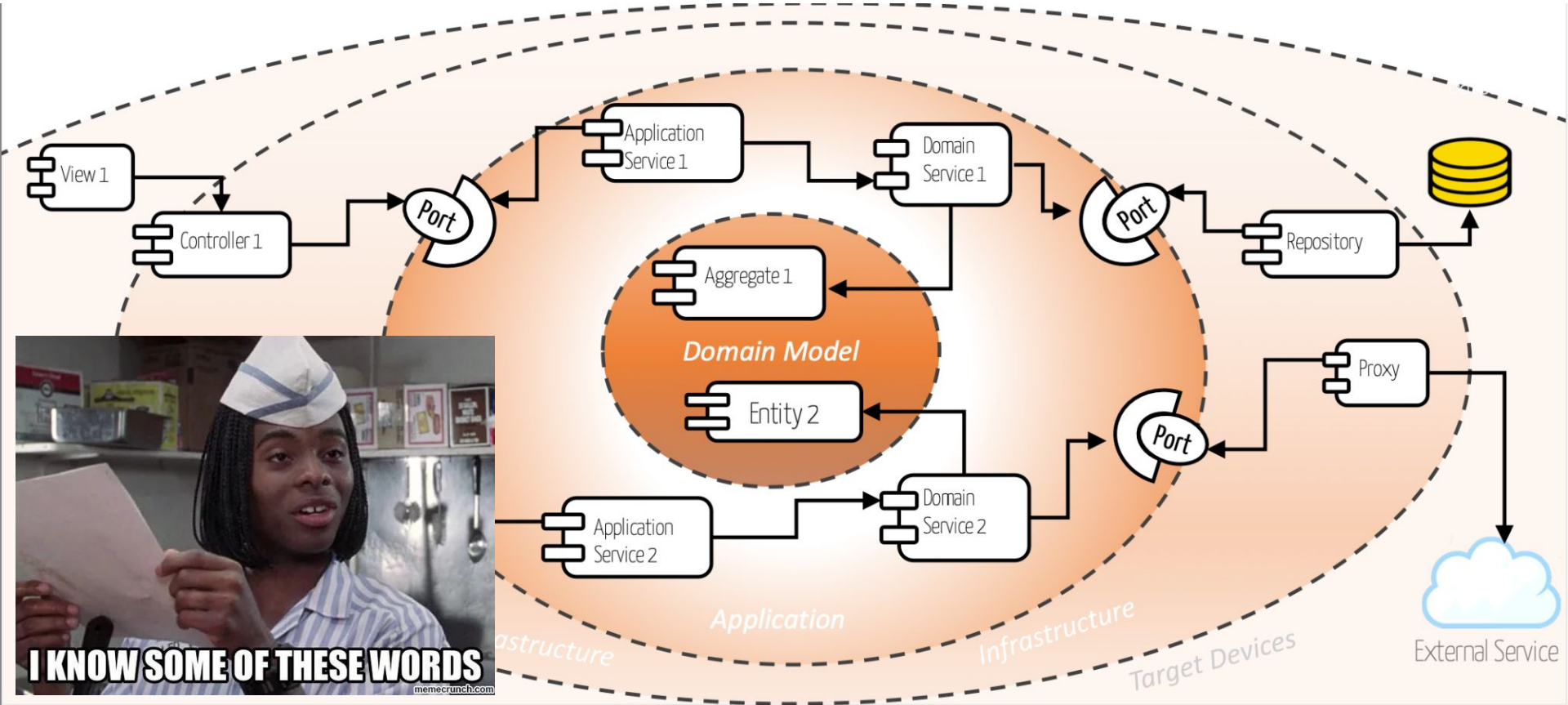
Tactical Patterns of Domain Driven Design

High-level Intro by Maria Schönholzer

Motivation


View 1
Controller 1
Port
Application Service 1
Domain Service 1
Port
Repository
Aggregate 1
Domain Model
Entity 2
Controller 2
Port
Application Service 2
Domain Service 2
Port
Proxy
Application
Infrastructure
Target Devices
Target Devices
Infrastructure
External Service

Motivation


View 1
Controller 1
Port
Application Service 1
Domain Service 1
Port
Repository
Aggregate 1
Domain Model
Entity 2
Proxy
Port
Application Service 2
Domain Service 2
Application
Infrastructure
Target Devices
External Service
I KNOW SOME OF THESE WORDS
memecrunch.com

Domain Driven Design (DDD)

Eric Evans' *Domain-Driven Design: Tackling Complexity in the Heart of Software* (2003)

- Approach for dealing with highly complex domains
  - Domain – the main focus of the project
  - Software model reflects a deep understanding of the domain
- Collaboration with domain expert
- Shared understanding of the business context

Two phases of DDD

- Strategic
  - Large-scale structure of the system
  - Architecture remains focused on business capabilities

Two phases of DDD

- Strategic
  - Large-scale structure of the system
  - Architecture remains focused on business capabilities
- Tactical
  - More precise domain structure
  - Set of design patterns to create loosely coupled and cohesive model

Tactical DDD Patterns

- Entity
- Value Object
- Domain Event
- Service
- Module
- Aggregate
- Repository
- Factory

Entities

- Unique identifier
- Attributes may change over time
- Can hold references to other entities

Examples (in a banking application): Customers, Accounts and Transactions

Value Objects

- No identity
- Defined by attribute values
- Immutable

Examples: dates & times, currency values

Aggregates

- Consistency boundary around one or more entities
- Only one root entity
- Only root can be referenced from outside
- Main purpose: model transactional invariants

Examples: Customers create Orders, Orders contain Products, Products have Suppliers etc.

Domain events

- Notify other parts of the system when something happens
- Should mean something within the domain

Example: a delivery was cancelled

Anti-example: a record was inserted into a table

Domain and Application Services

- Domain services: encapsulate domain logic
  - Model behavior that spans multiple entities
- Application services: provide technical functionality
  - E.g. user authentication or sending an SMS message

Conclusion

Values of Agile Manifesto (Extended)

Individuals and interactions		Processes and tools
Working software		Comprehensive documentation
Customer collaboration	OVER	Contract negotiation
Responding to change		Following a plan
Collaboration with domain experts		*Implementing a handful of patterns*
(Strategic DDD)		*(Tactical DDD)*

Any Questions?

Reference

- Using tactical DDD to design microservices
  - https://learn.microsoft.com/en-us/azure/architecture/microservices/model/tactical-ddd
- Domain-Driven Design Reference: Definitions and Pattern Summaries by Eric Evans (2015)
  - https://www.domainlanguage.com/wp-content/uploads/2016/05/DDD_Reference_2015-03.pdf
- Agile Technical Practices Distilled by Pedro Moreira Santos, Marco Consolaro and Alessandro Di Gioia (2019)
  - Domain-Driven Design: What domain?