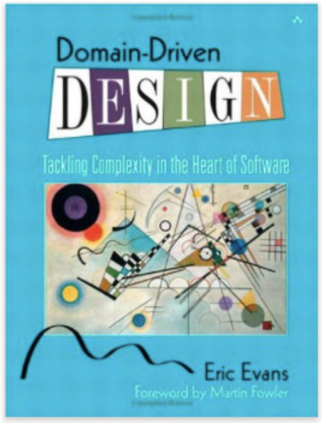
EDOM - Engenharia de Domínio
Mestrado em Engenharia Informática
Lecture 02.1
Domain Modeling Fundamentals

Alexandre Bragança atb@isep.ipp.pt

Dep. de Engenharia Informática – ISEP

2017/2018

Acknowledgement


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

"Domain-Driven Design: Tackling Complexity in the Heart of Software", Eric Evans, Addison Wesley, 2003

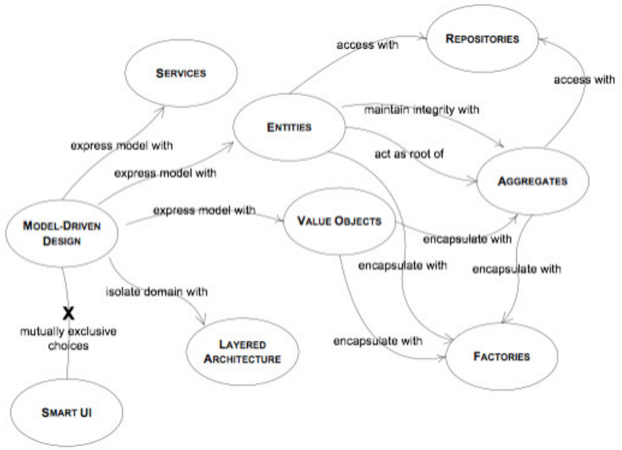
This lecture is based on the contents of this book.

There is a reference document explaining the principles of the book that you can find at: http://domainlanguage.com/wp-content/uploads/2016/05/PatternSummariesUnderCreativeCommons.doc)

There is a reference document containing some updates that you can find at: https://domainlanguage.com/ddd/reference/DDD_Reference_2015-03.pdf)

We will be focusing on the section "Building Blocks of a Model-Driven Design".

The Building Blocks of Model-Driven Design



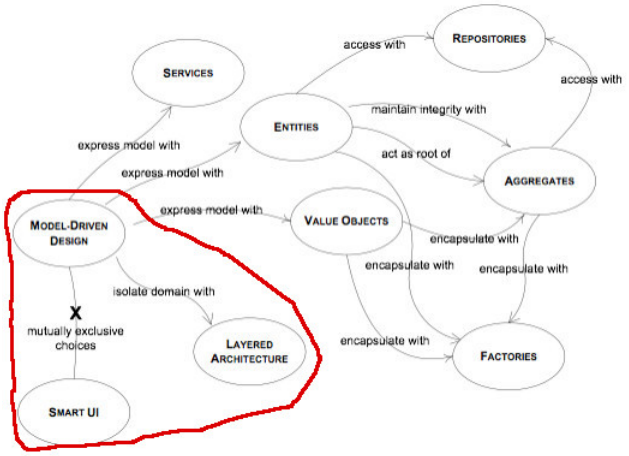
Our focus will be on conceptual guidelines that we can use to construct models. Therefore we will be working with **metamodels**. This is different from the focus of the book. However, the basic principles apply and are very important.

Metamodeling

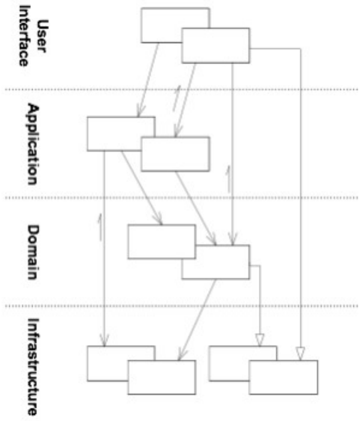
- **Metamodel** or surrogate model is a model of a model, and **metamodeling** is the process of generating such metamodels. Metamodeling or meta-modeling is the **analysis, construction and development of the frames, rules, constraints, models and theories** applicable and useful for modeling a predefined **class of problems**. As its name implies, this concept applies the notions of meta- and modeling in software engineering and systems engineering. Metamodels are of many types and have diverse applications.[1]
- In the course we will use the **Eclipse Modeling Framework (EMF)** as our tool for metamodeling (`https://eclipse.org/modeling/emf/`). EMF is included in Eclipse when we download the package **Eclipse Modeling Tools** or the package **Eclipse IDE for Java and DSL Developers**.

[1] From `https://en.wikipedia.org/wiki/Metamodeling`

Isolating the Domain



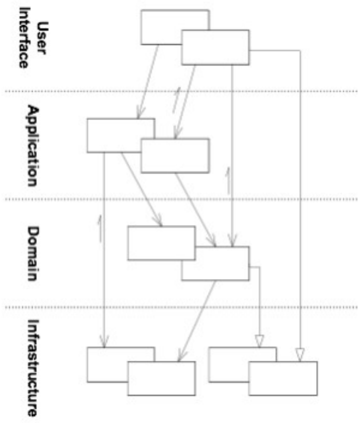
Layered Architecture



- Partition a complex program into layers.
- Develop a design within each layer that is cohesive and that depends only on the layers below.
- Follow standard architectural patterns to provide loose coupling to the layers above[2].
- **Concentrate all the code related to the domain model in one layer and isolate it from the user interface, application, and insfrastructure code**.

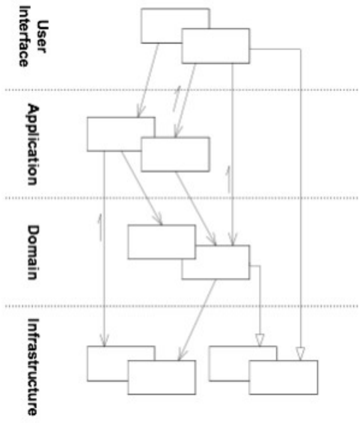
[2]Do you remember the MVC pattern from EAPLI? Or the Boundary-Control-Entity?

User Interface or Presentation Layer



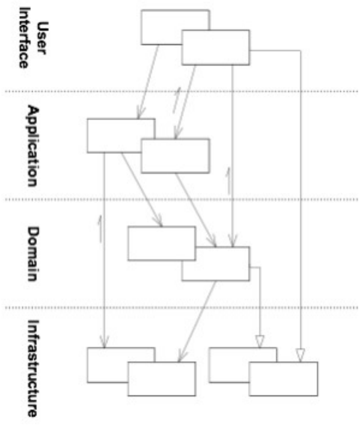
- Responsible for showing information to the user and interpreting the user's commands.
- The external actor might sometimes be **another computer system** rather than a human user.

Application Layer



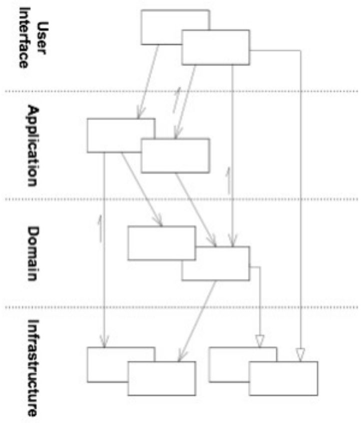
- Defines the jobs that the software is supposed to do and directs the expressive domain objects to work out problems.
- The tasks this layer is responsible for are meaningful to the business or necessary for interaction with the application layers of other systems.
- **This layer is kept thin**.
- It does not contain business rules or knowledge, but only coordinates tasks and delegates work to collaborations of domain objects in the next layer down.
- It **does not have state reflecting the business situation**, but it can have state that reflects the progress of a task for the user or the program.

Domain Layer (or Model Layer)



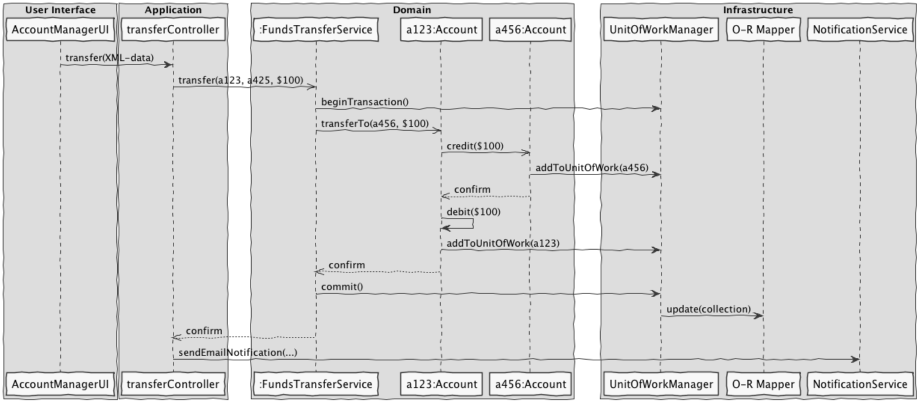
- Responsible for **representing concepts of the business, information about the business situation, and business rules**.
- State that reflects the business situation is controlled and used here, even though the technical details of storing it are delegated to the infrastructure.
- *This layer is the heart of business software.*

Infrastructure Layer



- Provides generic technical capabilities that support the higher layers: message sending for the application, persistence for the domain, drawing widgets for the UI, and so on.
- The infrastructure layer may also support the pattern of interactions between the four layers through an architectural framework.

Example



User Interface
Application
Domain
Infrastructure
AccountManagerUI
transferController
:FundsTransferService
a123:Account
a456:Account
UnitOfWorkManager
O–R Mapper
NotificationService
transfer(XML–data)
transfer(a123, a425, $100)
beginTransaction()
transferTo(a456, $100)
credit($100)
addToUnitOfWork(a456)
confirm
debit($100)
addToUnitOfWork(a123)
confirm
commit()
update(collection)
confirm
sendEmailNotification(...)

Domain Model vs Domain-Specific Language

- Usually a DSL is a language that expresses part of a Domain Model.
- They can also be used to express technical parts of the solution (i.e., user interface, database, etc).
- DSL Tools or Metamodeling Tools are based on three aspects[3]:
    - **Semantic Model schema** defines the data structure of the Semantic Model, together with static semantics, usually by using a meta-model;
    - **DSL editing environment** defines a rich editing experience for people writing DSL scripts, through either source editing or projectional editing;
    - **Semantic Model behavior** defines what the DSL script does by building off the Semantic Model, most commonly with code generation.

[3] "Domain-Specific Languages", Martin Fowler, Addisson-Wesley Professional, 2010.

Hands-on Eclipse Modeling Framework

- Lets see a simple example of how to use EMF for creating a domain model...
- We will start Eclipse and select "File->"New->"Ecore Modeling Project"...
- More details during a future lab class...