ulm university universität ulm

Evaluating Domain-Driven Design for Refactoring Existing Information Systems

Final Master Thesis Presentation at Ulm University

Institute of Database and Information Systems

Hayato Hess | May 18, 2016

Agenda

Introduction

Challenges

Domain-Driven Design

Refactoring

Prototype

Introduction

Herbert Hayato Hess
- Computer Science Student at Ulm University
- Submitted Master Thesis

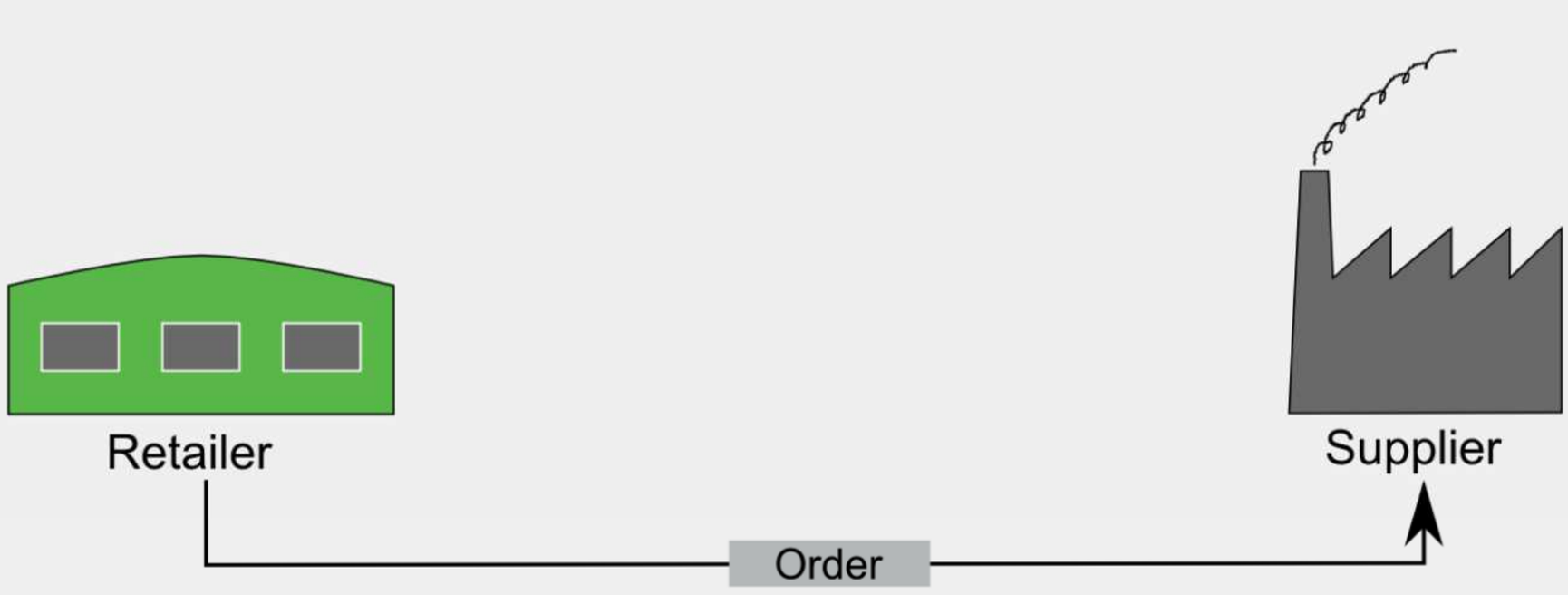
Dr. Jan Scheible
- MERCAREON's Employee
- Master Thesis Advisor

Mercareon

- Spin-Off of TRANSPOREON (Logistic software company)
- Located in Ulm and Kraków
- 28 Employees
- Main Product: Time Slot Management System

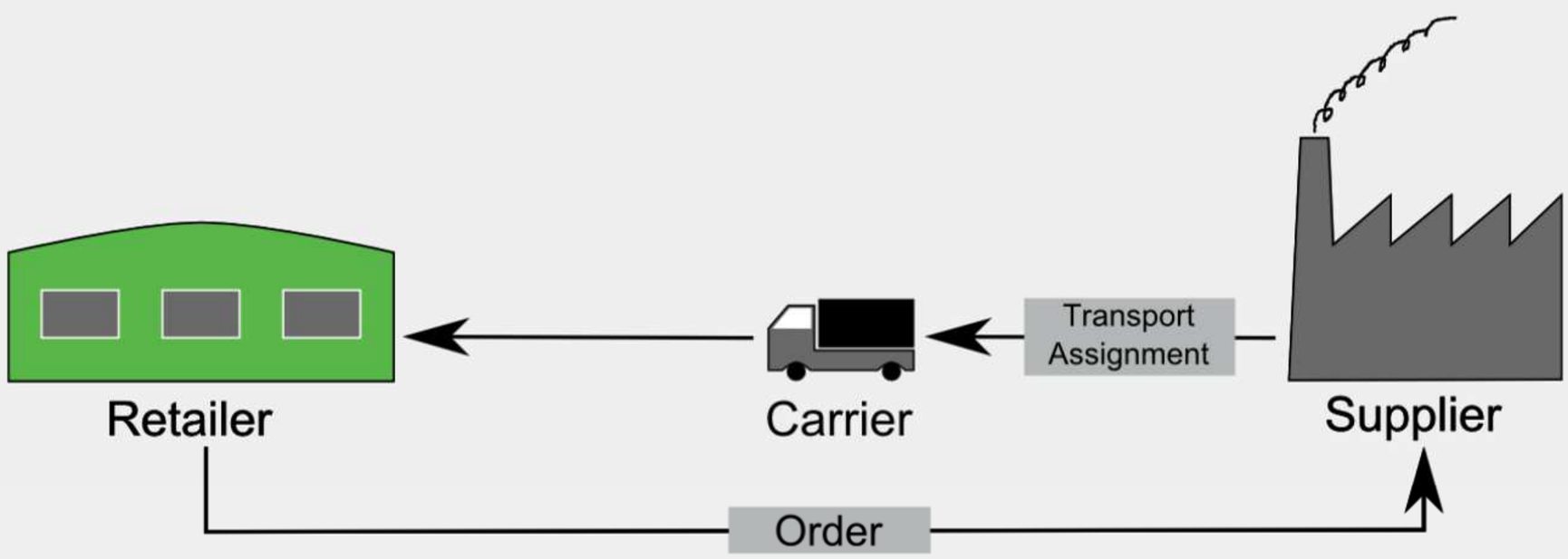
Mercareon

- Spin-Off of TRANSPOREON (Logistic software company)
- Located in Ulm and Kraków
- 28 Employees
- Main Product: Time Slot Management System



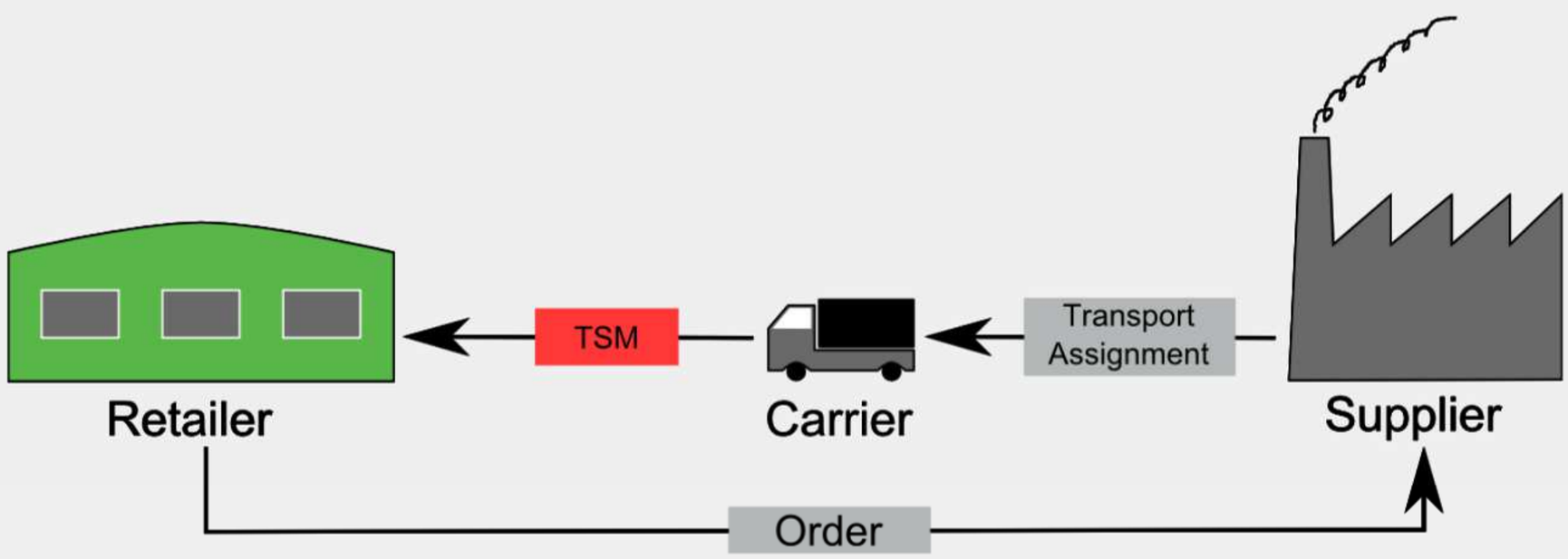
Mercareon

- Spin-Off of TRANSPOREON (Logistic software company)
- Located in Ulm and Kraków
- 28 Employees
- Main Product: Time Slot Management System



Mercareon

- Spin-Off of TRANSPOREON (Logistic software company)
- Located in Ulm and Kraków
- 28 Employees
- Main Product: Time Slot Management System



Agenda

Introduction

Challenges

Domain-Driven Design

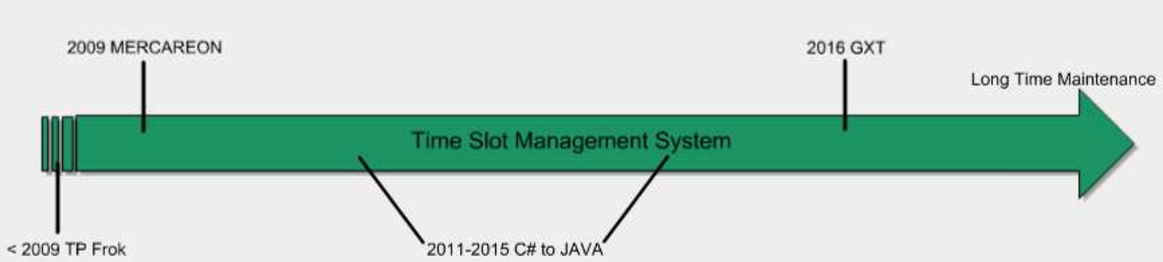
Refactoring

Prototype

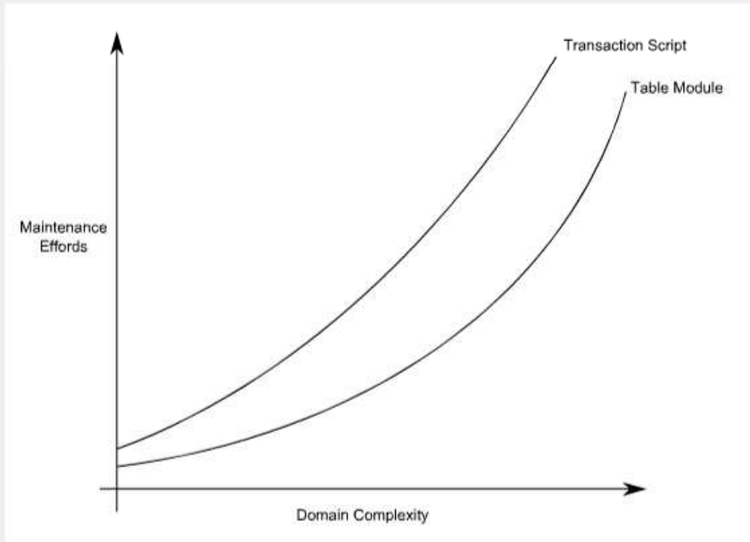
Challenges

MERCAREON

Long time maintenance
- Add new features
- Improve existing features

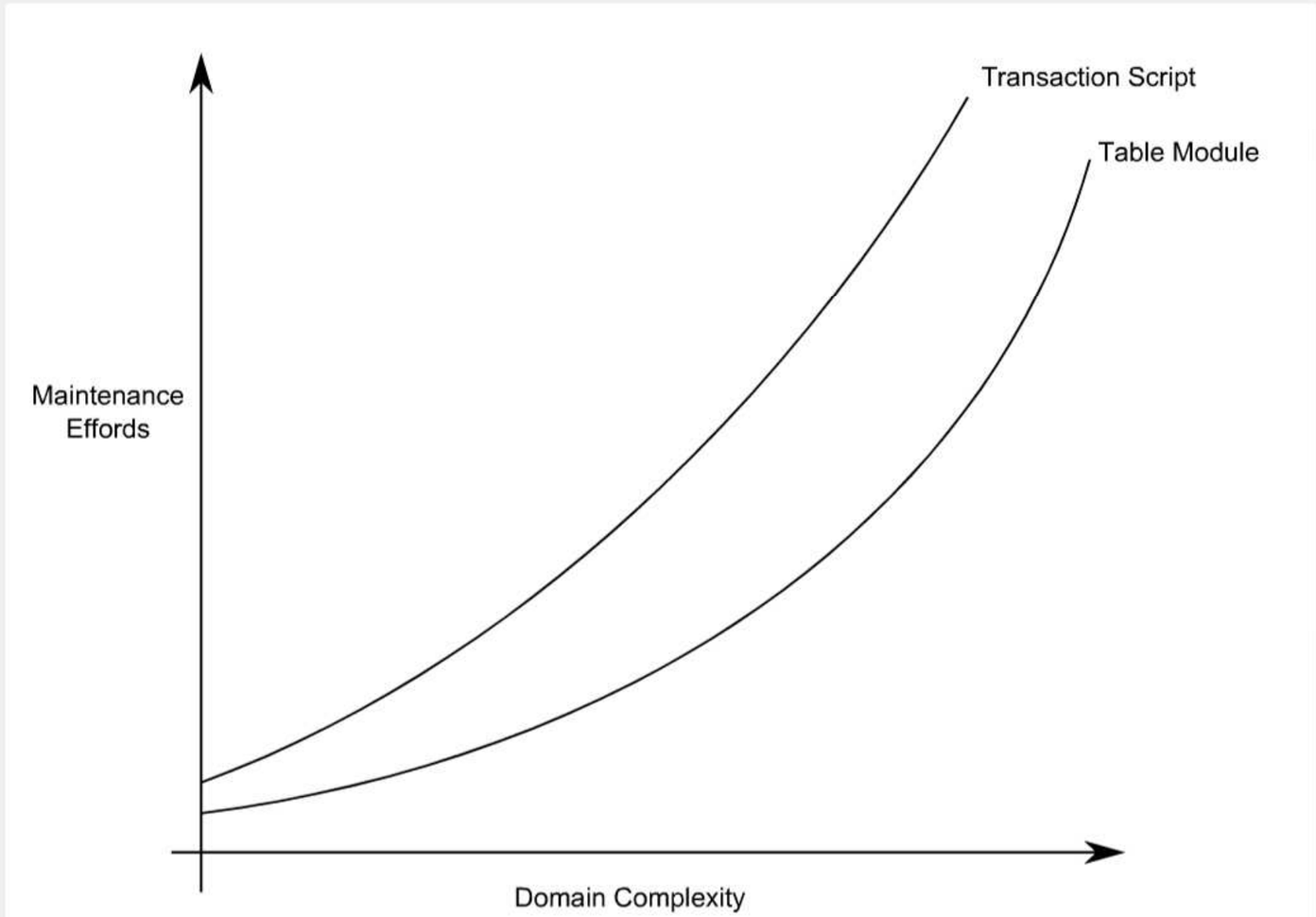


Maintenance Challenges



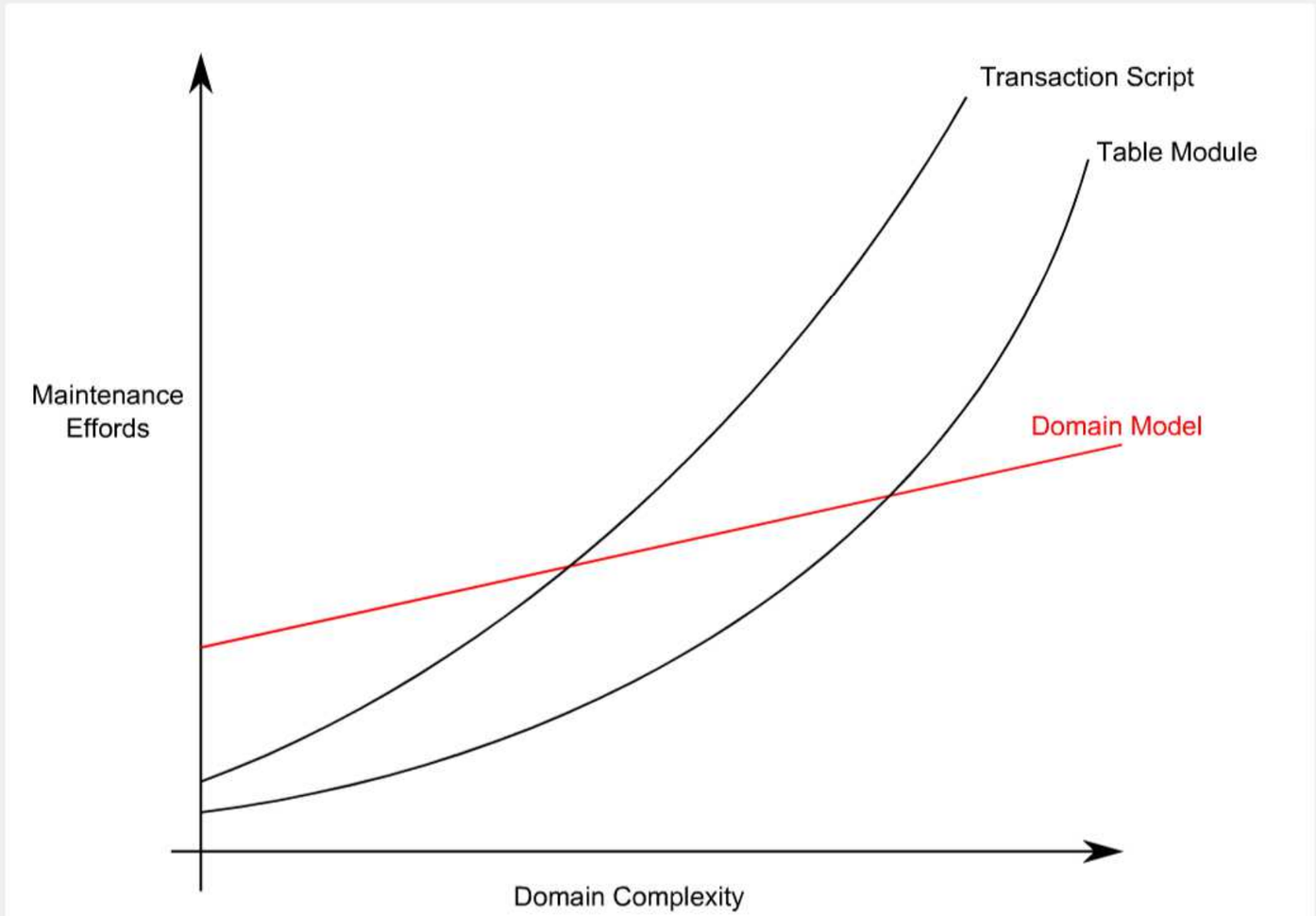
Source: Martin Fowler. *Patterns of enterprise application architecture.*

Maintenance Challenges



Source: Martin Fowler. *Patterns of enterprise application architecture*.

Maintenance Challenges

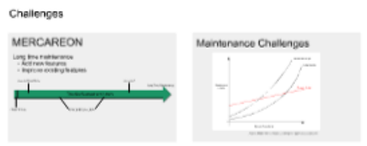


Source: Martin Fowler. *Patterns of enterprise application architecture.*

Agenda

Introduction

Challenges

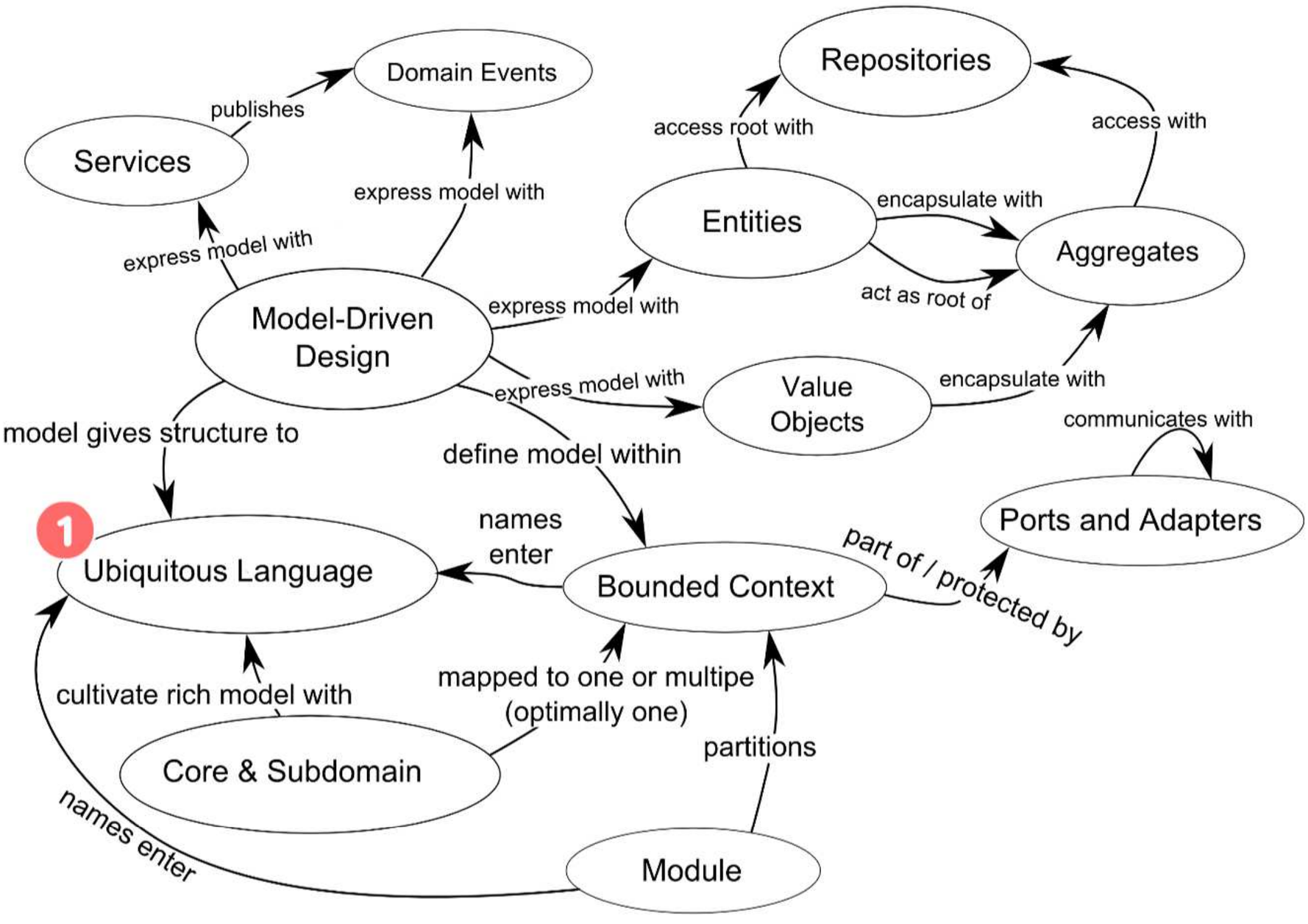


Domain-Driven Design

Refactoring

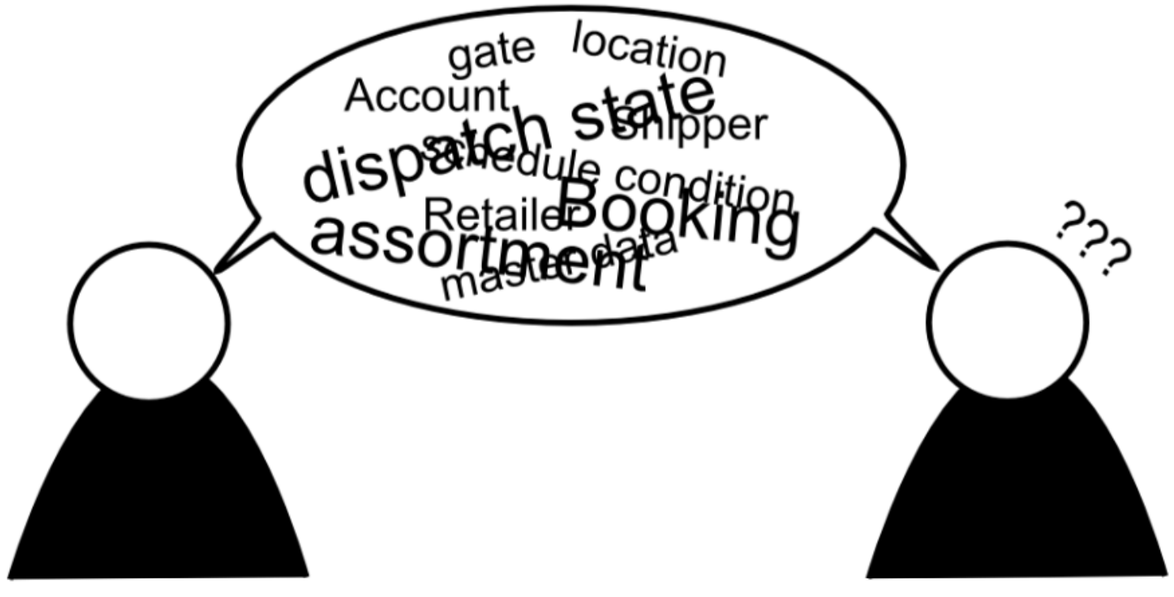
Prototype

Domain-Driven Design

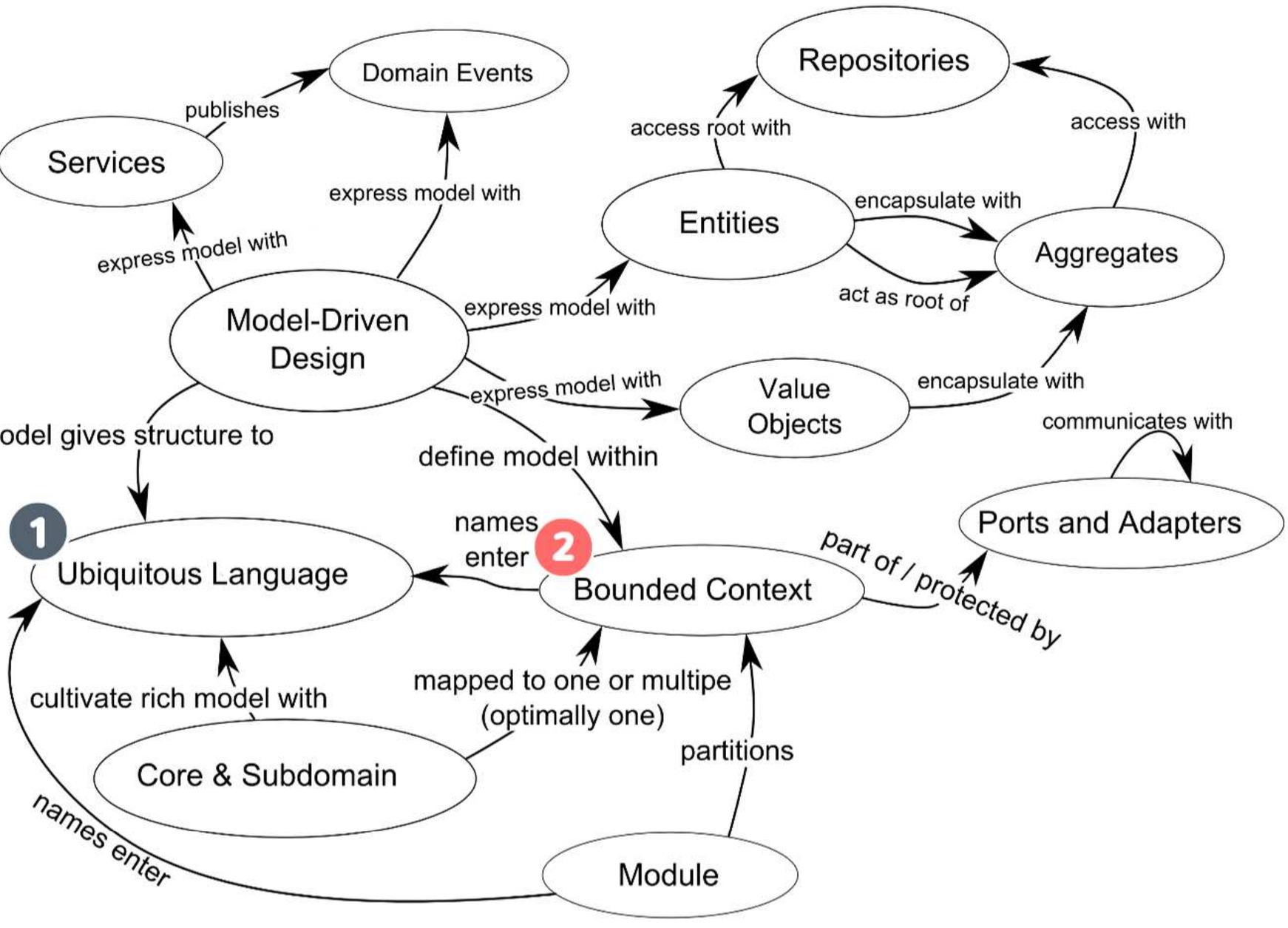


Ubiquitous Language

- Language shared by all involved parties
- Improves communication and prevents misunderstandings

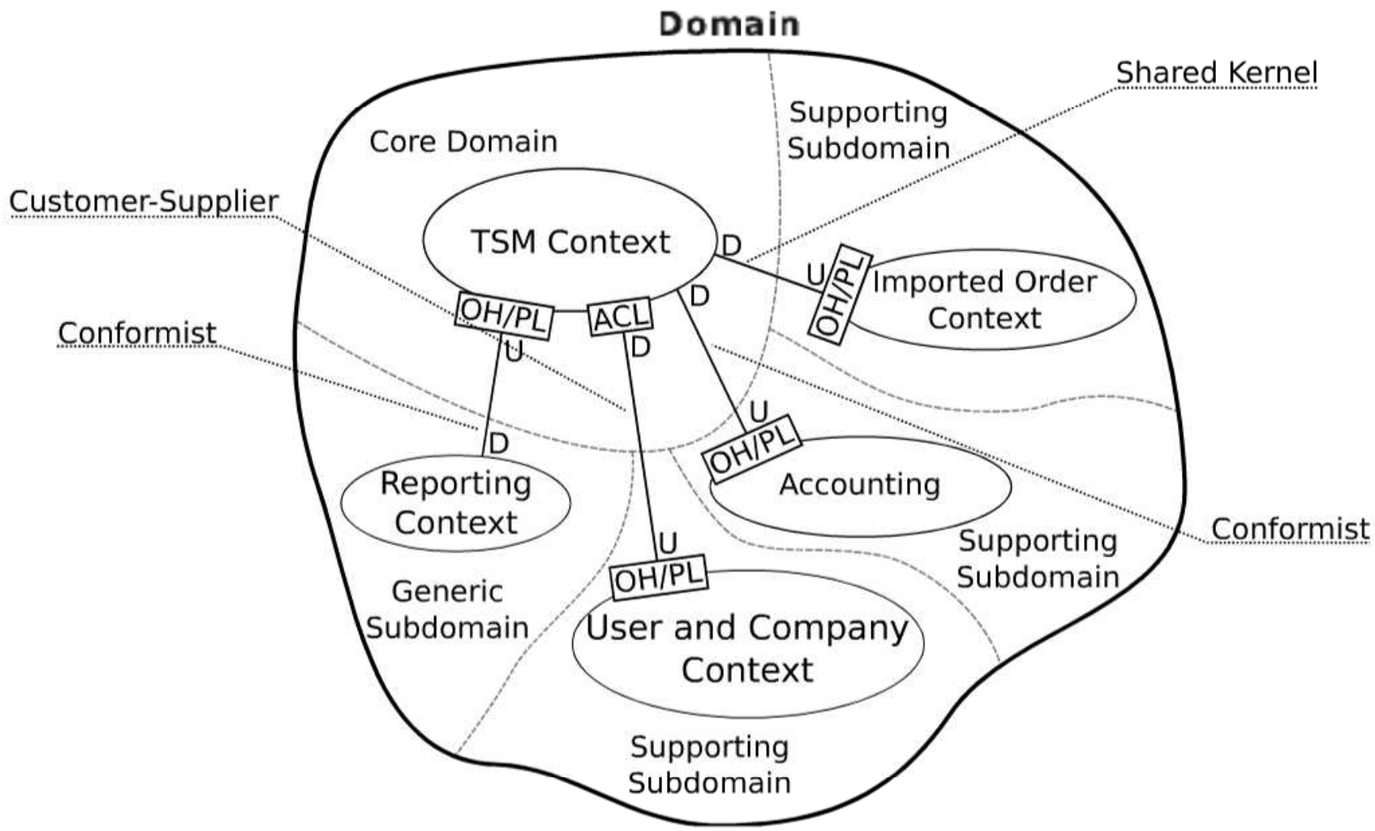


Domain-Driven Design

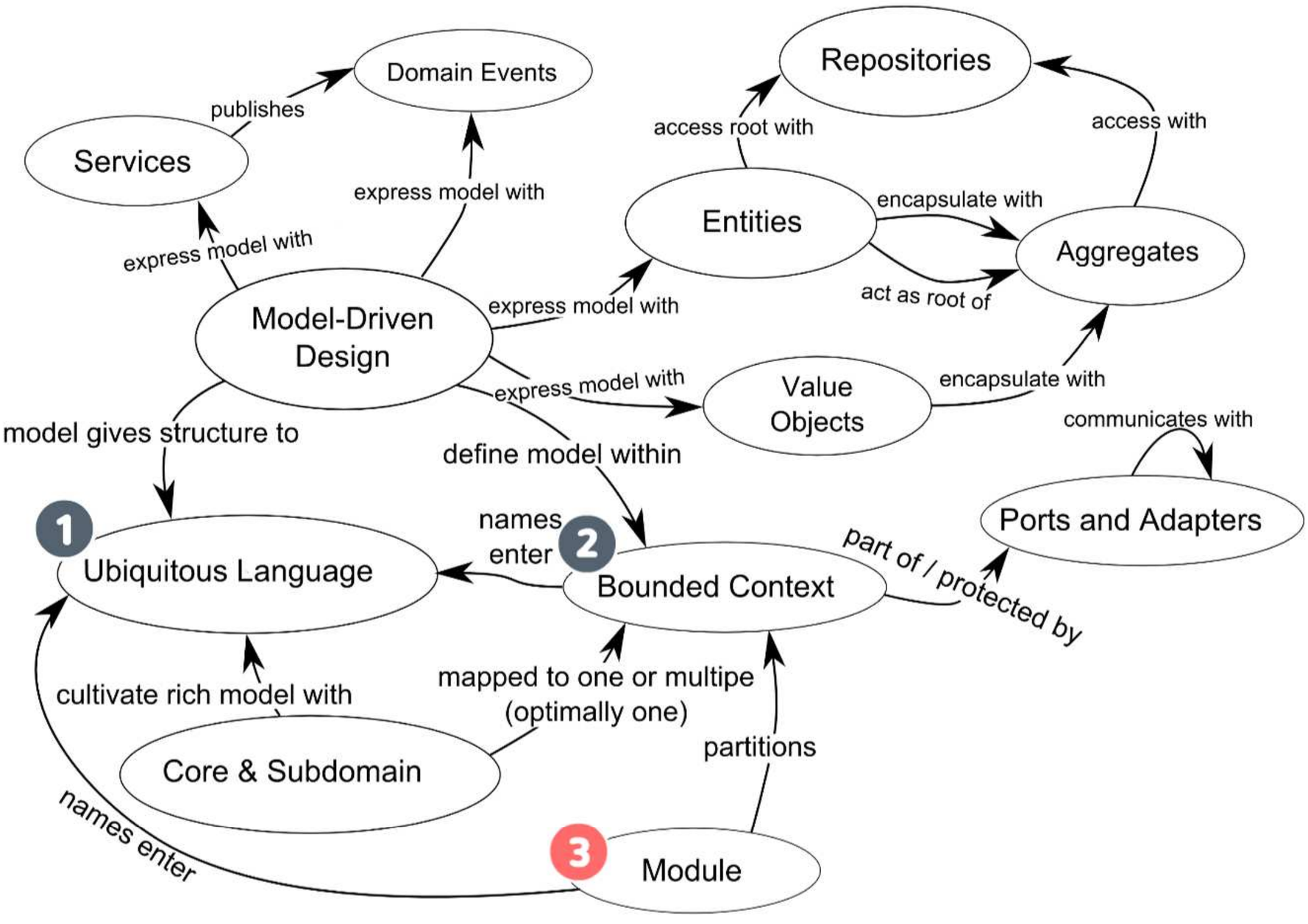


Bounded Contexts

- Ubiquitous Language Boundary
- Code, Database, Team Distribution Boundary

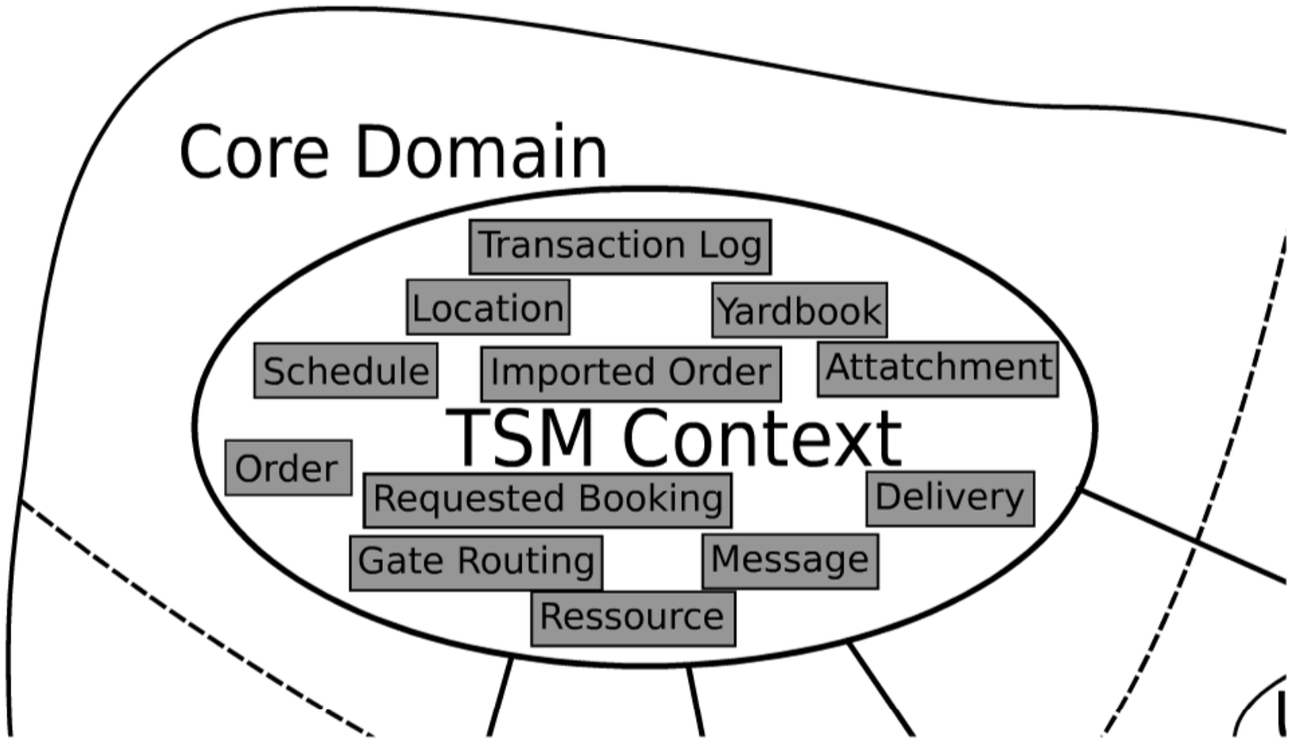


Domain-Driven Design

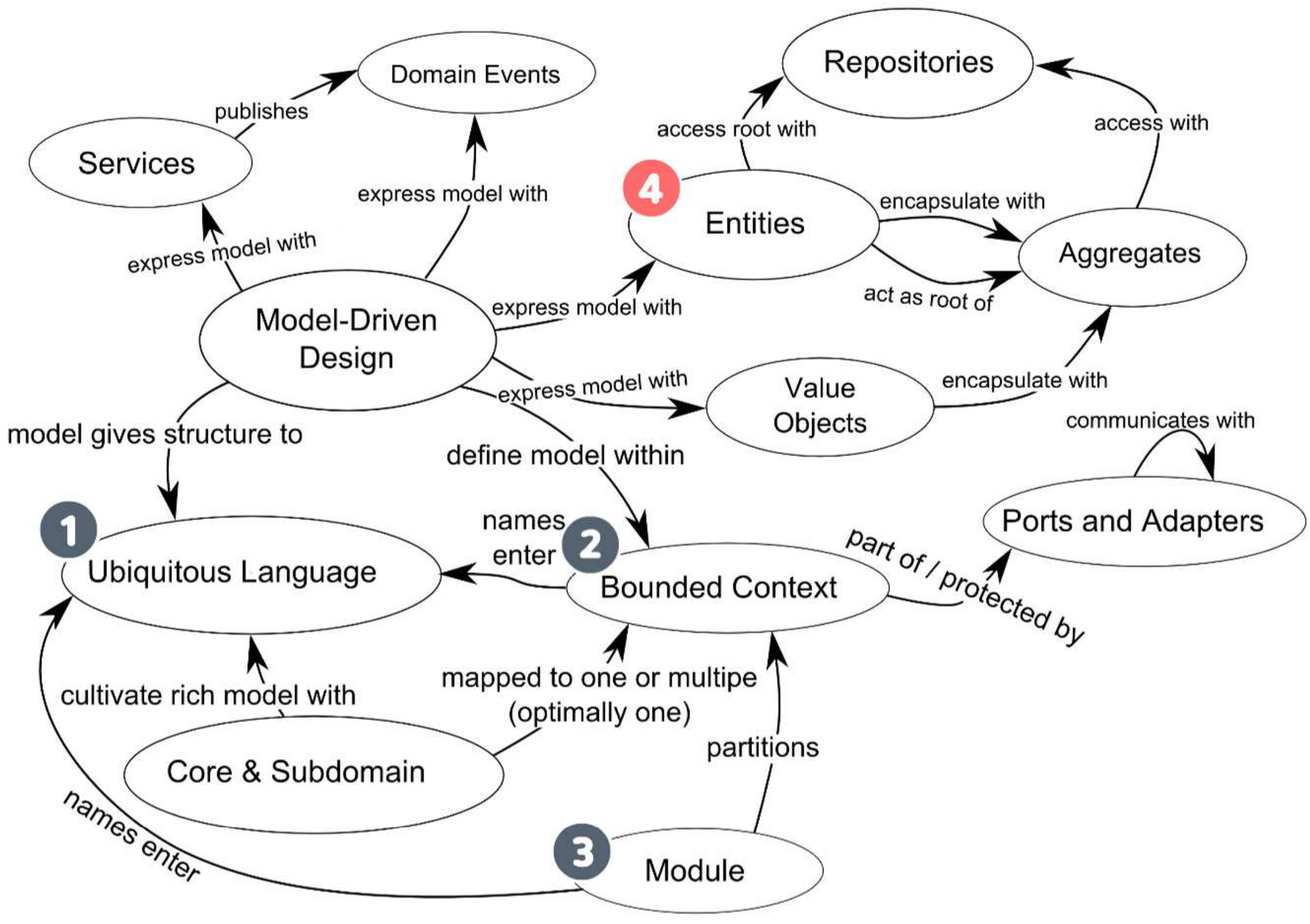


Modules

- Organize related concepts and tasks
- Reduce complexity
- Increase Cohesion
- Decrease Coupling



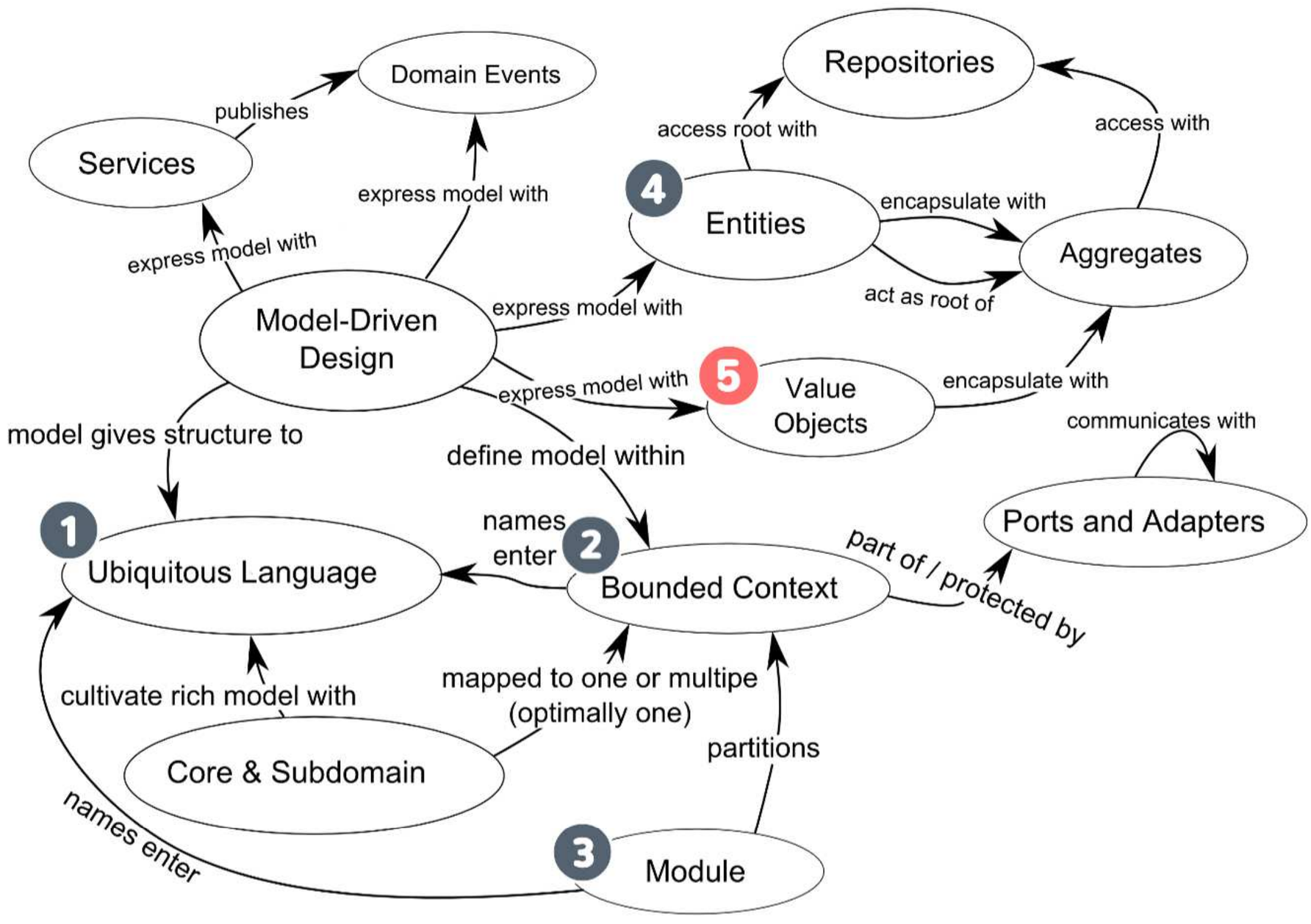
Domain-Driven Design



Entity

- Defined by its identity
- Unique beyond the life cycle of the system

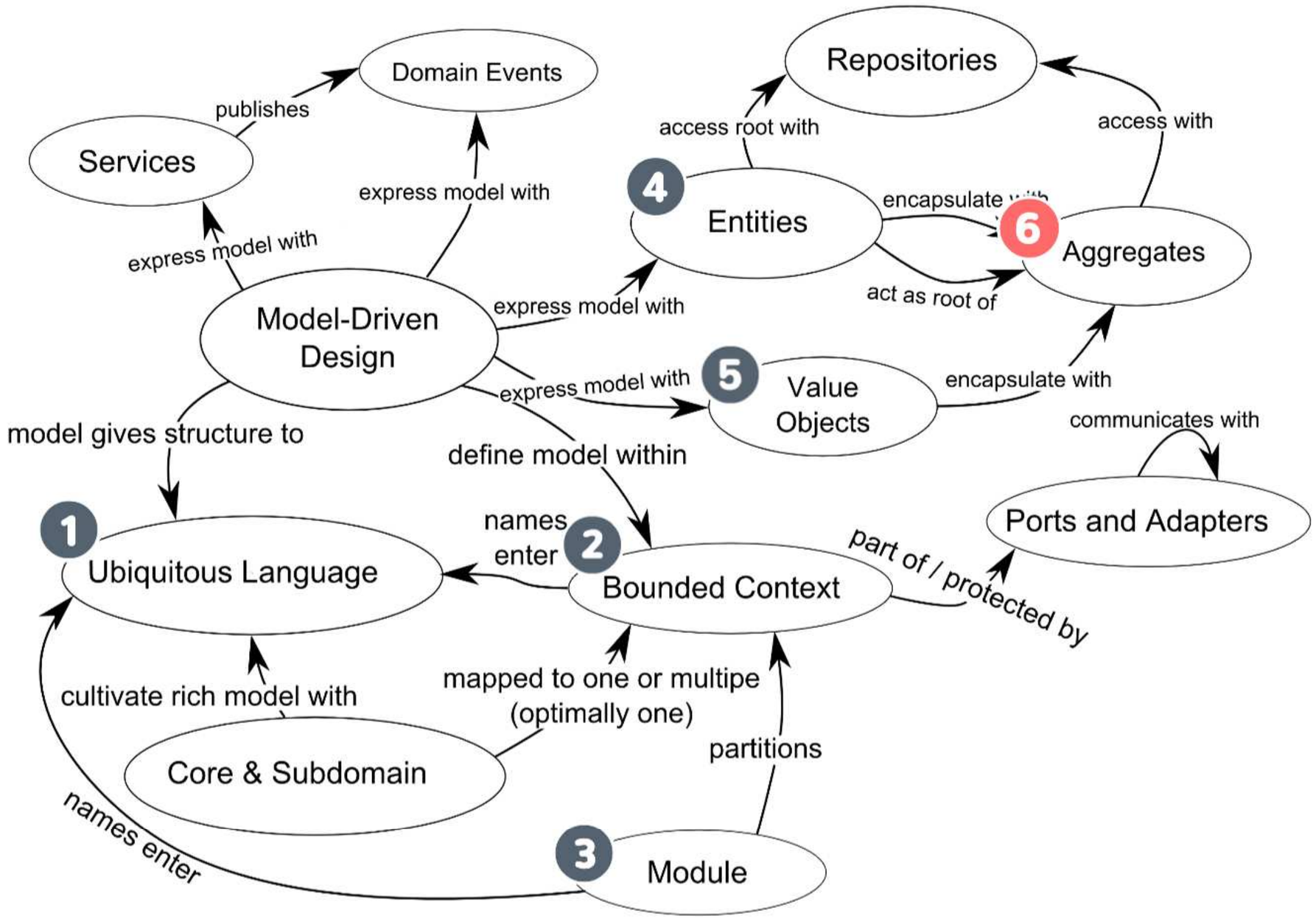
Domain-Driven Design



Value Objects

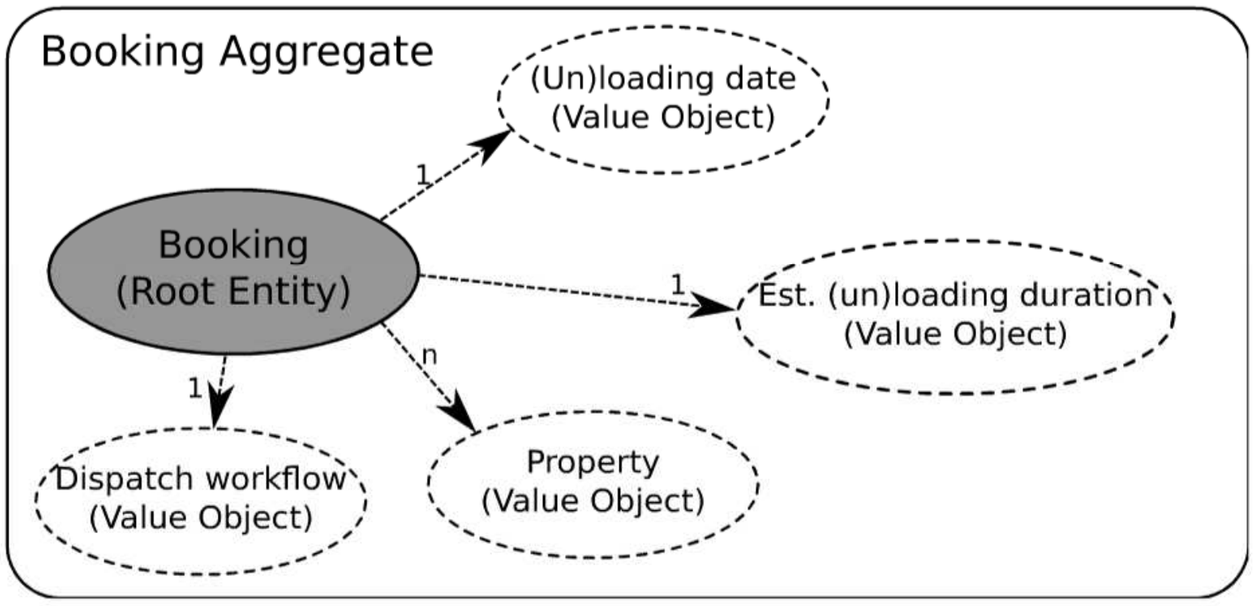
- Defined by its attributes
- Immutable
- When an attribute is changed, a new value object is instantiated

Domain-Driven Design

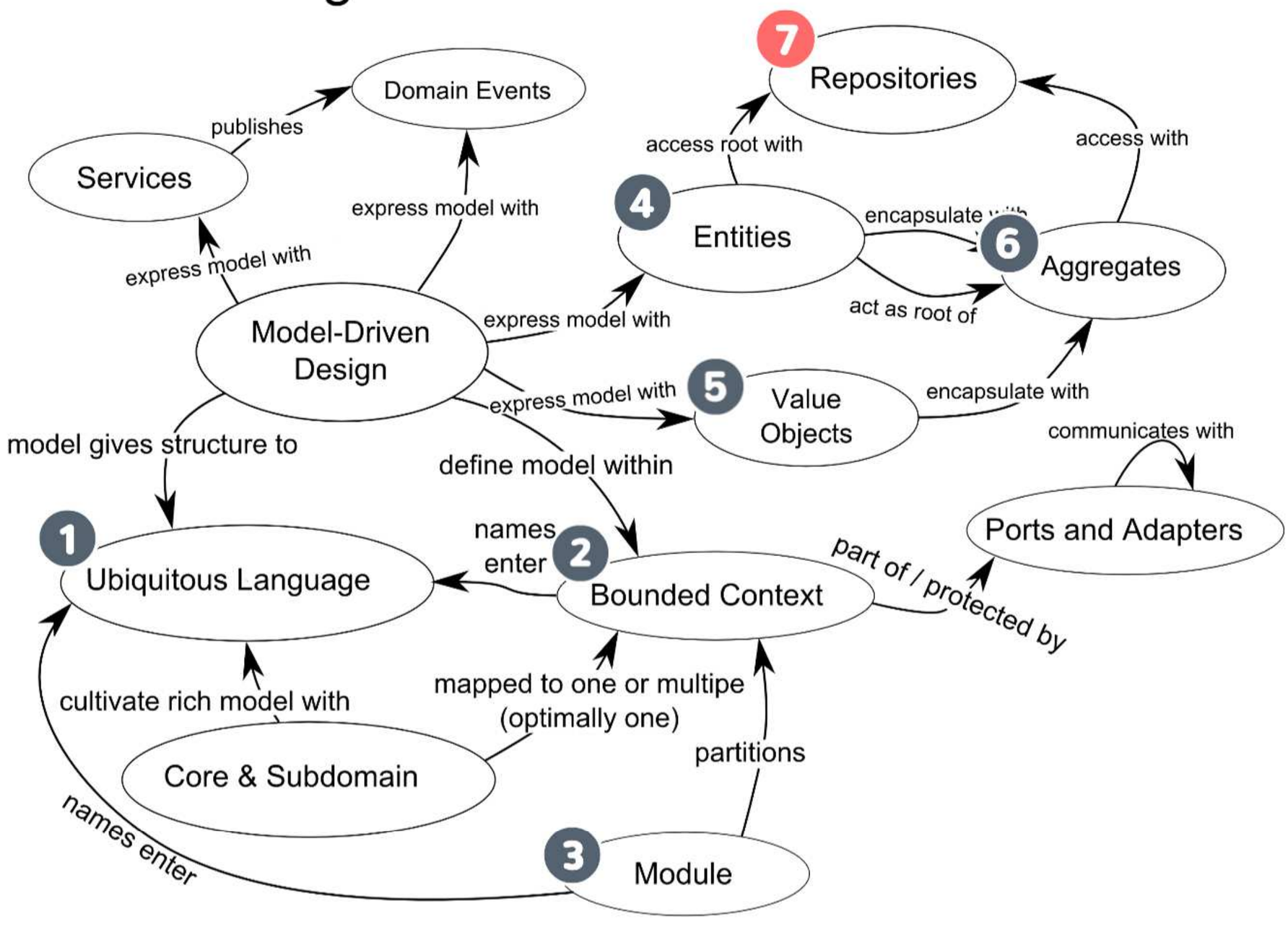


Aggregate

- One root entity
- Multiple entities
- Multiple value objects
- Consistency boundary
  - Transactions
  - Concurrency

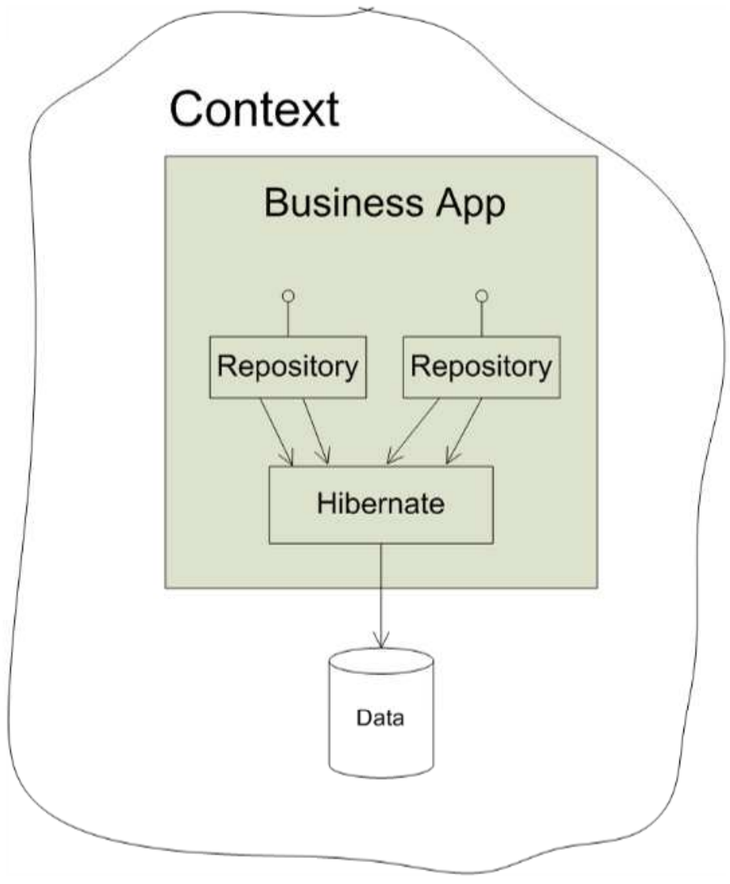


Domain-Driven Design

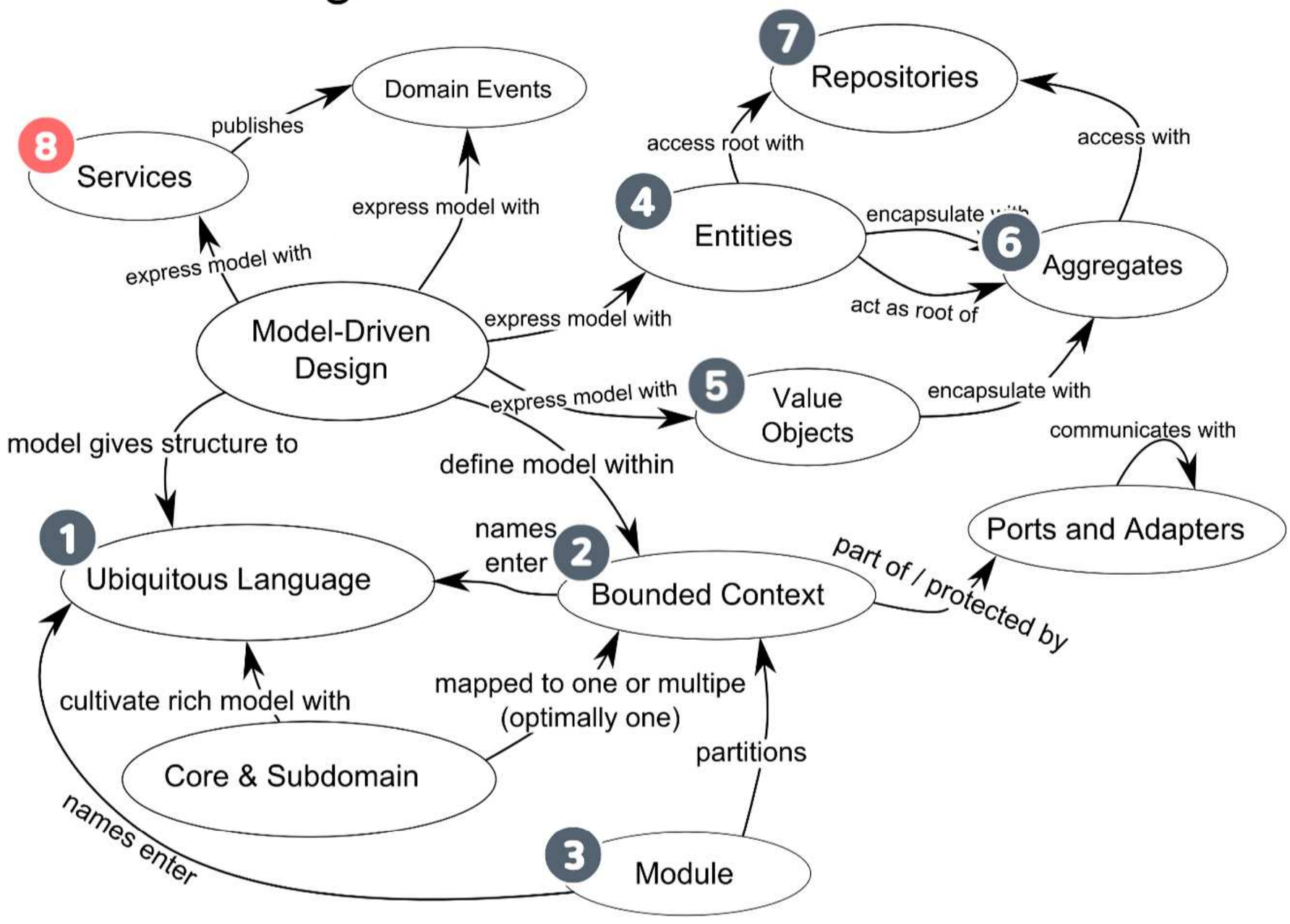


Repositories

- Provide access to aggregates
- Wraps data sources such as databases
- Implementation can be replaced for testing



Domain-Driven Design



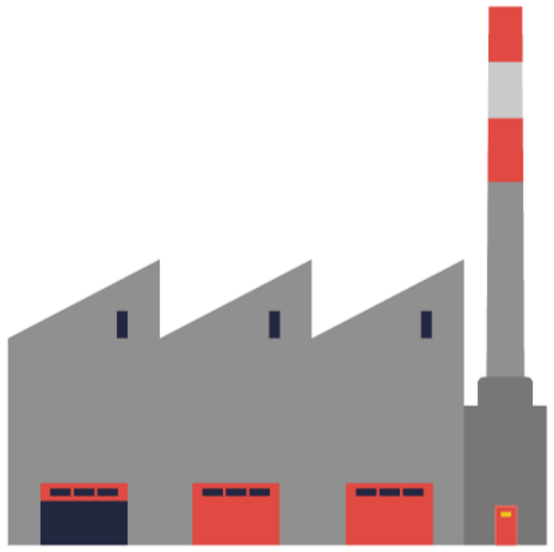
Services

Application Services

- Offer operations supported by the bounded context
- Can access repositories
- Controls the transactions & safety
- In charge of event based messaging

Domain Services

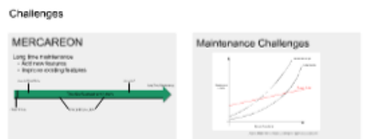
- Contains domain knowledge
- Encapsulates business logic that doesn't naturally fit within a domain object



Agenda

Introduction

Challenges



Domain-Driven Design

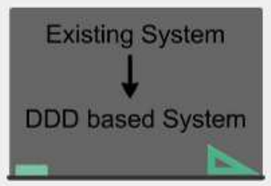
Refactoring

Prototype

Refactoring

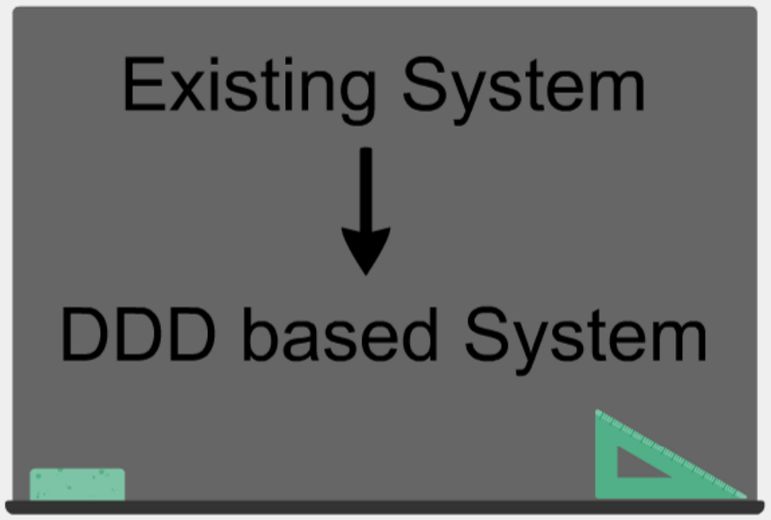
Challenge

- Domain-Driven Design was not designed for architectural refactoring
- How to derive entities, value objects, aggregates, object methods and services?



Challenge

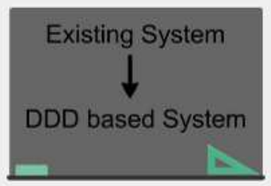
- Domain-Driven Design was not designed for architectural refactoring
- How to derive entities, value objects, aggregates, object methods and services?



Refactoring

Challenge

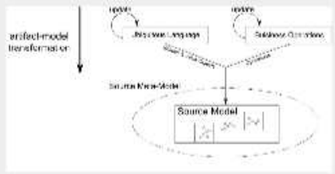
- Domain-Driven Design was not designed for architectural refactoring
- How to derive entities, value objects, aggregates, object methods and services?



Solution: Model Transformations

Artifact - Model Transformation:
- Ubiquitous Language
- Buisiness Operations

> Source Model

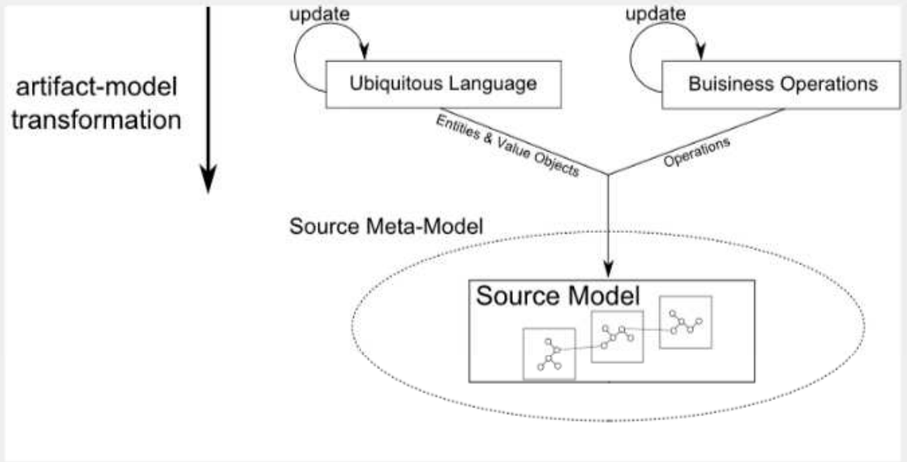


Solution: Model Transformations

Artifact - Model Transformation:

- Ubiquitous Language
- Buisiness Operations

> Source Model



Solution: Model Transformations

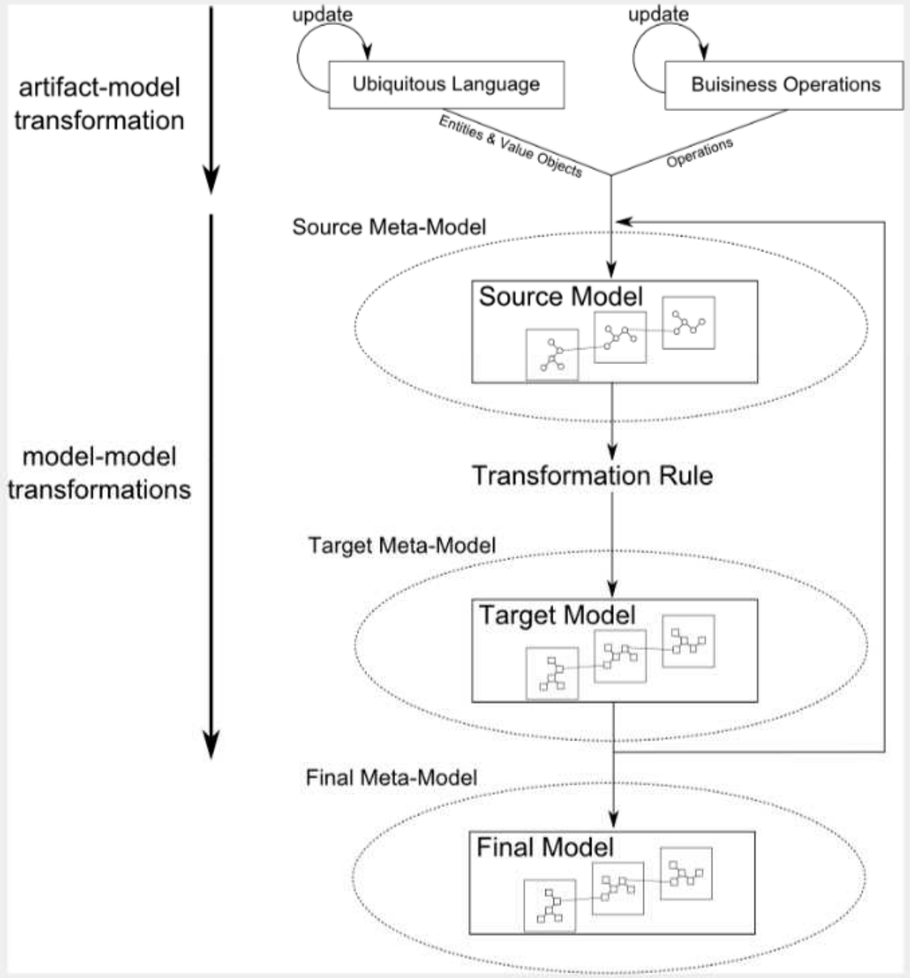
Artifact - Model Transformation:

- Ubiquitous Language
- Buisiness Operations

> Source Model

Model - Model Transformations:

- Source Model > Final Model



Solution: Model Transformations

Artifact - Model Transformation:

- Ubiquitous Language
- Buisiness Operations

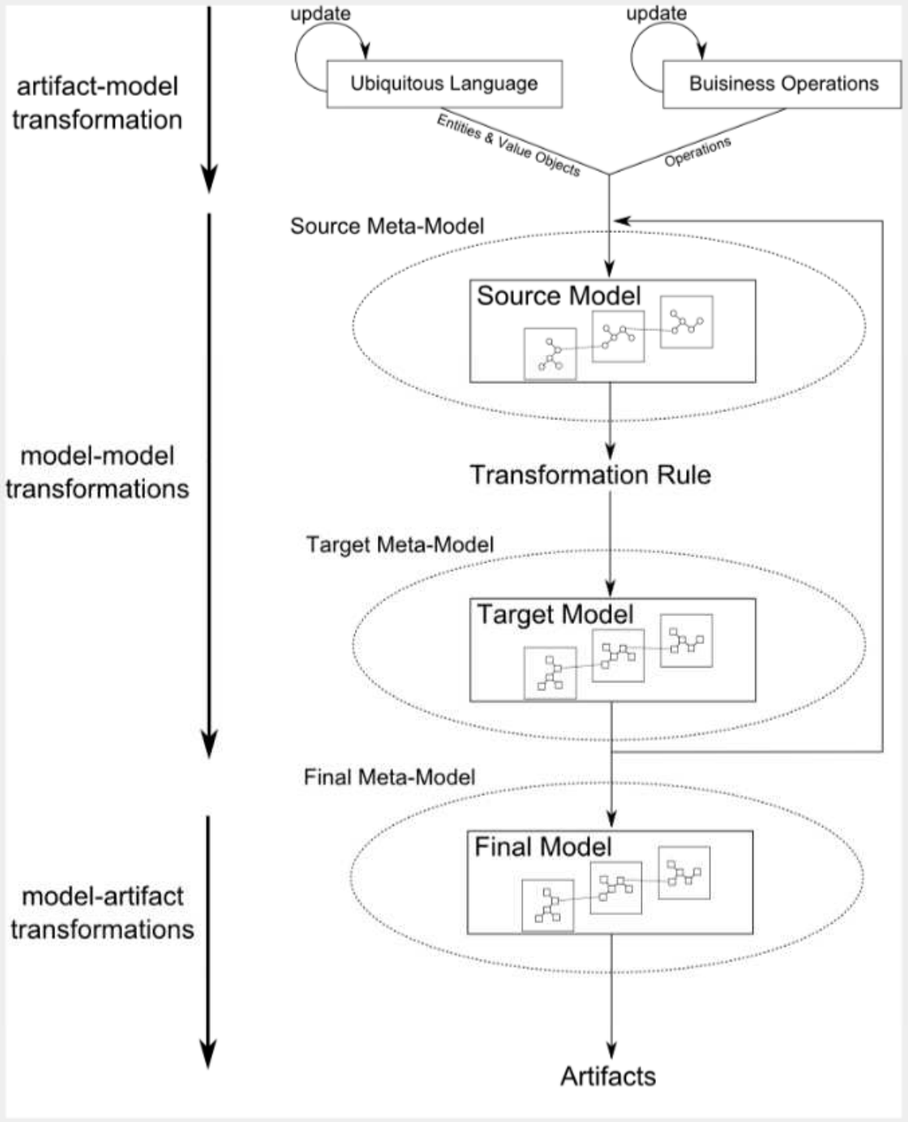
> Source Model

Model - Model Transformations:

- Source Model > Final Model

Model - Artifact Transformations:

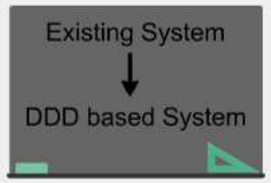
- Final Model > Visualizations



Refactoring

Challenge

- Domain-Driven Design was not designed for architectural refactoring
- How to derive entities, value objects, aggregates, object methods and services?



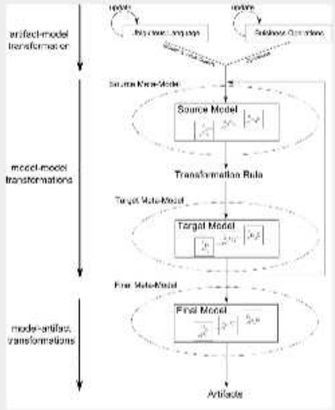
Solution: Model Transformations

Artifact - Model Transformation:
- Ubiquitous Language
- Buisiness Operations

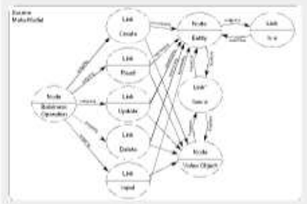
> Source Model

Model - Model Transformations:
- Source Model > Final Model

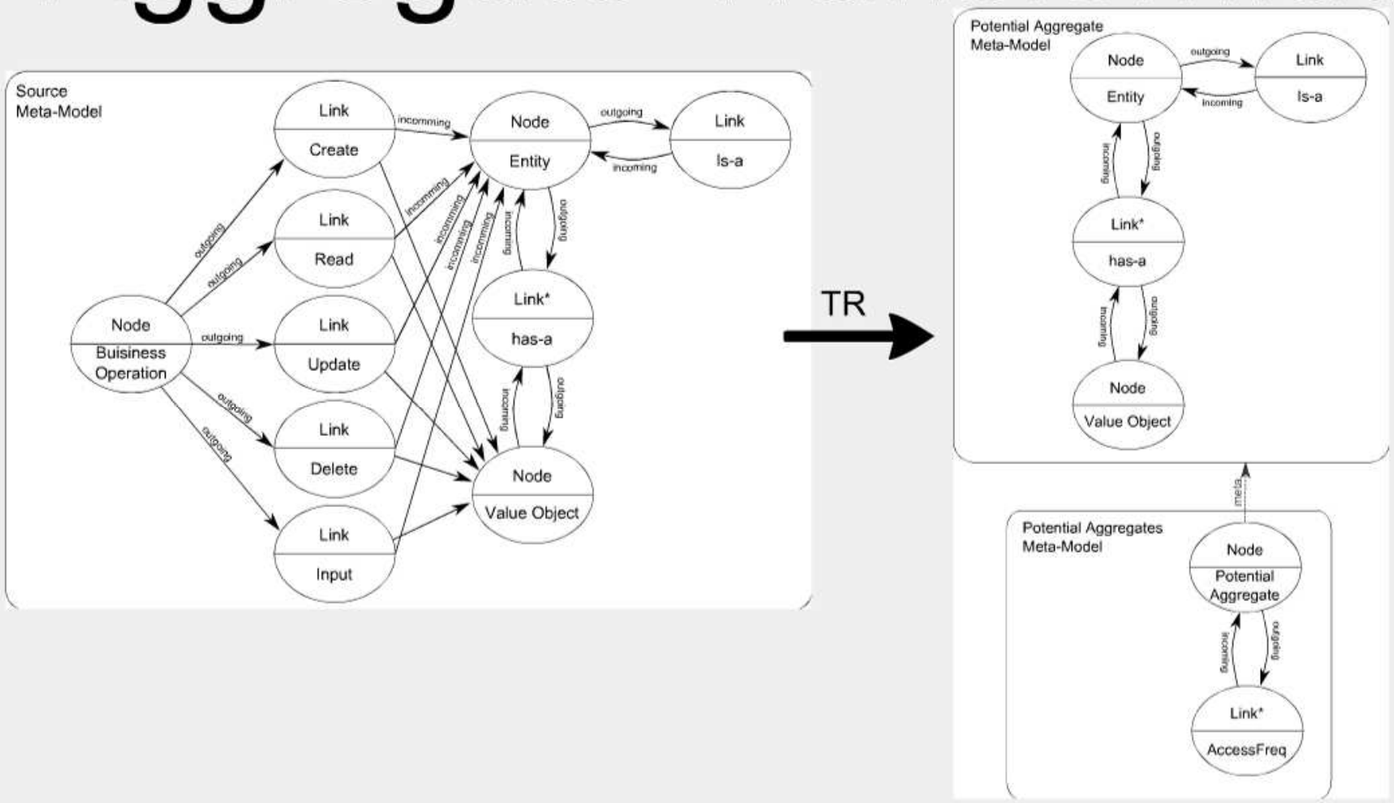
Model - Artifact Transformations:
- Final Model > Visualizations



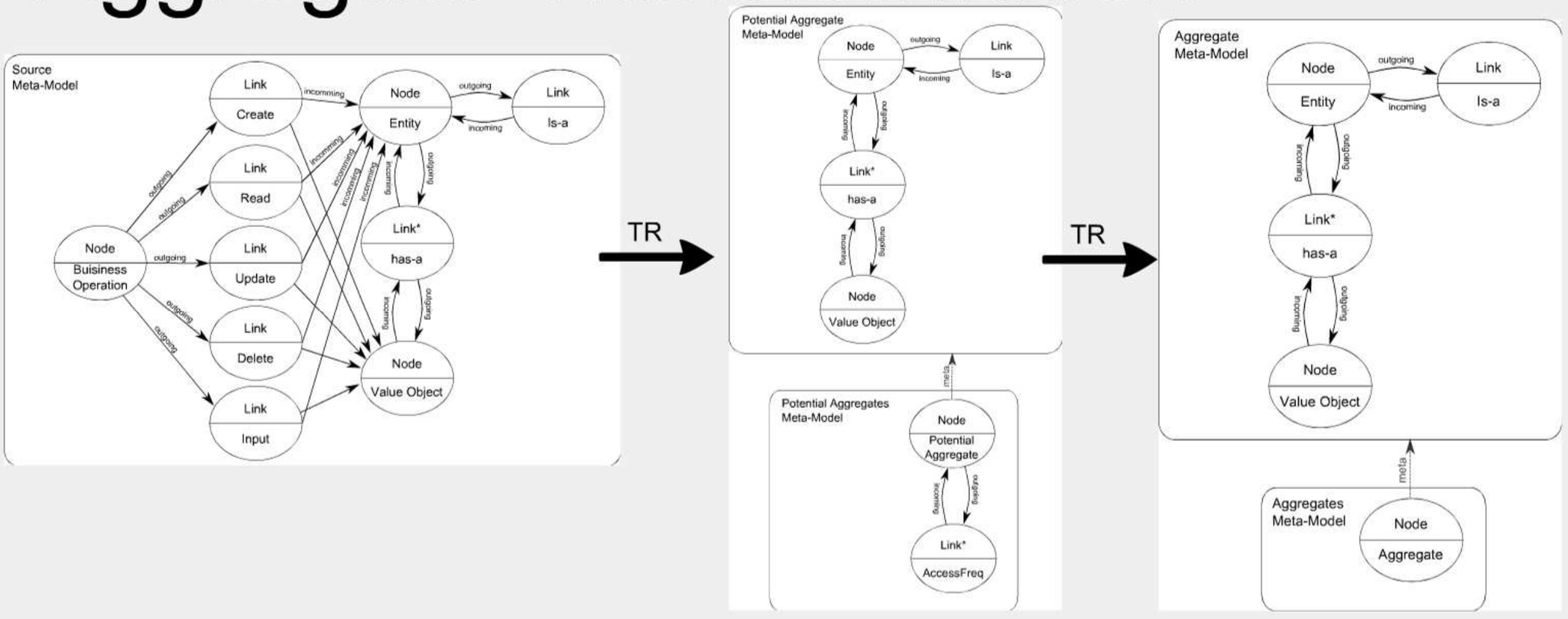
Aggregate Transformation



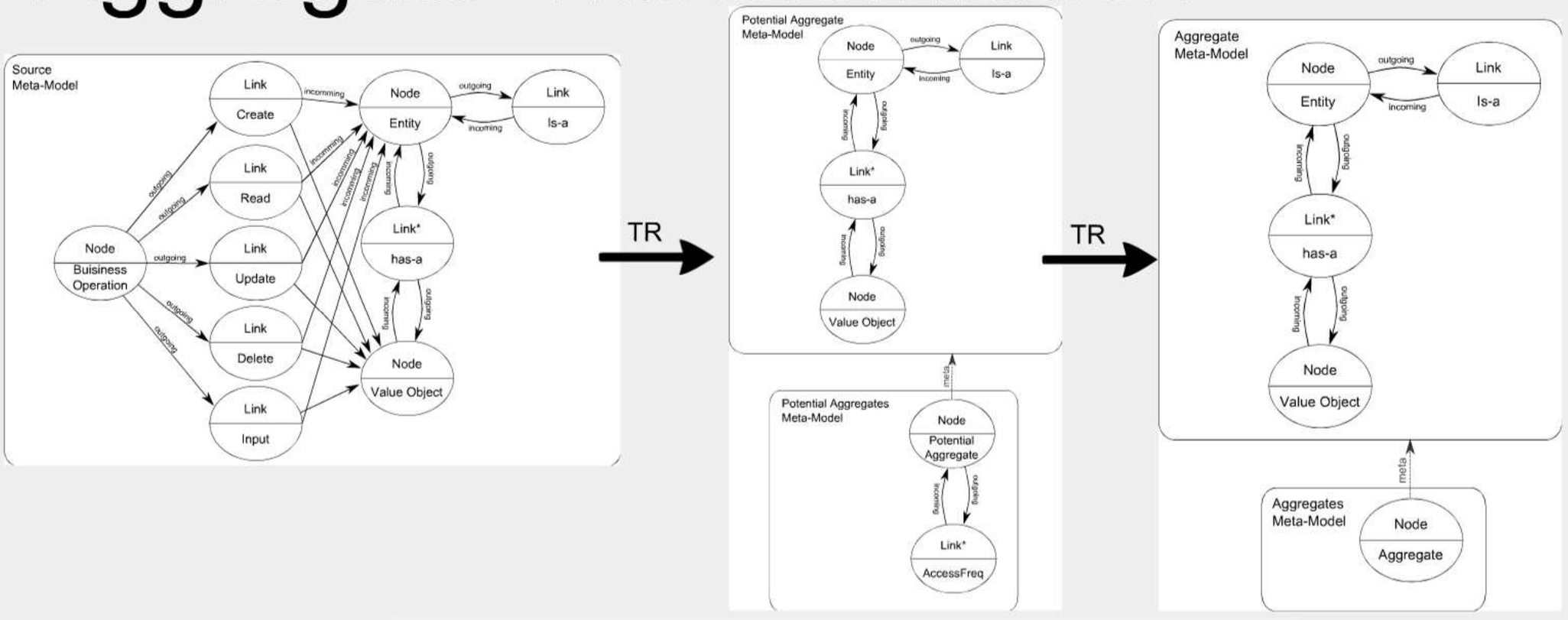
Aggregate Transformation



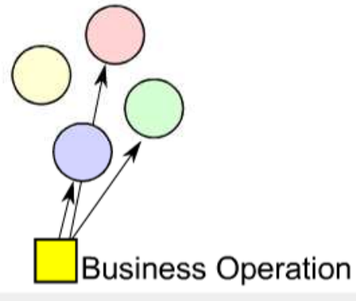
Aggregate Transformation



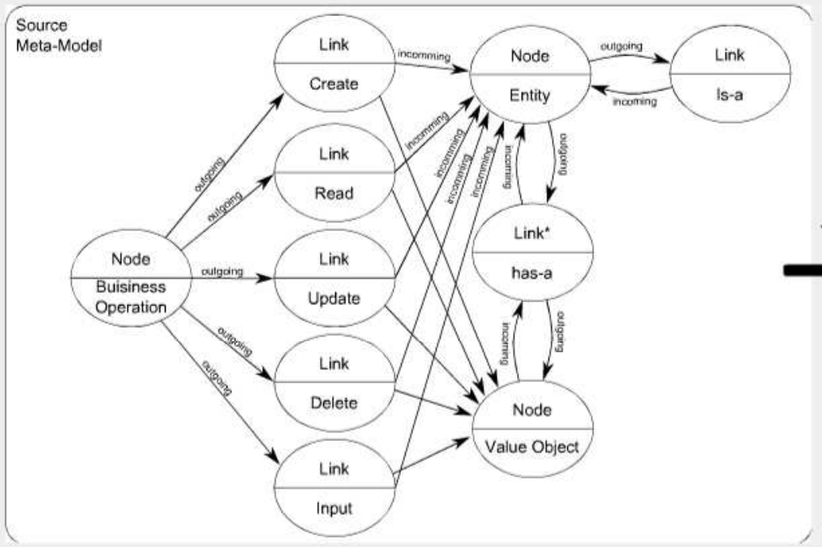
Aggregate Transformation



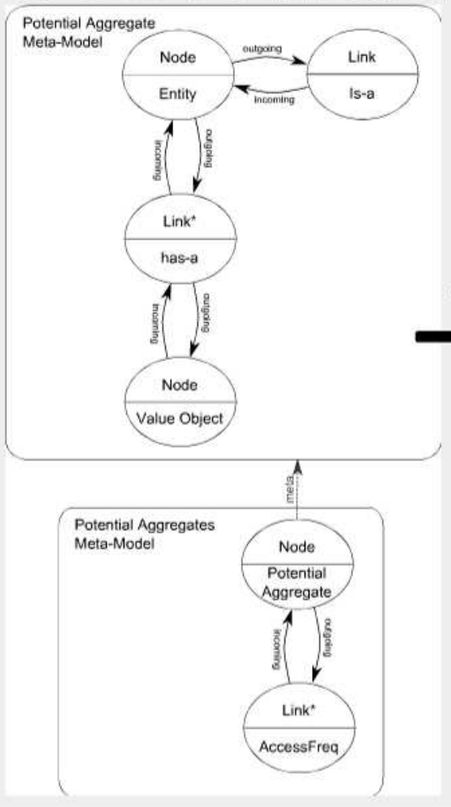
Entity & Value Objects



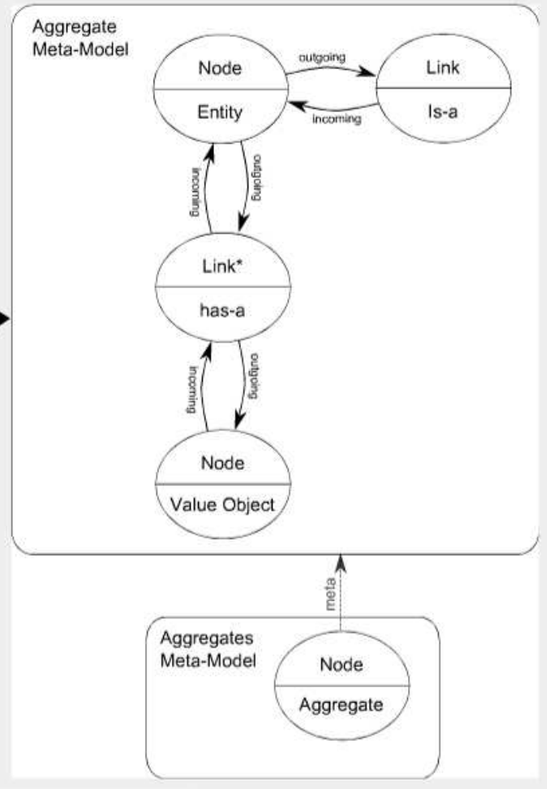
Aggregate Transformation

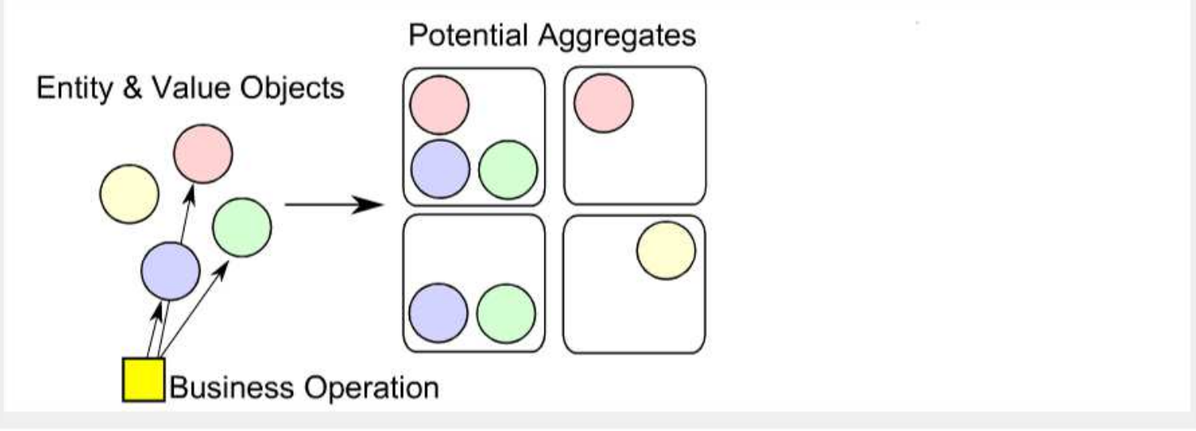


TR

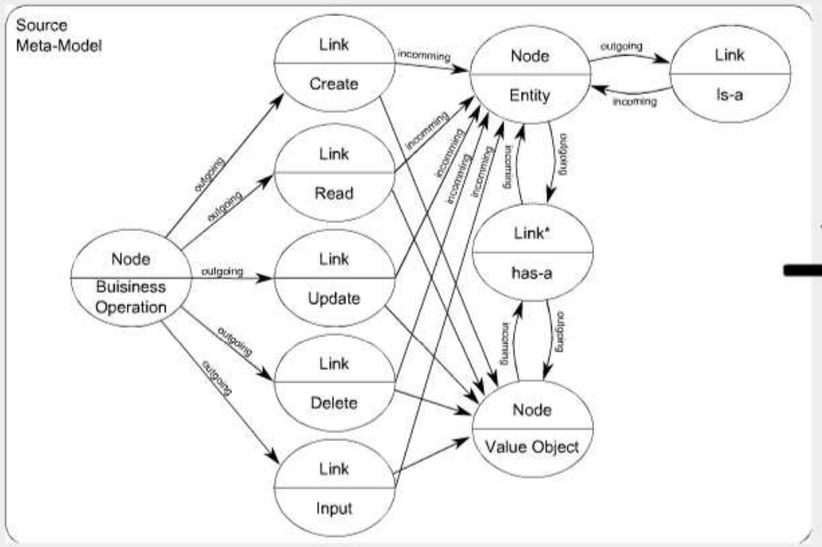


TR

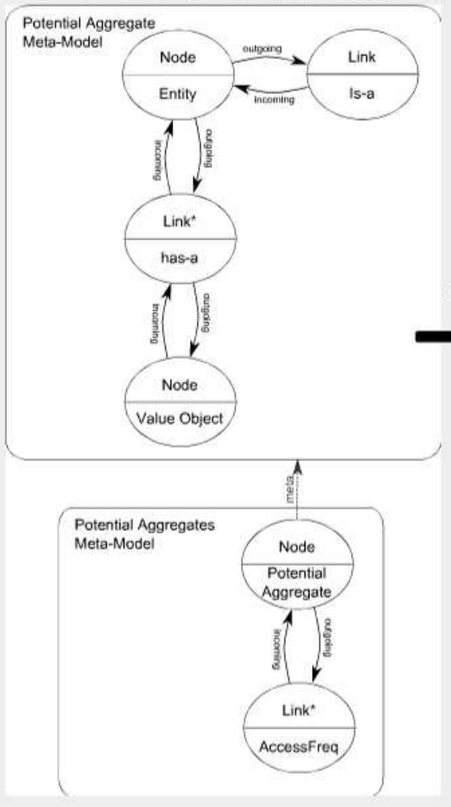




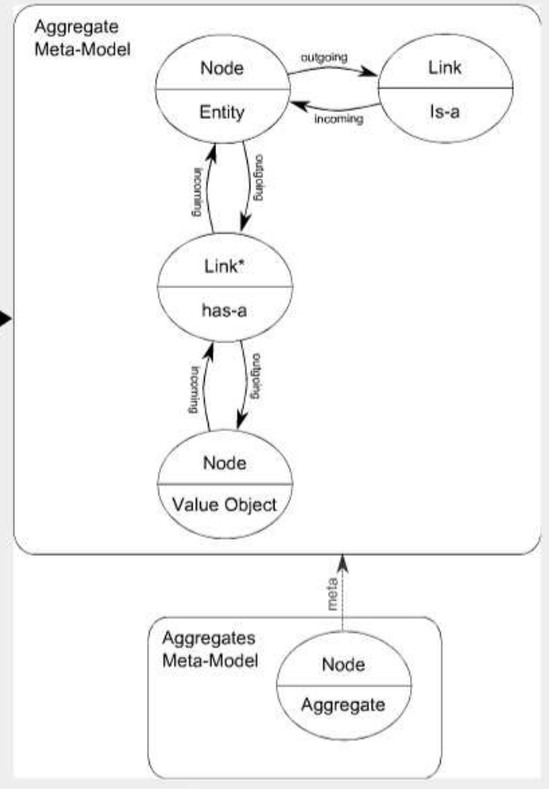
Aggregate Transformation

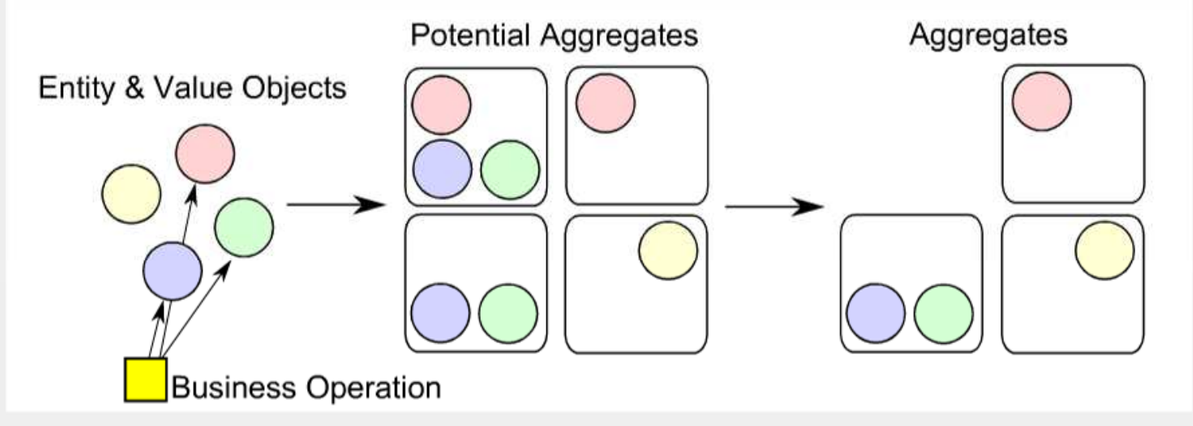


TR

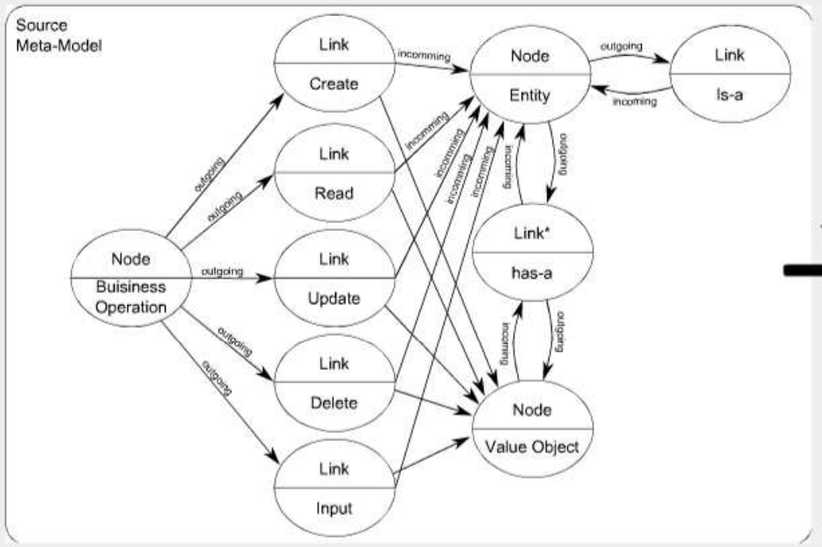


TR

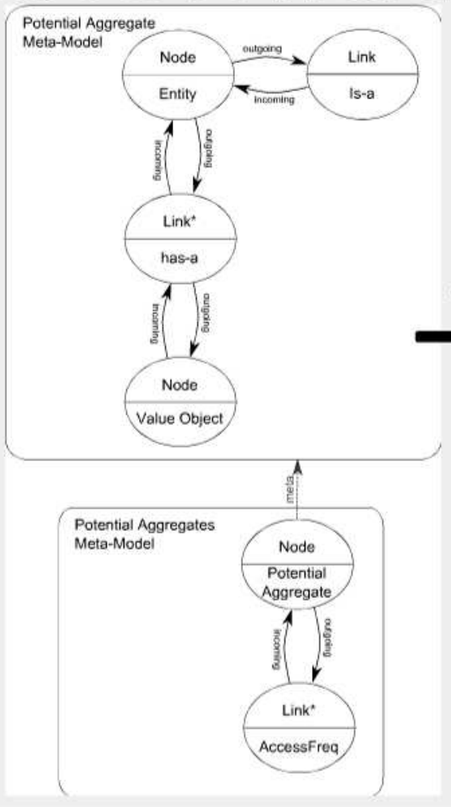




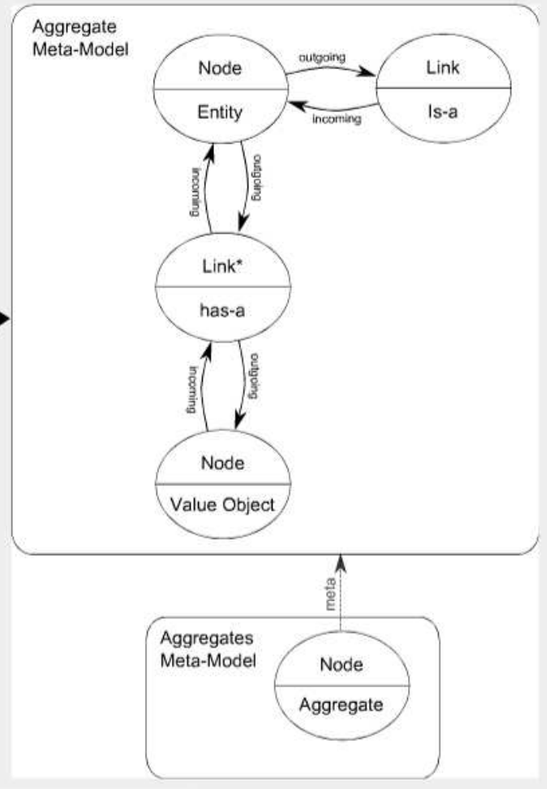
Aggregate Transformation

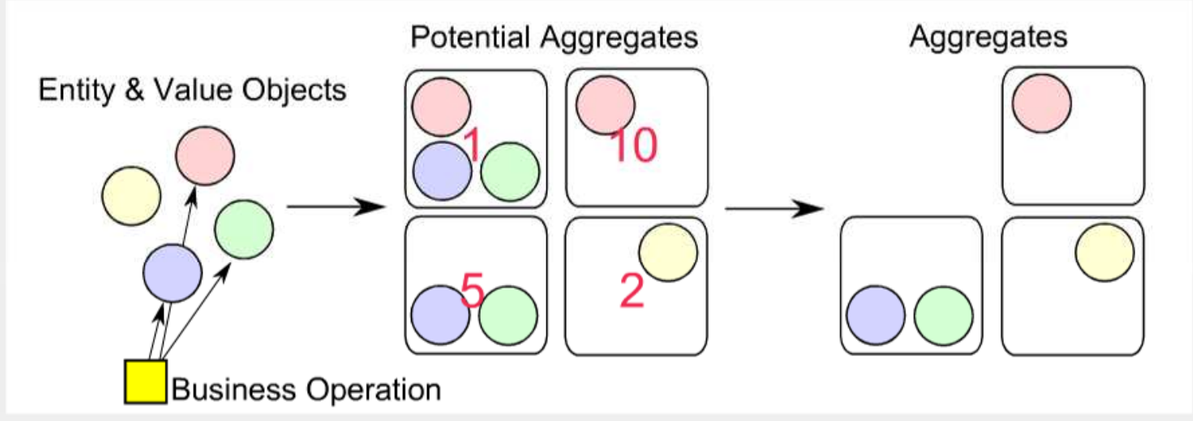


TR

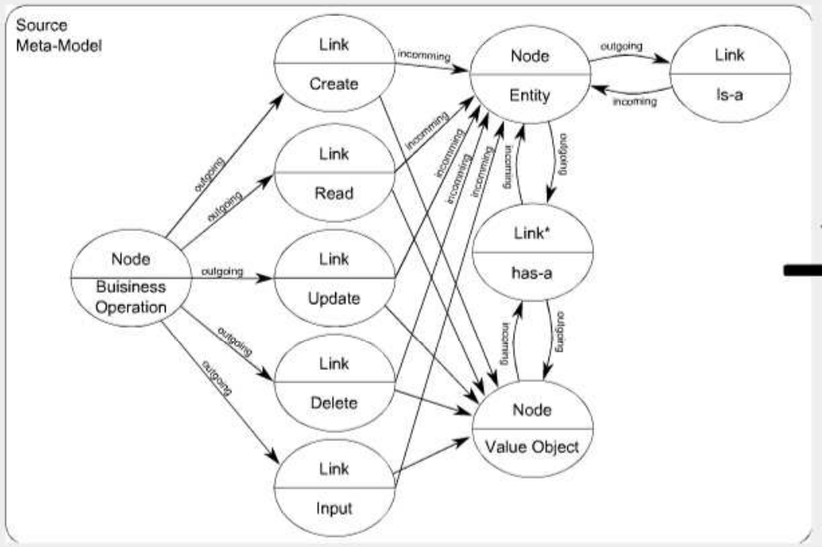


TR

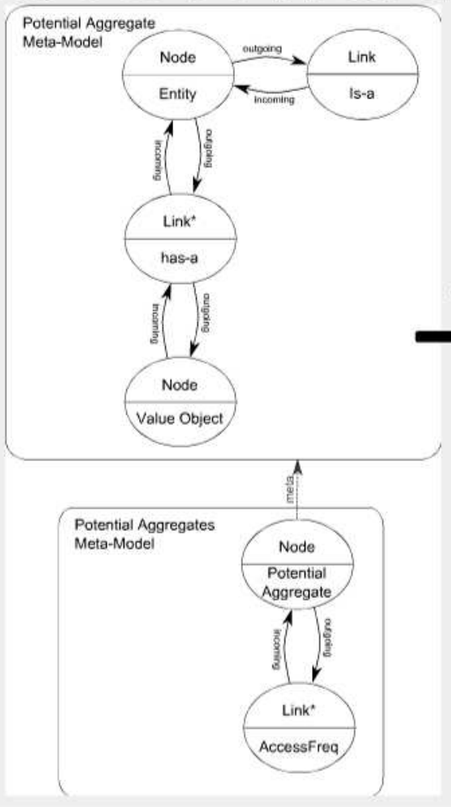




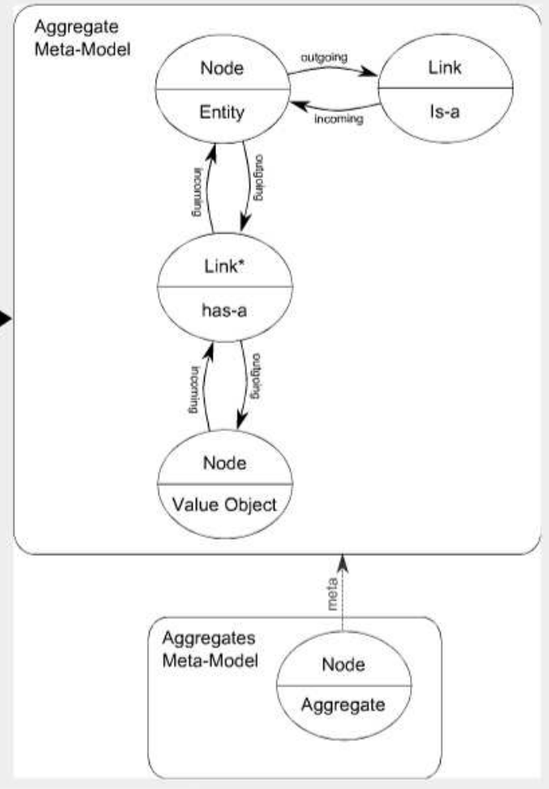
Aggregate Transformation

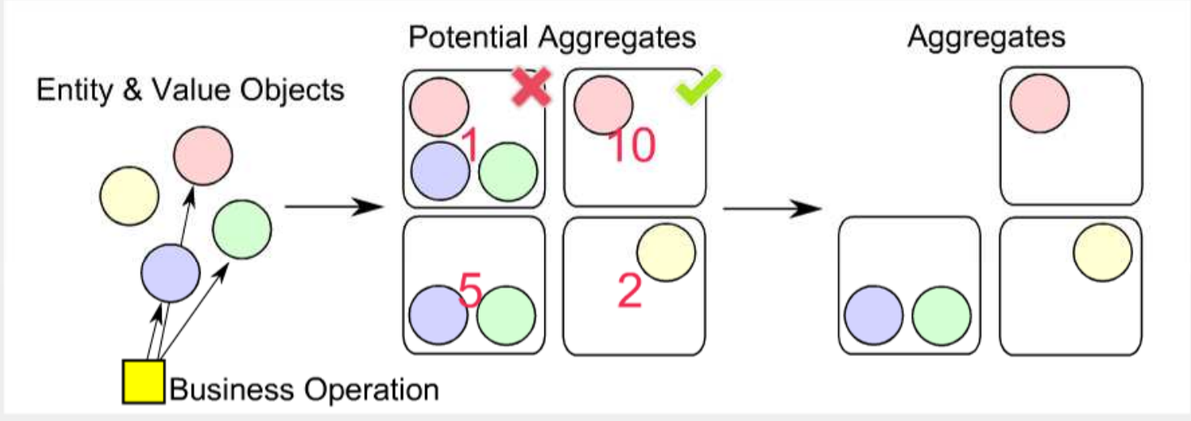


TR

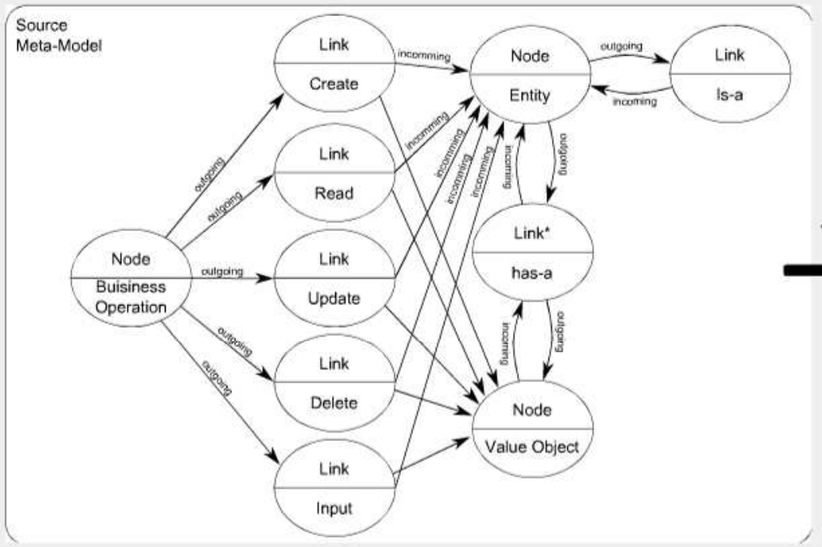


TR

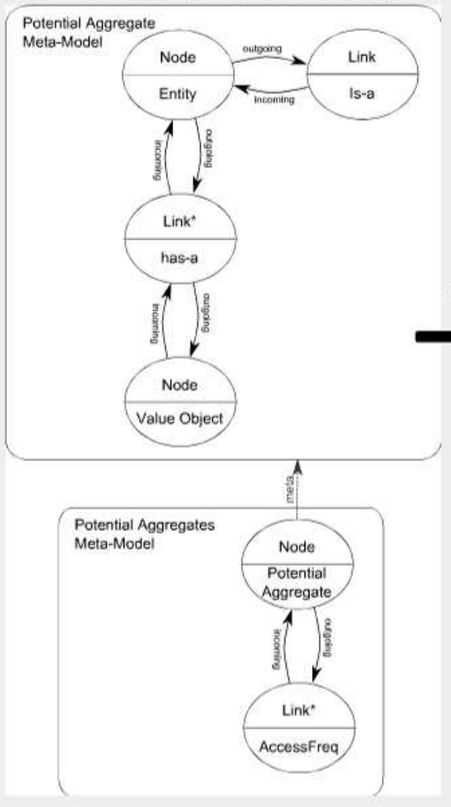




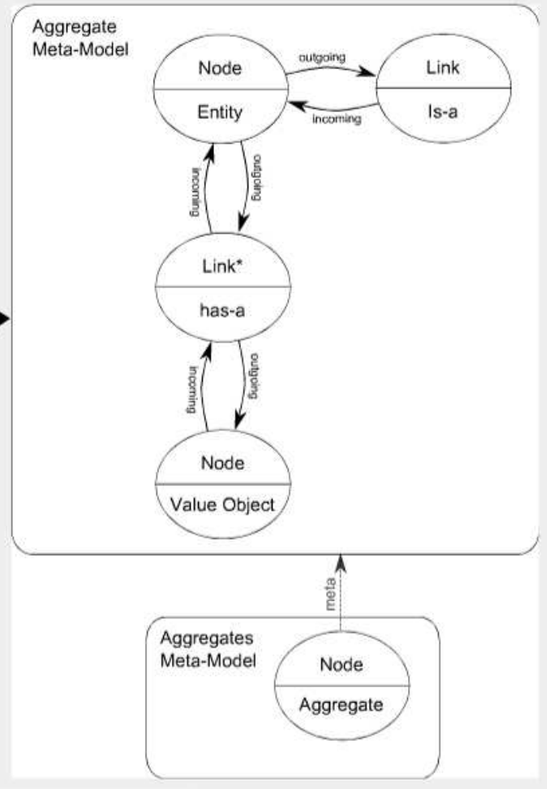
Aggregate Transformation

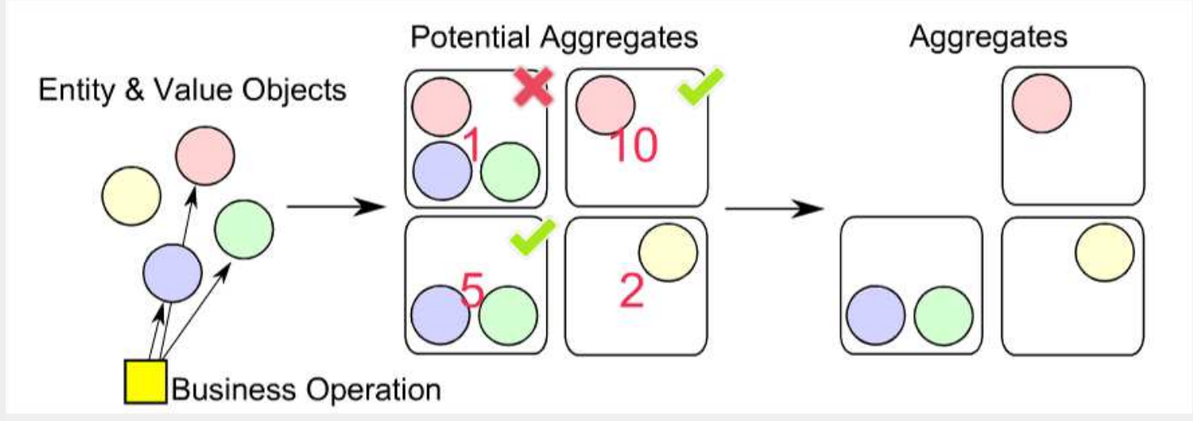


TR

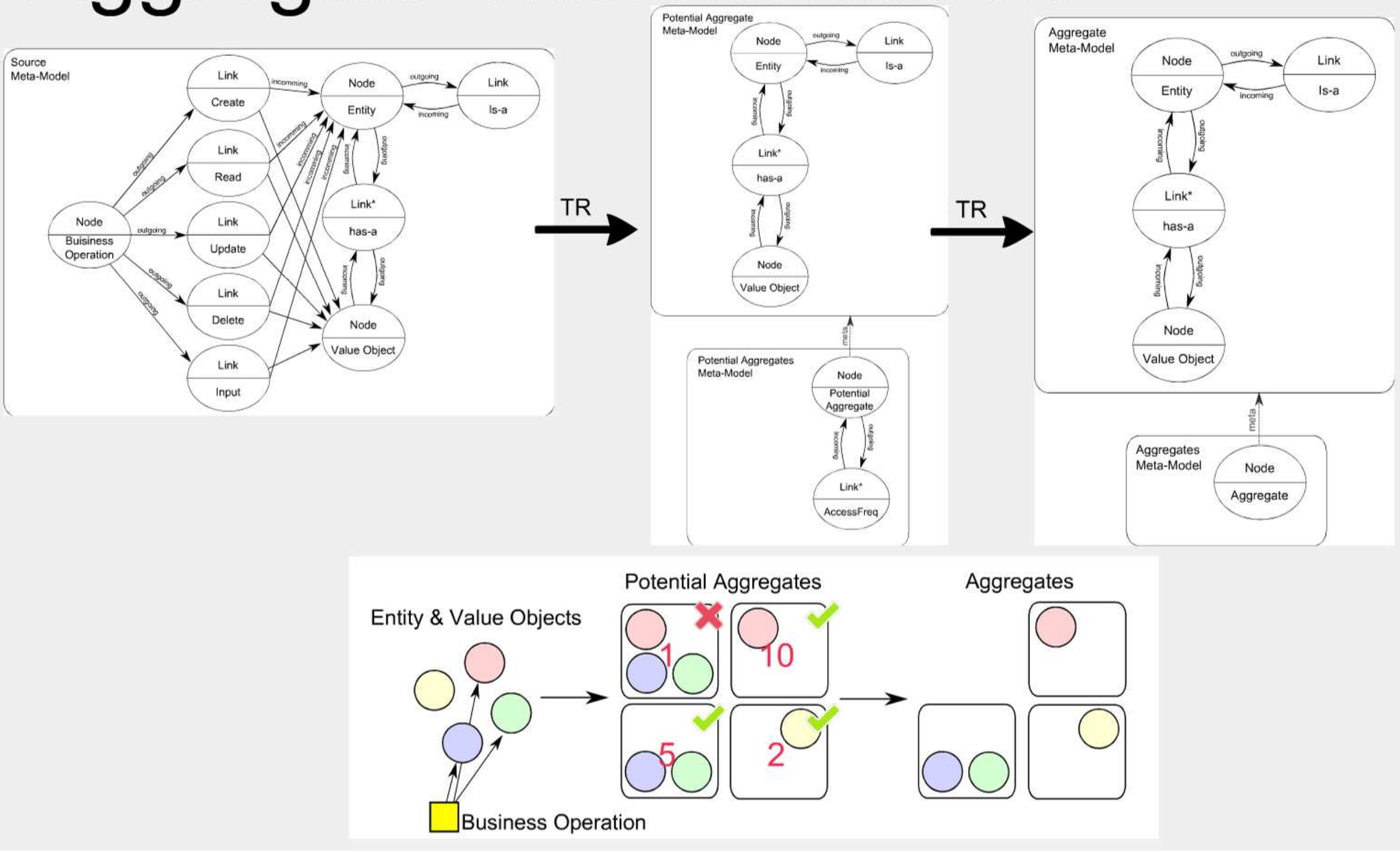


TR

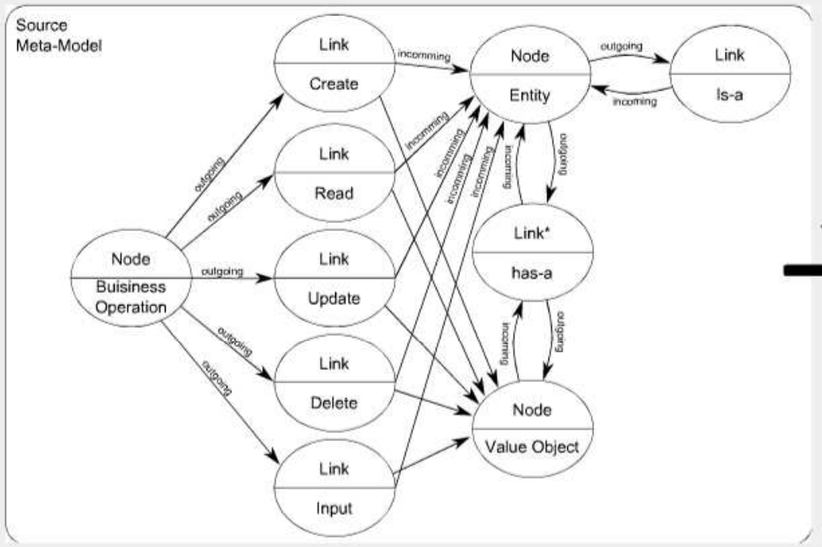




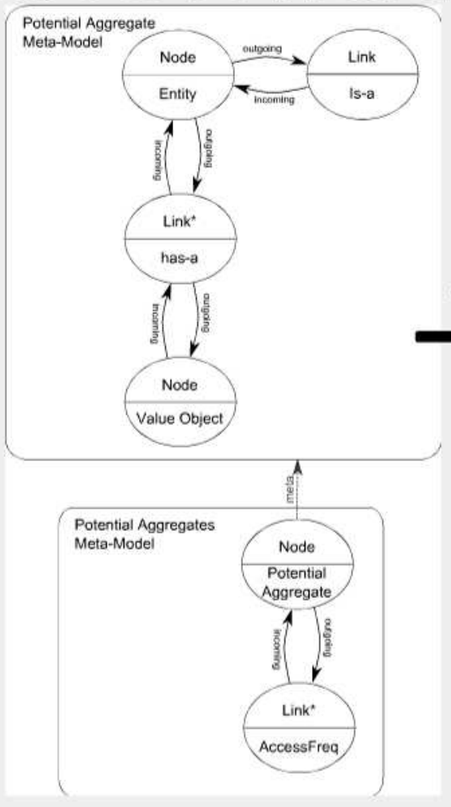
Aggregate Transformation



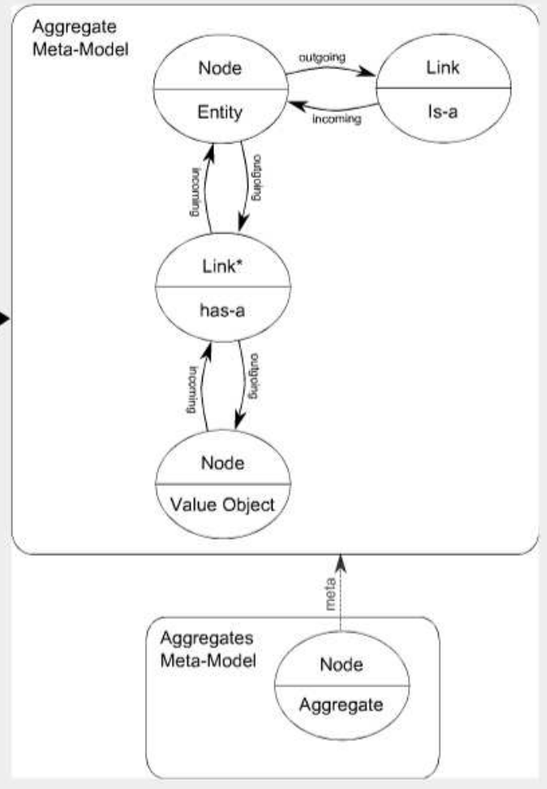
Aggregate Transformation



TR



TR



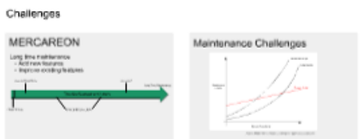
Excerpt of a Transformation Rule (TR):

$$\langle move \rangle ::= \text{Priority 1} : \text{BusinessOperation(bo)} \wedge \text{Entity(e)} \wedge (\text{Create(bo,e)} \vee \text{Read(bo,e)}$$
$$\vee \text{Update(bo,e)} \vee \text{Delete(bo,e)} \vee \text{Input(bo,e))} \wedge \text{name(e,eName)}$$
$$\rightarrow \text{Entity(newE)} \wedge \text{name(newE,eName)} \wedge \text{PotentialAggregate(p)} \wedge \text{partOf(newE, p)};$$

Agenda

Introduction

Challenges



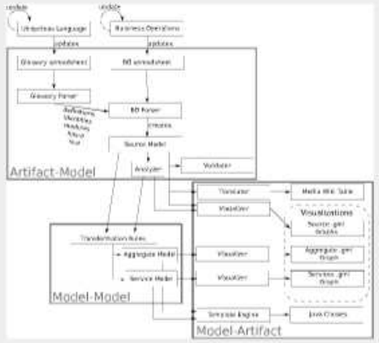
Domain-Driven Design

Refactoring

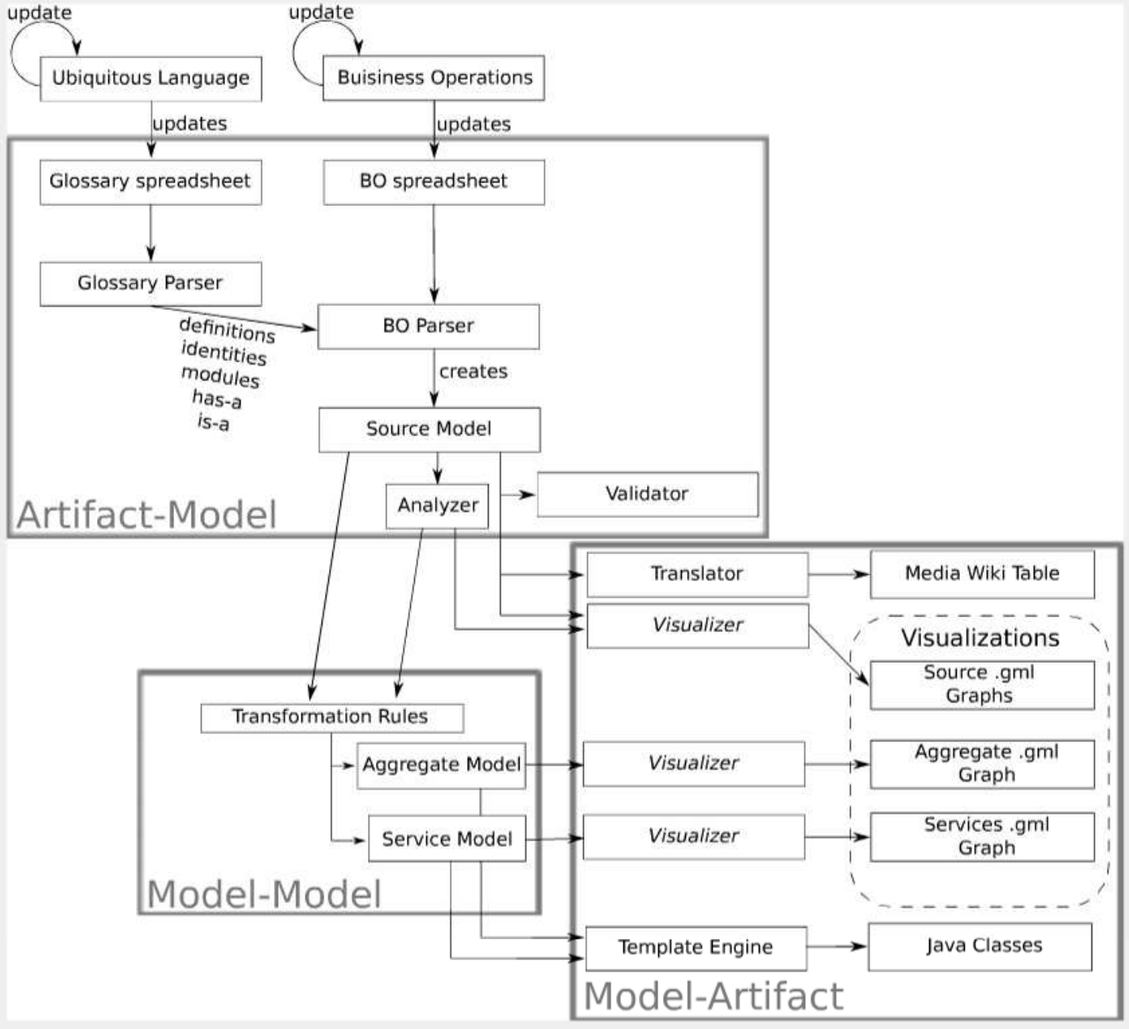
Prototype

Prototype

Architecture

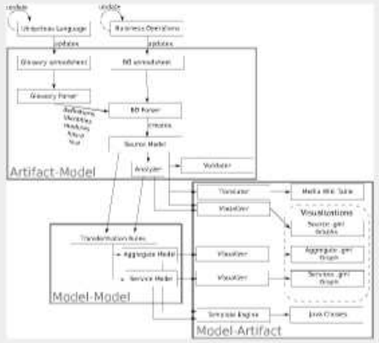


Architecture



Prototype

Architecture

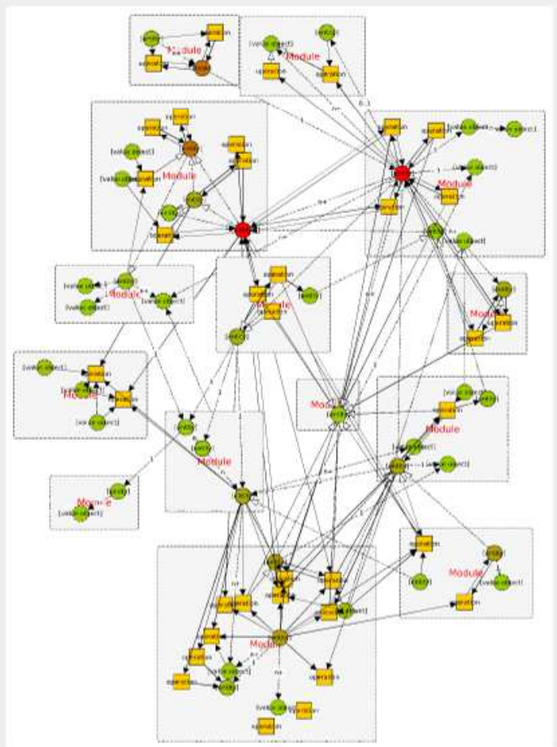


Resulting Artifacts

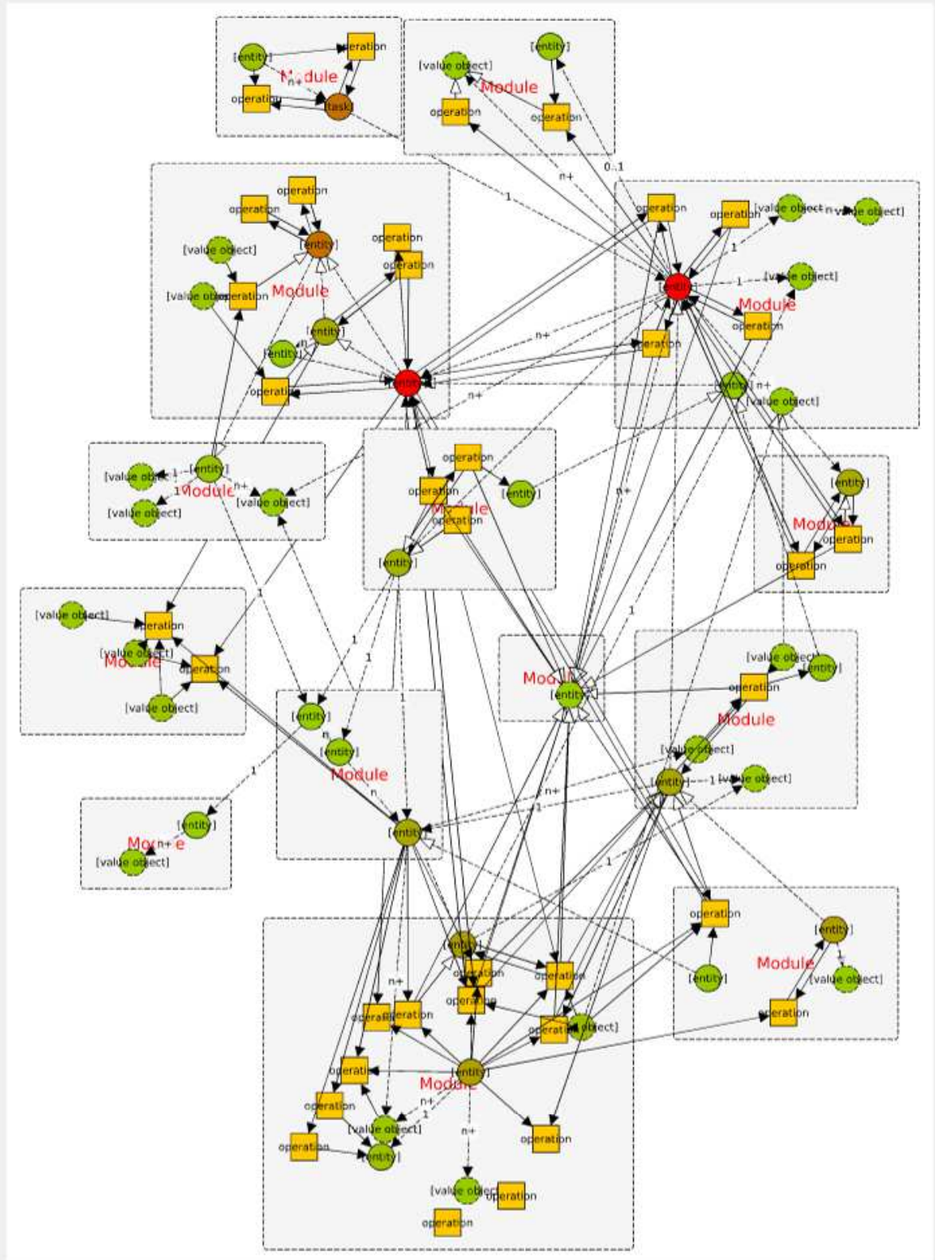
Source Model

Resulting Artifacts

Source Model

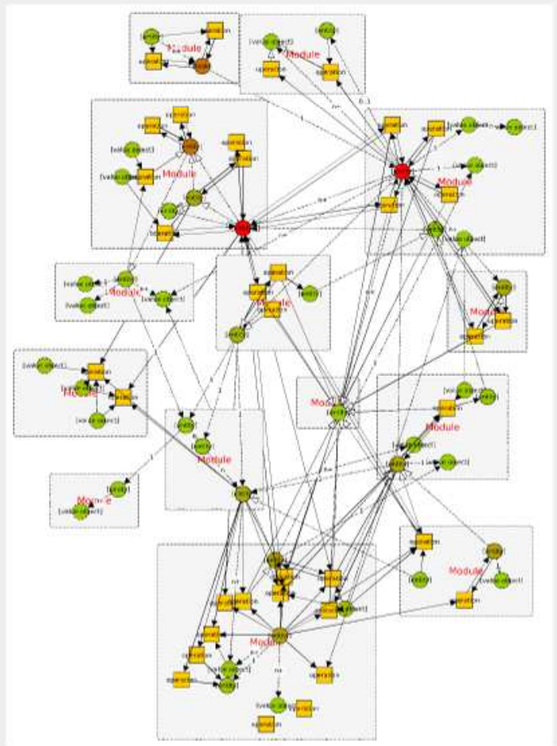


Source Model

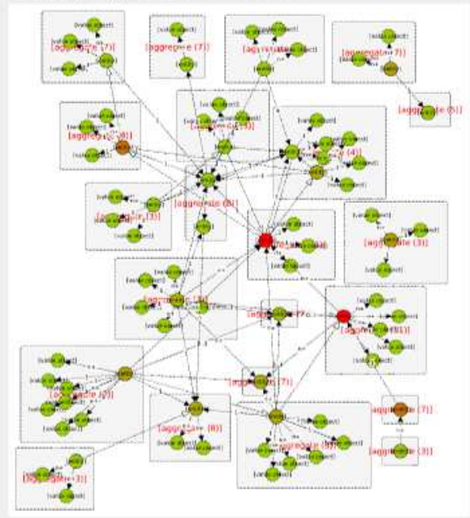


Resulting Artifacts

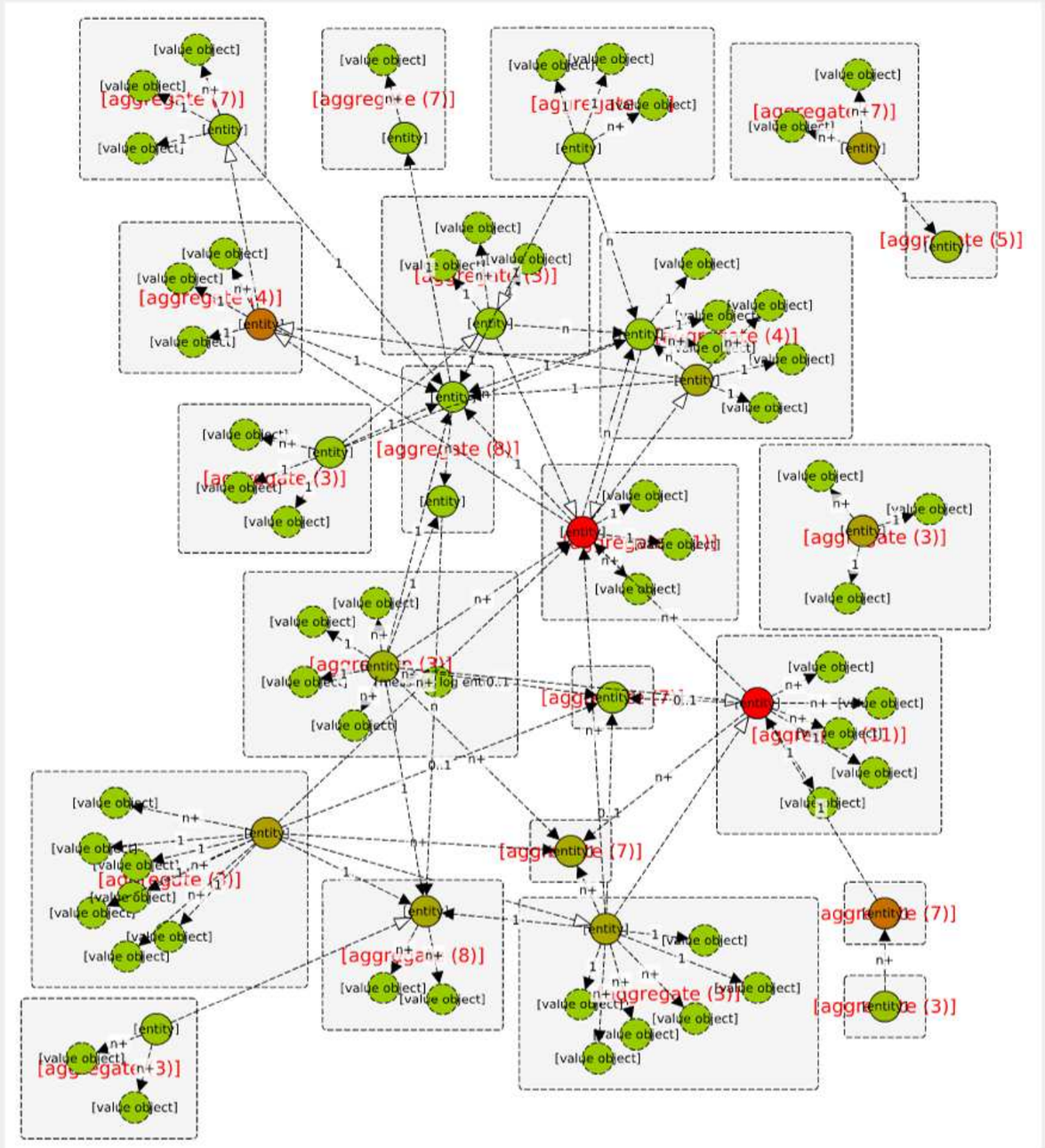
Source Model



Aggregate Model

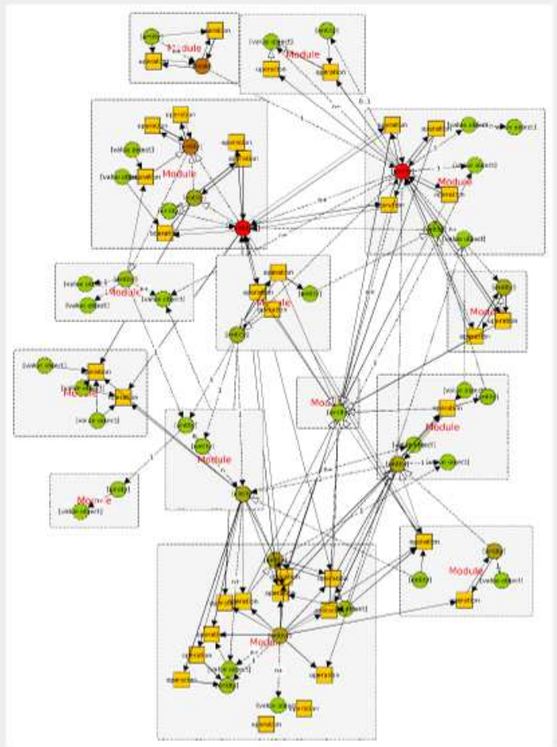


Aggregate Model

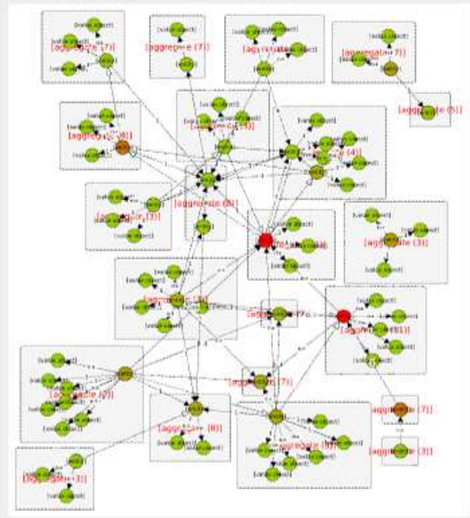


Resulting Artifacts

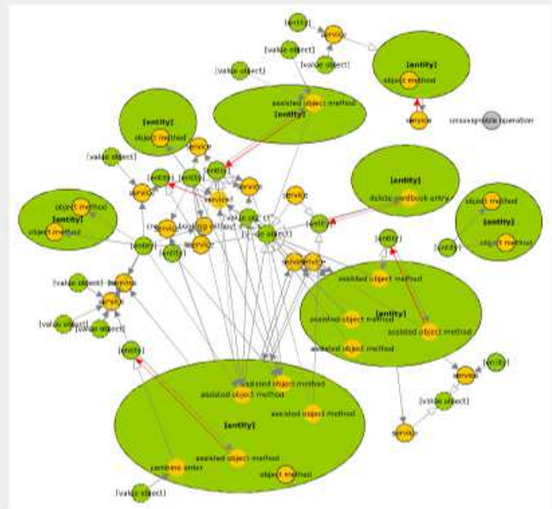
Source Model



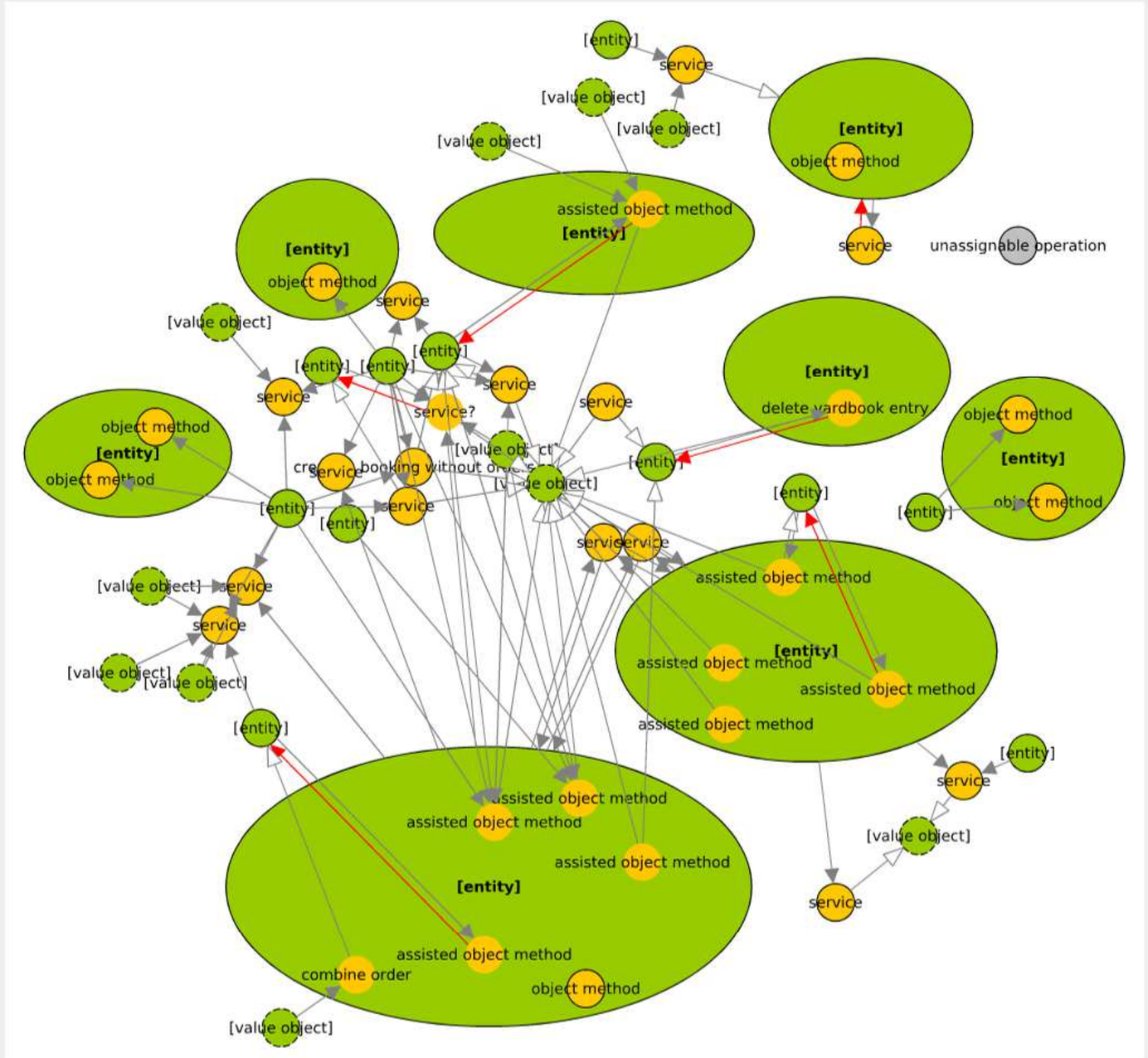
Aggregate Model



Service Model

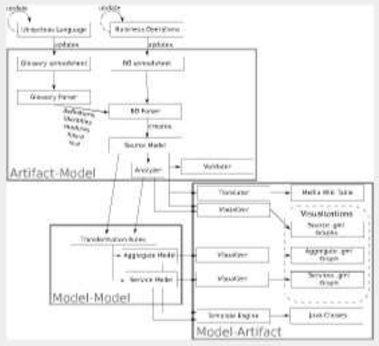


Service Model

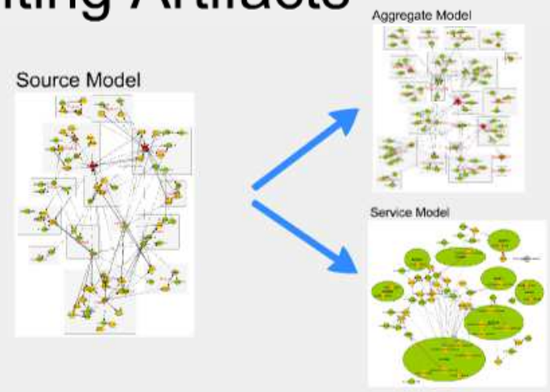


Prototype

Architecture

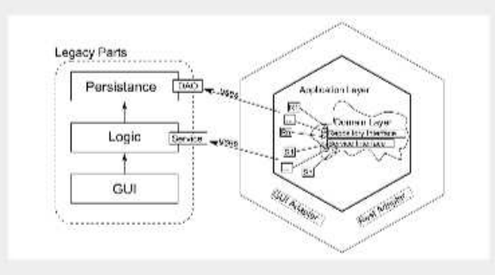


Resulting Artifacts



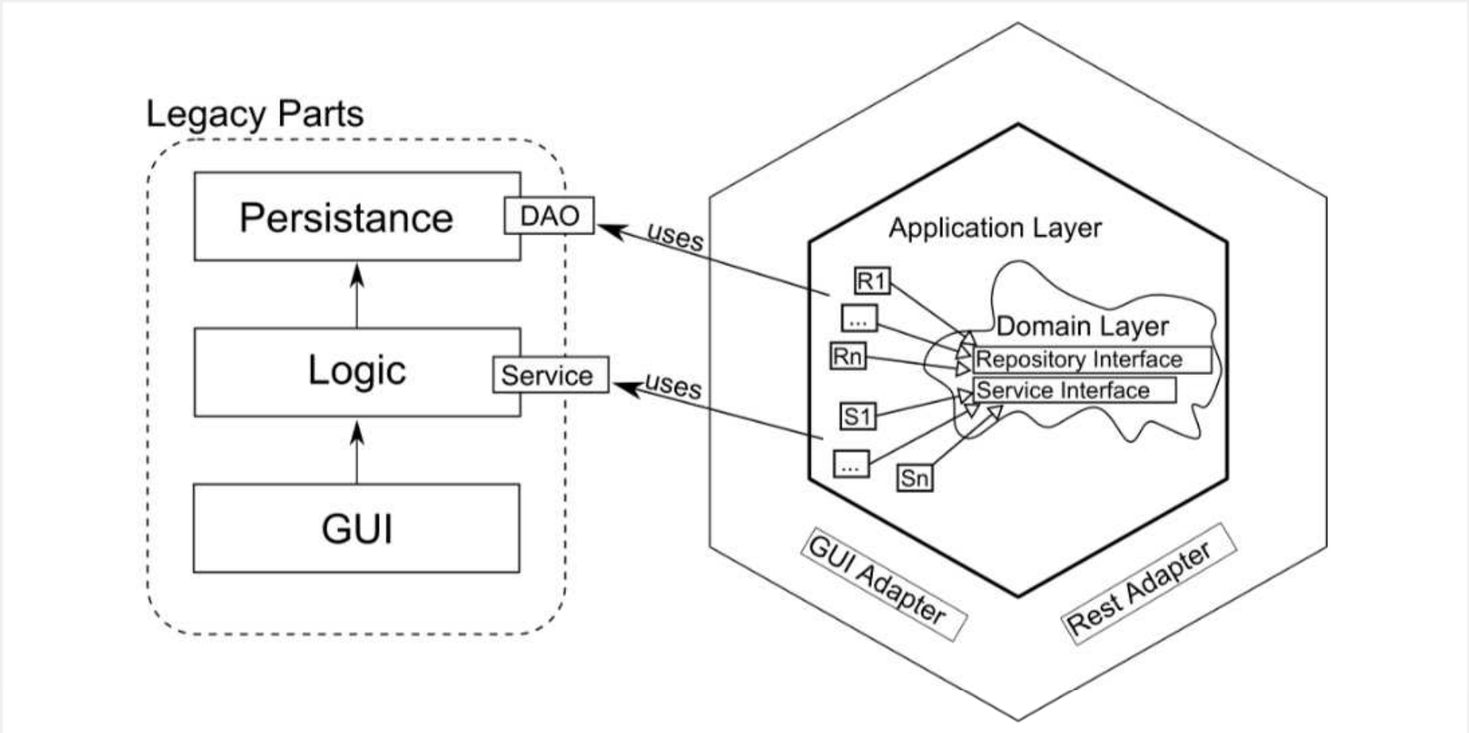
Refactoring Realization

- Bubble Context for the Liveyard View
- Communication With The Legacy System by utilizing Ports and Adapters



Refactoring Realization

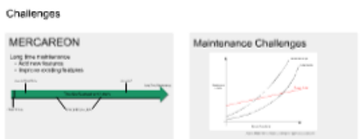
- Bubble Context for the Liveyard View
- Communication With The Legacy System by utilizing Ports and Adapters



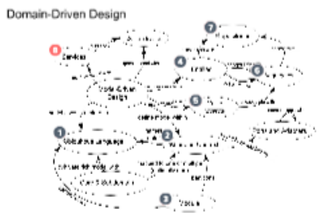
Thank You For Your Attention!

Introduction

Challenges



Domain-Driven Design



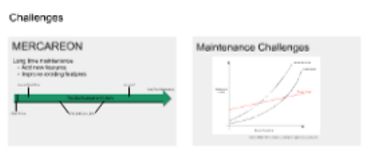
Refactoring

Prototype

Thank You For Your Attention!

Introduction

Challenges



Domain-Driven Design

Refactoring

Questions?

Prototype