Software Development

Group A Lyngby

11/10/2024

Systems Integration

OLA 3

Name	Mail
Andreas Fritzbøger	cph-af167@cphbusiness.dk
Owais Dashti	cph-od42@cphbusiness.dk

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Contents

1. Introduction

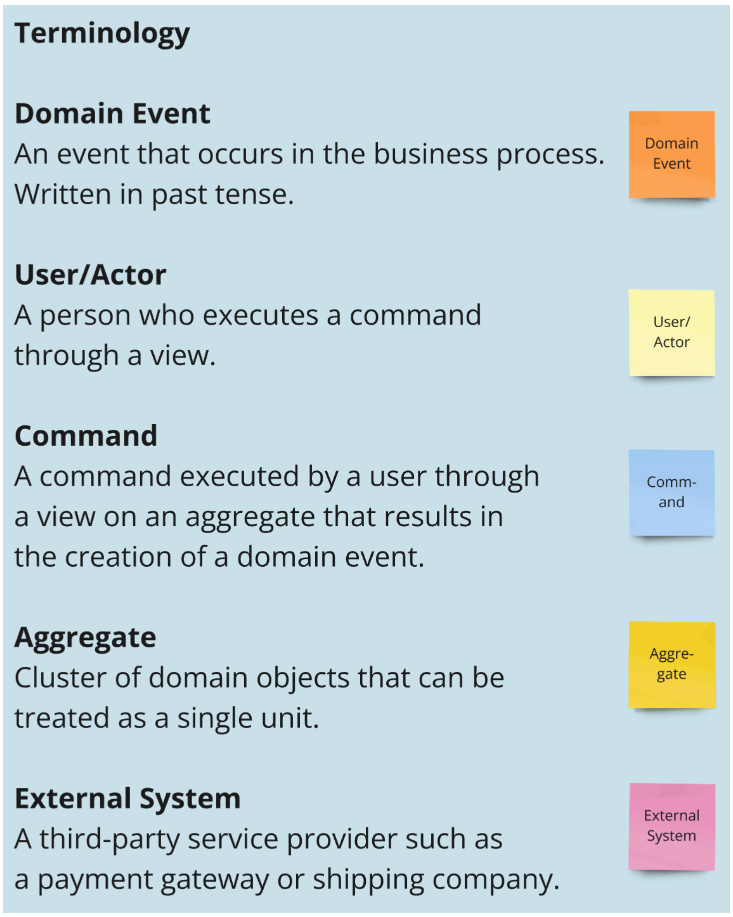
In this project, we will apply Domain-Driven Design (DDD) to model the MyTrailer system. We begin with strategic design, where we will conduct an event storming session to identify key domain events and processes, which will be detailed in a separate section. Following this, we will define the ubiquitous language, ensuring consistent terminology throughout the system. Next, we will establish the bounded contexts of the subdomains, which will form the foundation of our architecture. Afterward, we will create a context map to illustrate the relationships between these subdomains. Once the strategic design is complete, we will move on to tactical design, where we define the internal structure of the bounded contexts, including aggregates, repositories, and domain services. Finally, we will provide a short reflection on our design process and the approaches used in building the MyTrailer system.

2. Video

Video on YouTube

3. Event Storming

We've created an event storming session in which we used *Judith Birmosers* Event Storming template. These are our results (link):



Terminology

Domain Event
An event that occurs in the business process.
Written in past tense.

Domain Event

User/Actor
A person who executes a command
through a view.

User/ Actor

Command
A command executed by a user through
a view on an aggregate that results in
the creation of a domain event.

Comm- and

Aggregate
Cluster of domain objects that can be
treated as a single unit.

Aggre- gate

External System
A third-party service provider such as
a payment gateway or shipping company.

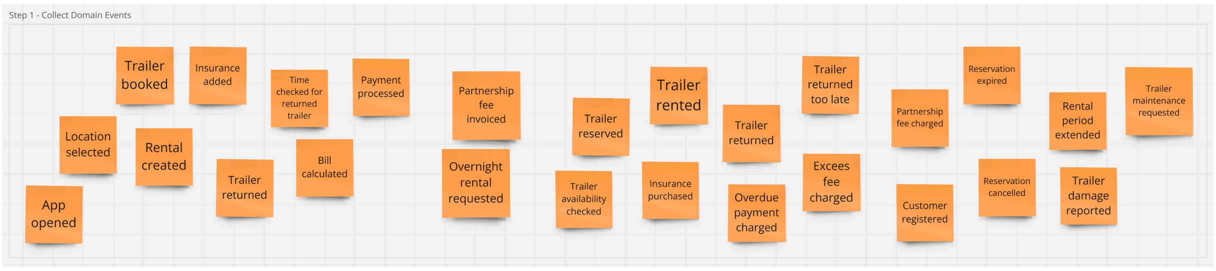
External System

Figure 1: The terminology we have used for our event storming[1]

[1]Terminology defined by Judith Birmisor: Event Storming Template

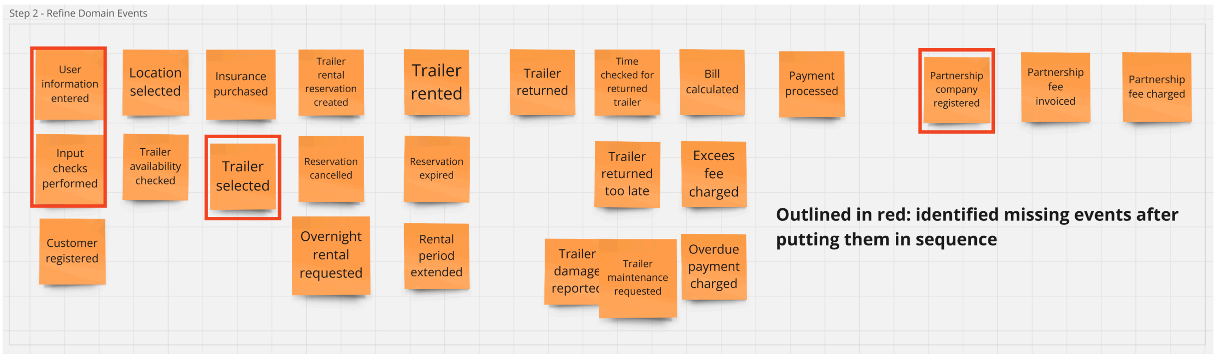
3.1. Step 1: Collecting Domain Events

In this step, orange Post-Its are used to capture all the relevant domain events in the MyTrailer system. These represent key business events that have occurred, such as *TrailerBooked*, *TrailerReturned*, and *InsurancePurchased*. Each event is later placed in chronological order, laying the groundwork for understanding the flow of events through the rental process.



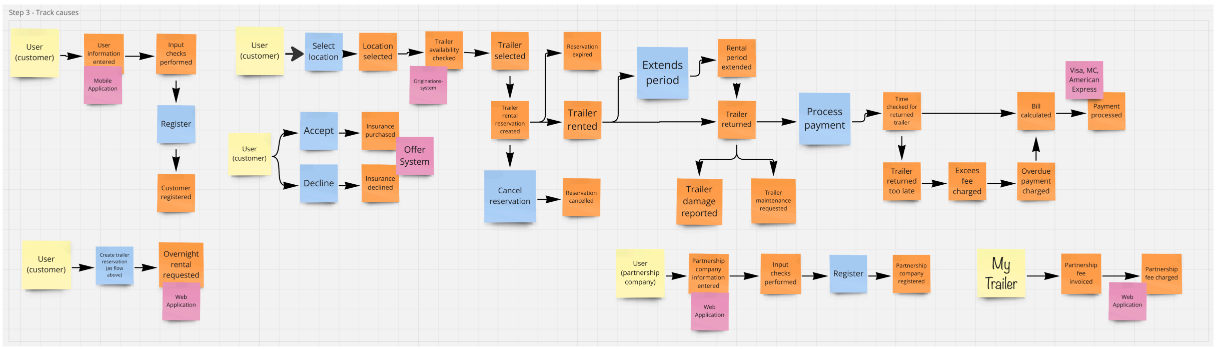
3.2. Step 2: Refining Domain Events

During this step, the domain events from step 1 are refined. We reviewed the events in the team to ensure that there are no duplicates, syntactically correct, meaningful, and in the correct order. This step ensures that all events are clearly defined and properly aligned with the business process.



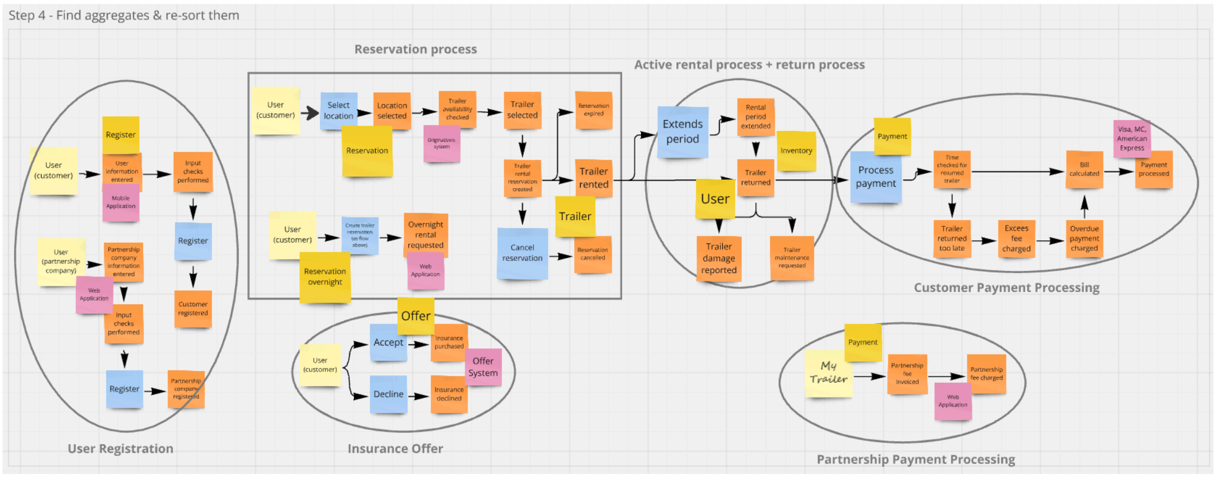
3.3. Step 3: Tracking Causes

In step 3, the focus shifts to understanding the cases behind each domain event. Events are categorized based on their triggers, whether they are caused by user actions, external systems, or other domain events. This step helps map out the relationships between events and their triggers.



3.4. Step 4: Finding aggregates and re-sorting them

The final step involves grouping the domain events around the aggregates they impact. Aggregates such as *Trailer*, *Offer*, *Payment* are identified, and events are clustered according to their relevance. For example, *Trailer rental reservation* created affects the *Reservation* and *Trailer* aggregates, while the *TrailerReturned* impacts the *Inventory* and *User* aggregates.

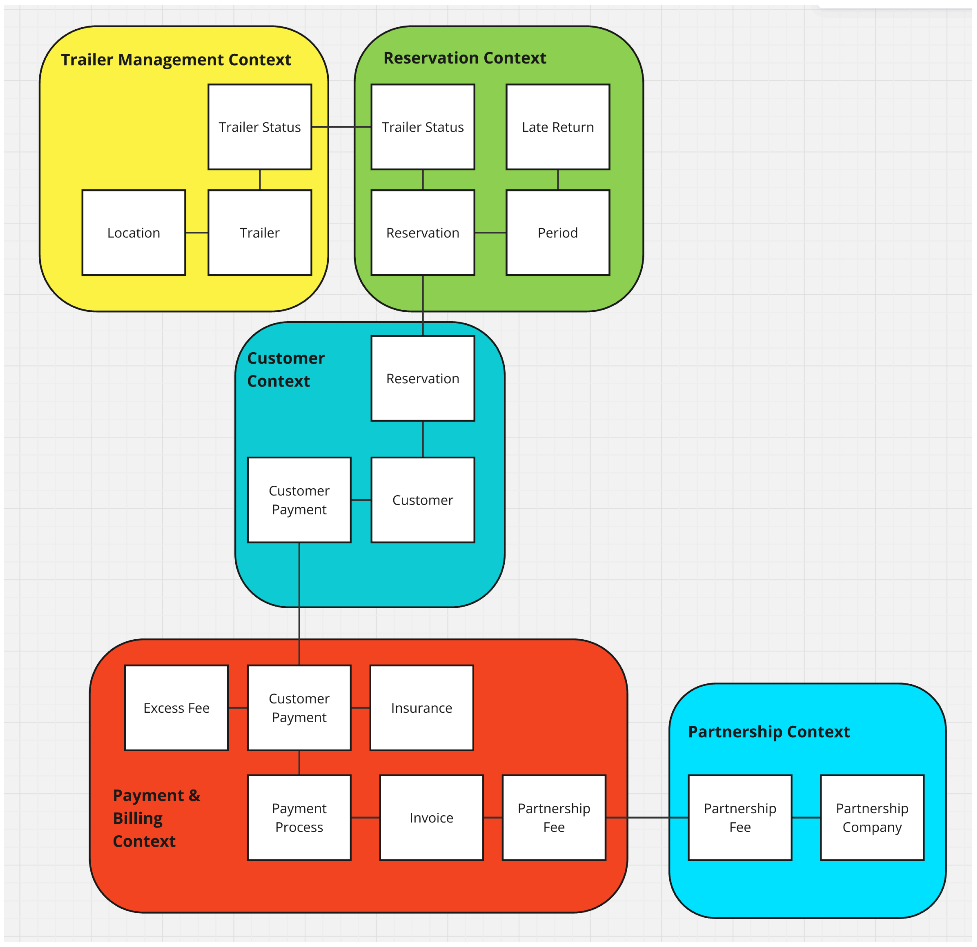


4. Ubiquitous Language

- **Trailer:** A vehicle available for short-term or long-term rentals, identified by a location and trailer number.
- **Reservation:** A booking made by a customer to rent a specific trailer. Short-term rentals are booked via the app, while long-term rentals require booking through the website.
- **Rental:** When a customer starts the actual borrowing of the trailer.
- **Insurance:** An optional service for customers to protect against damage, costing 50 kr.
- **Excess fee:** A charge applied when a trailer is returned after the rental period, typically calculated based on the duration of the delay.
- **Partner location:** Companies hosting trailers at their premises (e.g Jem og Fix, Silvan and more..), where the trailers display both the MyTrailer and the partners branding.
- **Partnership fee**: A payment made by partners to MyTrailer for providing trailer services, helping attract customers to their stores.
- **Customer**: A registered user who can book a trailer from MyTrailer via the app or website.
- **Short-term rental**: Rentals that are limited to 24 hours, but always finish at midnight at the latest. This means a trailer rented at 9pm could only be rented for a maximum of three hours.
- **Long-term rental:** Rentals that are available for overnight use. These need to be booked from the website.
- **Location ID**: A unique identifier for each trailer, based on its location and trailer number.
- **Late return**: Occurs when a trailer is returned after the end of the rental period, resulting in an excess fee.
- **Trailer status**: Represents the current state of a trailer (e.g., available, booked, in-use, returned), determining its readiness for rental.
- **Notification**: Alerts sent to customers or system administrators, triggered by events like overdue rentals or completed bookings.

5. Bounded Contexts

The following are identified bounded contexts:



5.1. Trailer Management Context

Manages the details and status of trailers, such as availability, location, and condition. Its responsibilities include:

- Tracking trailer status (available, booked, in-use, returned).
- Associating trailers with specific locations and IDs.
- Updating trailer status after reservations or returns.

Key terms:

- Trailer, Location ID, trailer status

5.2. Reservation Context

Handles trailer reservations and ensures trailers are available for rental when needed. Its responsibilities include:

- Managing short-term and long-term trailer reservations.
- Validating the availability of trailers based on location and trailer ID.
- Enforcing rental periods (e.g. ensuring trailers are returned before midnight for short-term rental).

Key terms:

- Reservation, short-term rental, long-term rental, late return

5.3. Payment & Billing Context

Manages payments for insurance, excess fees, and partnership fees. Its responsibilities include:
- Charging customers for insurance and any excess fees due to late returns.
- Tracking payments made by partner locations for using the trailer services.
- Sending invoices and managing payment notifications.

Key terms:
- Insurance, Excess Fee, Partnership Fee, Payment

5.4. Partner Management Context

Handles relationships with partner locations, including partnership agreements and payments. Its responsibilities include:
- Tracking partner locations and trailers hosted by them
- Managing partnership agreements and payment processing.
- Ensuring proper branding on trailers as per partnership agreements.

Key terms:
- Partner Location, Partnership Fee

5.5. Customer Context

Manages customer registration, profile information, and interactions with the system. Its responsibilities include:
- Handling customer registration and account management.
- Managing customer booking and payment history.
- Sending notifications regarding rentals (e.g. booking confirmation, overdue trailer notifications).

Key terms:
- Customer, Notification

6. Context Map

- **Trailer Management** context interacts with **Reservation context** to ensure trailers are available or reserved.
  - **Reservation** queries **Trailer Management** to verify the availability of a trailer when a reservation is initiated.
- **Reservation context** interacts with **Payment & Billing context** to charge customers for additional services like insurance and excess fees.
  - Once a reservation is made, **Reservation** triggers a payment request in the **Payment & Billing context** for charging customers. If there is a late return **Reservation** notifies **Payment & Billing** to apply excess fees.
- **Partner Management context** interacts with **Payment & Billing context** to handle payments from partner locations.
  - Partner Management periodically triggers partnership fee requests to the **Payment & Billing context** for processing payments from partners.
- **Customer context** interacts with **Reservation context** and **Payment & Billing context** to manage their bookings and payments.

- Customer initiates a reservation through Reservation, which in turn triggers payment processing in **Payment & Billing**. The **Customer** is also notified of the payment status or any overdue charges.

7. Tactical Design

For the tactical design, we will define the key building blocks such as **Domain Objects** (**Entities** and **Value Objects**), **Aggregates**, **Repositories** and **Services**. The design will follow each bounded context from earlier.

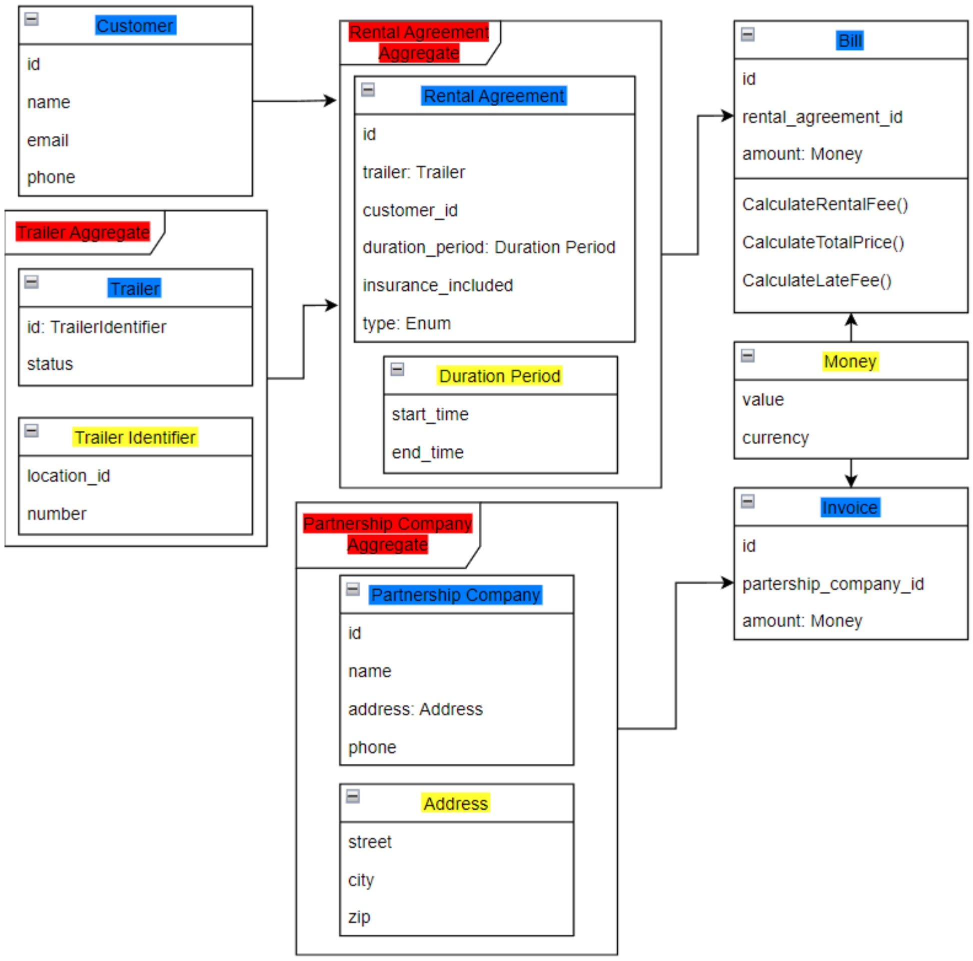


Figure 3: Our entities(blue), value objects(yellow), and aggregates(red), which encapsulates entities and value objects if appropriate.

7.1. Entities

Entities correspond to real-world objects. They are mutable and represent key concepts in the domain. They have a unique identifier.

For our domain, the entities are:

- Customer
- Trailer
- Rental Agreement
- Bill
- Invoice
- Partnership Company

Each and everyone of them having a unique identifier.

7.2. Value Objects

Value objects represent values in our domain. They are immutable–once created, they cannot be changed. If we want to a value to be something else, we need to create a new one in its place. Our value objects are:

- Trailer Identifier, used as Trailer entities' identifier. The case study said, MyTrailer's trailers were identified by a location and a number.
- Duration Period, used in Rental Agreement entities, when determining the start and end of a period for renting the booked trailer.
- Money, used in both Bill and Invoice entities, since they both are entities, which revolve around a payment process for an amount.
- Address, only used for Partnership Company. It could also have been used in Customer. However, we have interpreted the rental process as being the customer picks up the trailer, and MyTrailer does not deliver the booked trailers.

7.3. Aggregates

An aggregate is a cluster of domain objects that can be treated as a single unit.
We have entities and value objects encapsulated in aggregates, if we deemed the domains as a single unit.
The aggregates we have are:

- Rental Agreement Aggregate, which consist of Rental Agreement and Duration Period.
- Trailer Aggregate, which consist of Trailer and Trailer Identifier.
- Partnership Company Aggregate, which consists of Partnership Company and Address.

Each aggregate has one of its component objects be the aggregate root, from which any references from outside the aggregate should only go to.
The aggregates integrity is better ensured.

7.4. Repositories

Trailer Repository

Responsible for data access and management of trailer entities.

- **Methods**
  - `findById(trailer_id)`: Retrieves a trailer by its unique identifier.
  - `save(trailer)`: Persists a trailer entity.
  - `findAll()`: Retrieves all trailers.

Reservation Repository

Handles data access and management of reservation entities.

- **Methods**
  - `findById(reservation_id)`: Retrieves a reservation by its unique identifier.
  - `save(reservation)`: Persists a reservation entity.
  - `findByCustomerId(customer_id)`: Retrieves all reservations for a specific customer.

- **Customer Repository**

Manages data access for customer entities.

- **Methods**
  - `findById(customer_id)`: Retrieves a customer by their unique identifier.
  - `save(customer)`: Persists a customer entity.
  - `findAll()`: Retrieves all customers.

- **Partner Company Repository**

Handles data access for partner company entities.

- **Methods**
  - `findById(partner_id)`: Retrieves a partner company by its unique identifier.
  - `save(partnerCompany)`: Persists a partner company entity.

7.5. Services

Reservation Service

Manages the business logic related to creating and managing reservations.

- **Methods**
  - `createReservation(customerId, trailerId, duration)`: Creates a new reservation.
  - `cancelReservation(reservationId)`: Cancels an existing reservation.
  - `calculateExcessFee(reservation)`: Calculates any fees for late returns.

Payment Service

Handles all payment-related operations.

- **Methods**
  - `processPayment(reservationId, paymentMethod)`: Processes payment for a reservation.
  - `calculateInsuranceFee()`: Calculates the insurance fee for the reservation.

8. Reflections

Definition of DONE

Our definition of done includes the completion of the following tasks:

- Conducting an event storming session to identify domain events and interactions.
- Defining a ubiquitous language that reflects the core concepts and terms of the domain.
- Establishing bounded contexts to clarify the relationships and boundaries between subdomains.
- Creating tactical design elements, including entities, value objects, aggregates, repositories, and services.
- Note: What might be missing is documenting the requirements through user stories and acceptance criteria. Other than that, we've reached a point in our planning phase in the software development life cycle that we can say "done" for.

By defining these stages, we ensure that our model is well-structured and ready for implementation in the software development process.