Book a meal

Use Case "Book a meal"

- General flow
  1. Check if the account has enough balance
  2. Register the booking
  3. Register the debit movement on the account
  4. Update the account's balance

Questions

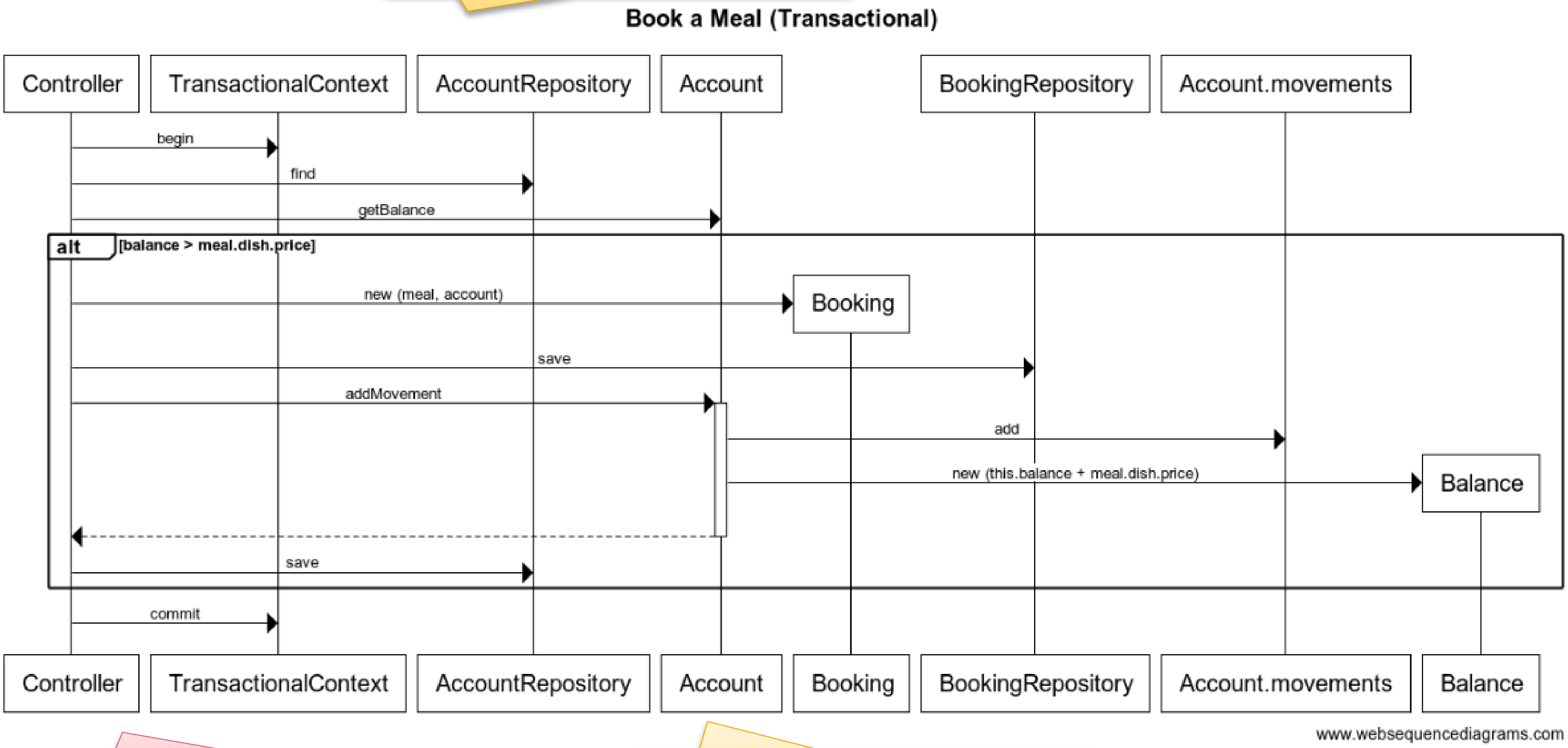
- To whom should we assign this responsibility?
  - Check if the account has enough balance
  - Update balance
- Should the use case change more than one aggregate root?

Transactional Approach

- Aggregate scenario (A)
  - Account has balance and list of movements
- Use case updates
  - Account
  - Booking

Transactional approach (A)

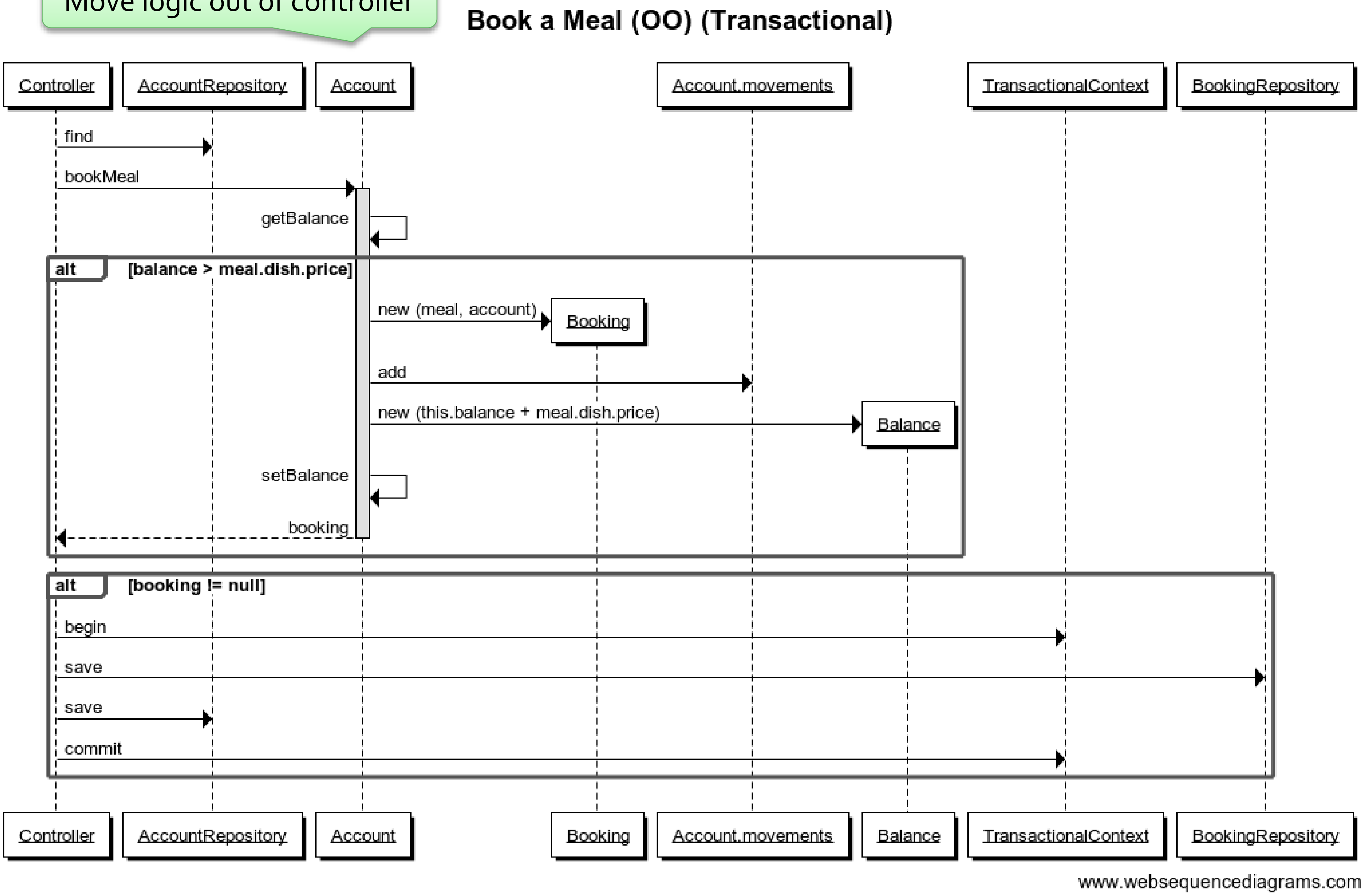
Saving the account saves movements and balance



Too much logic on controller

Account is responsible for updating the balance

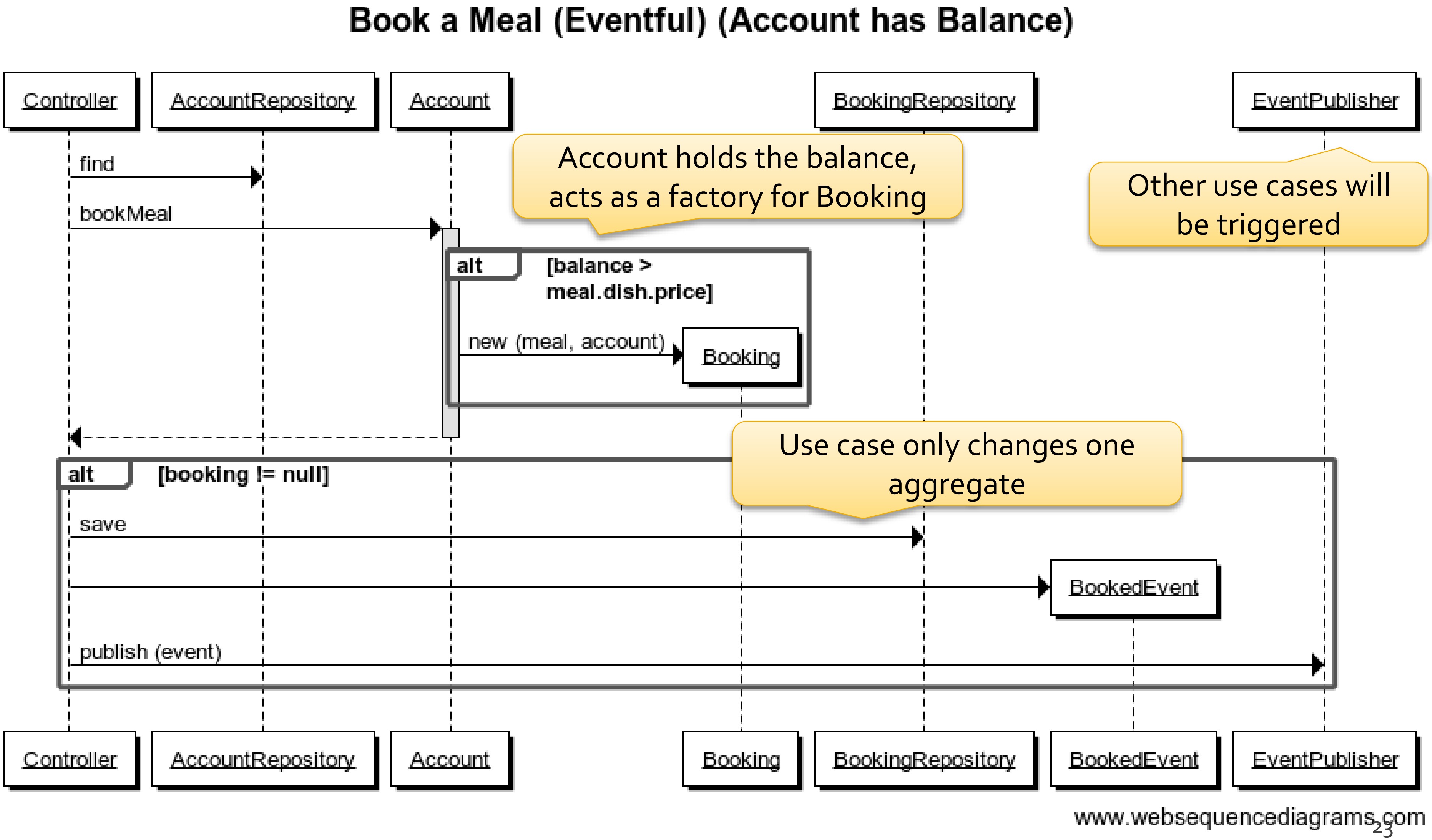
Move logic out of controller



Eventful approach

- Aggregate scenario (B)
  - Account (has balance)
  - Movement
- Use case updates only one aggregate

Eventful approach (B):
1. Book the meal

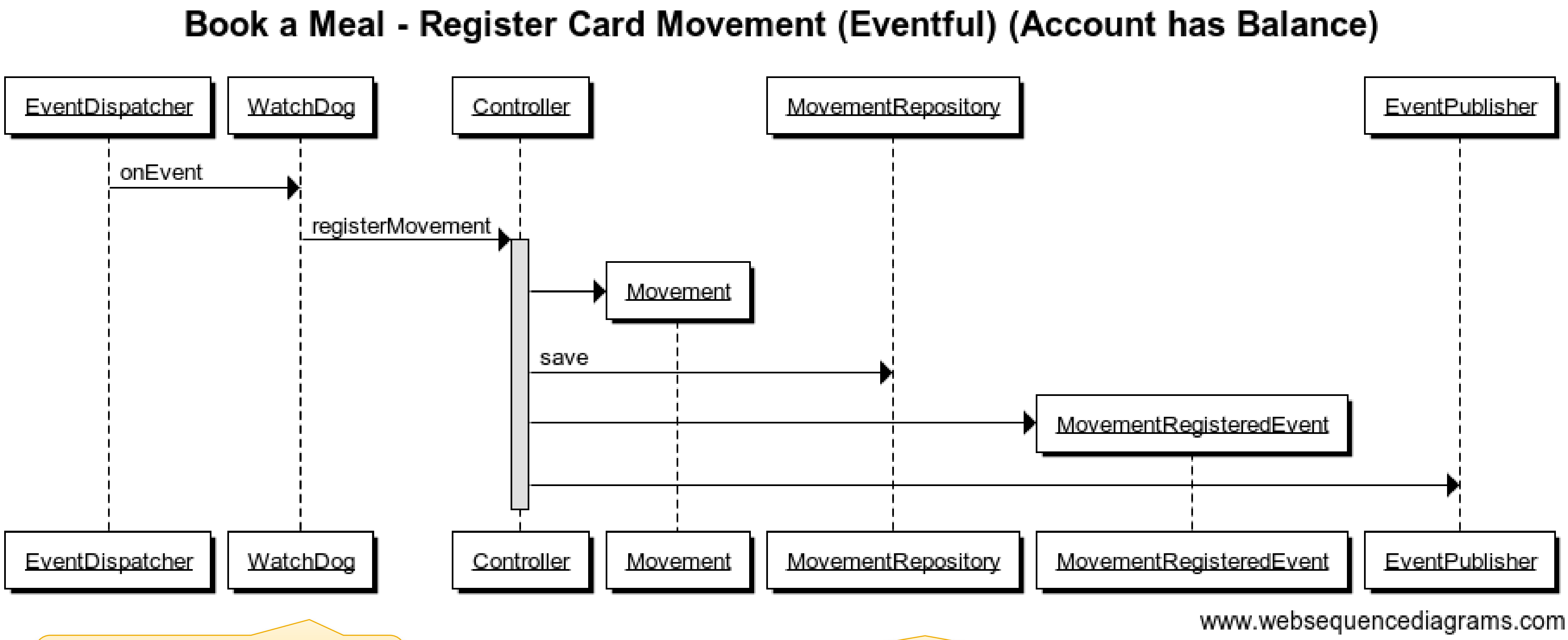


Account holds the balance, acts as a factory for Booking

Other use cases will be triggered

Use case only changes one aggregate

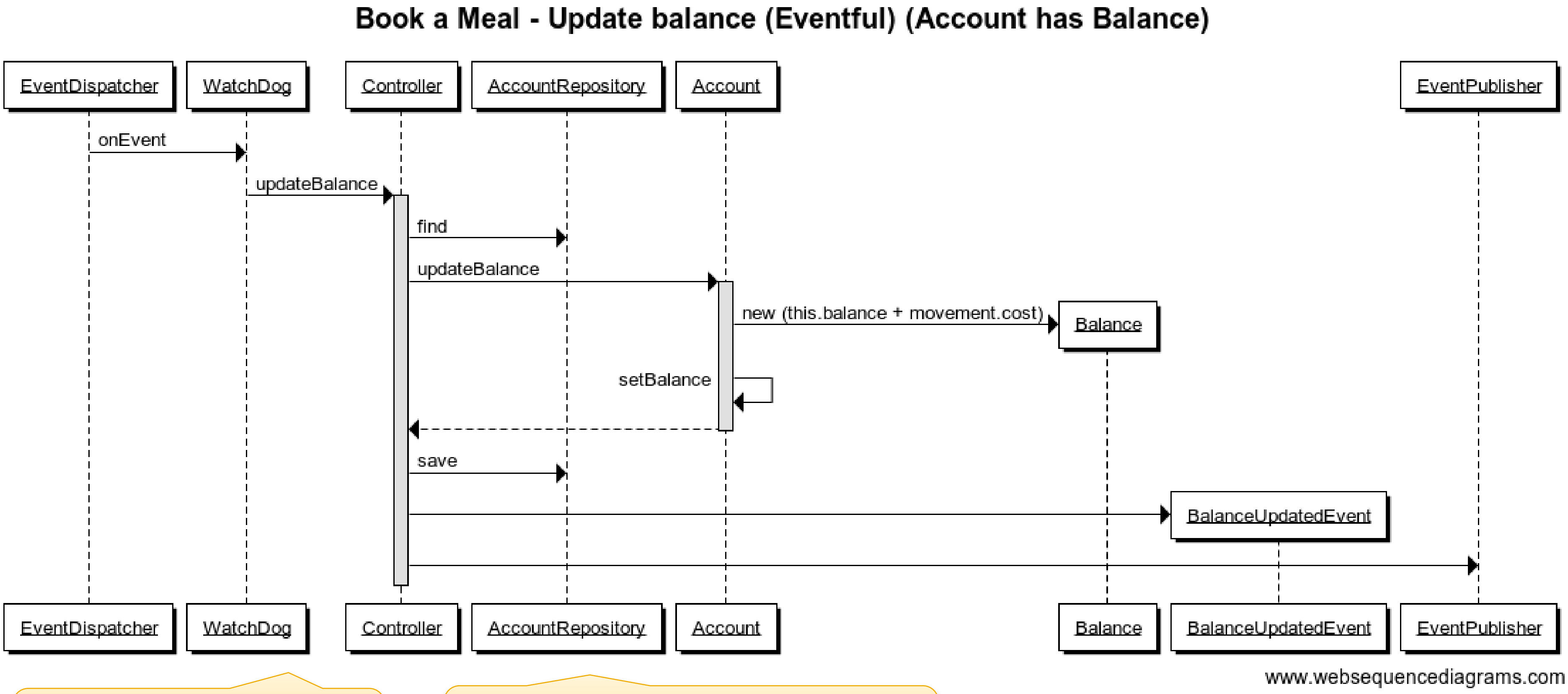
Eventful approach (B): 2. Register card movement



Dedicated watch dog

Use case only changes one aggregate

Eventful approach (B): 3. Update balance



Dedicated watch dog

Use case only changes one aggregate

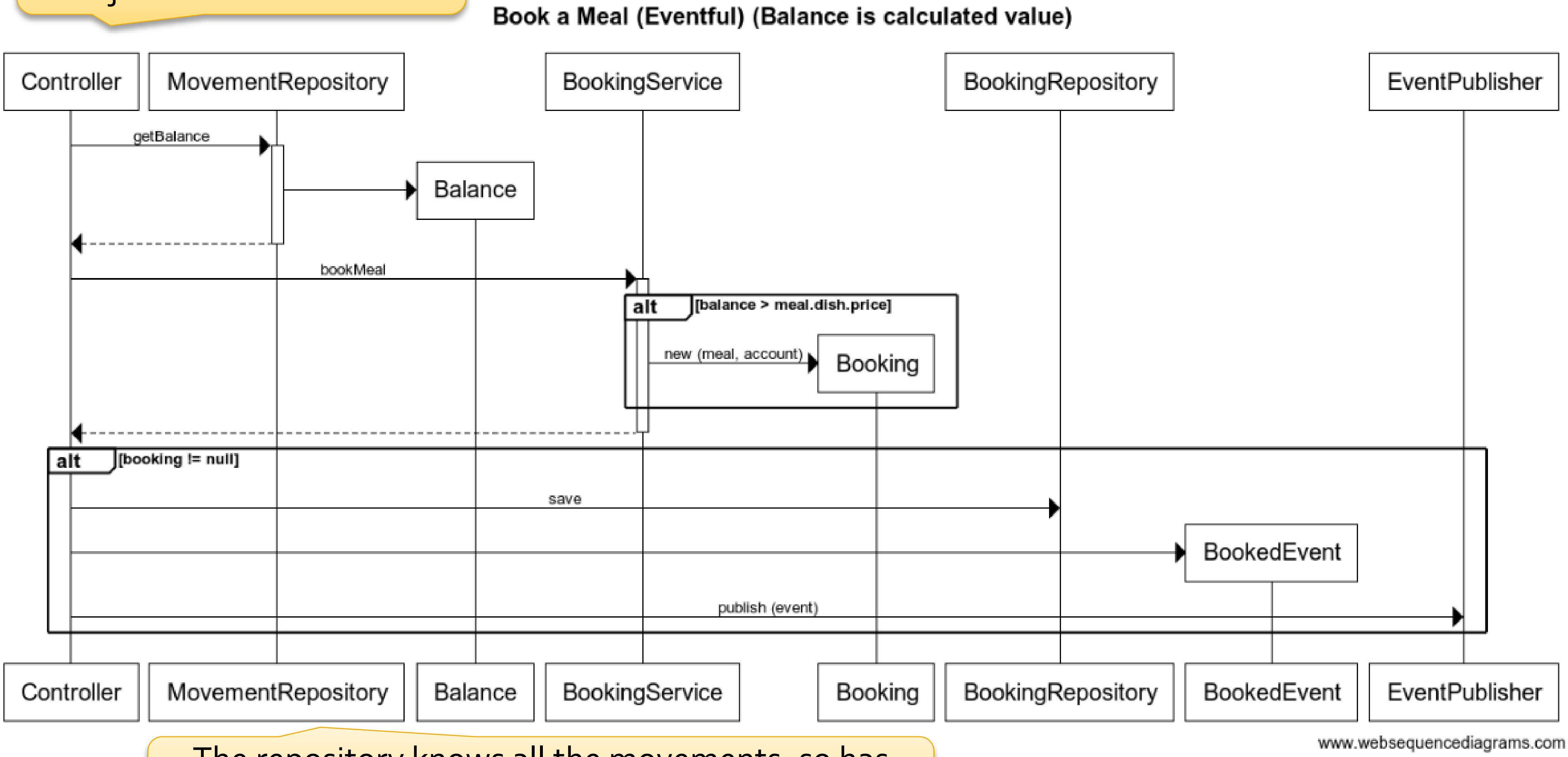
Eventful Approach

- Aggregate scenario (C)
  - Account
  - Movement
  - Balance is calculated (not persistent)
- Use case updates only one aggregate

Eventful approach (C): 1. Book the meal

Controller fetches objects to inject in domain services

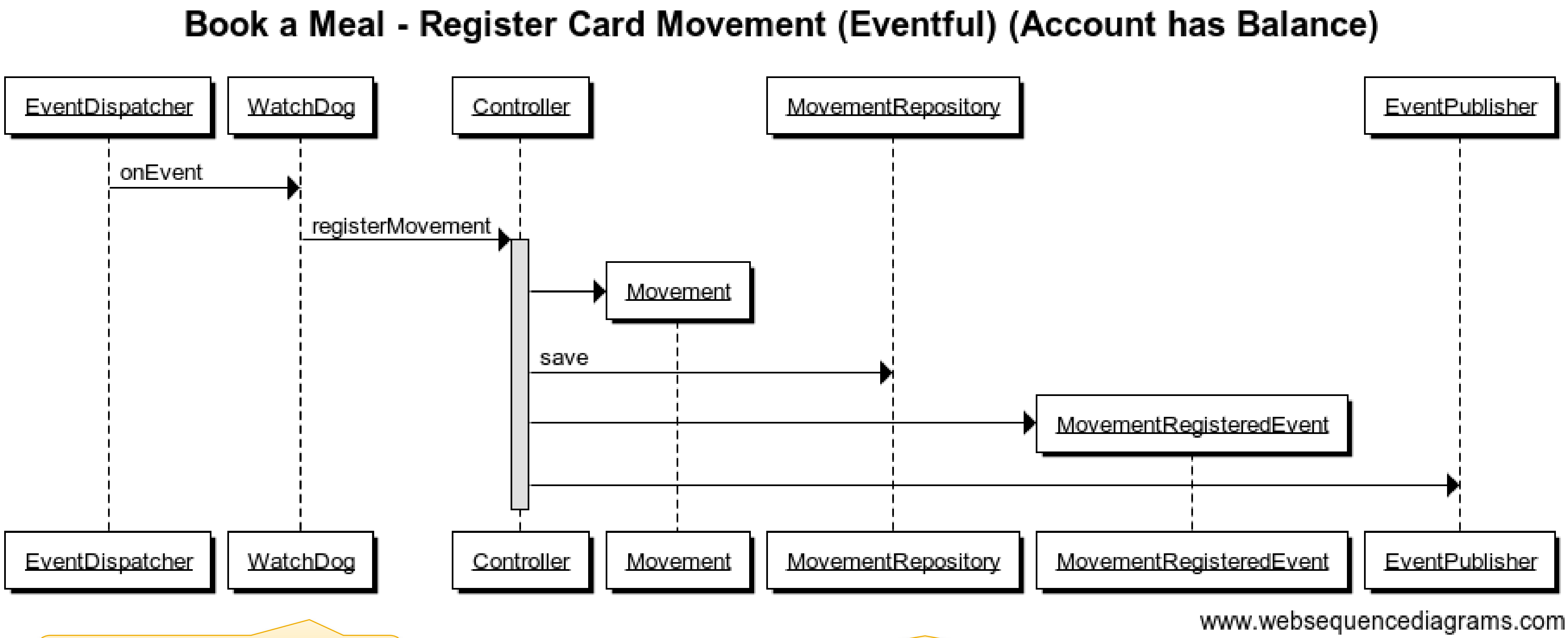
Domain service with business logic



The repository knows all the movements, so has the responsibility to calculate the balance

Eventful approach (C): 2. Register card movement

Similar to scenario (B)

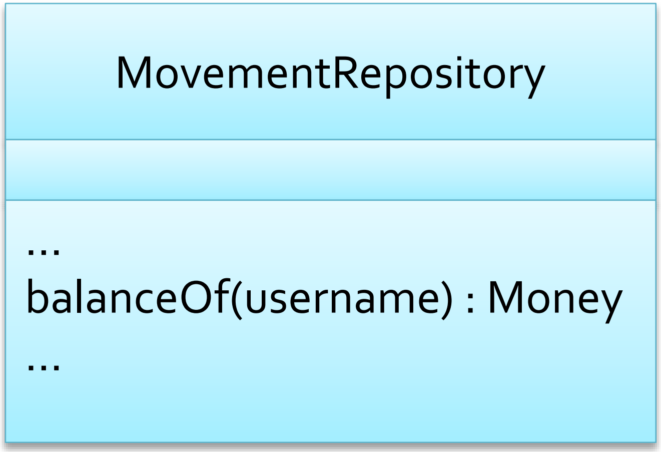


Dedicated watch dog

Use case only changes one aggregate

Eventful approach (C): 3. Update balance

- Nothing to do



MovementRepository

…
balanceOf(username) : Money
…