Interaction Diagrams II

Lecture # 5

Department of Computer Science and Technology
University of Bedfordshire

Written by David Goodwin,
based on the lectures of Dayou Li and
on the book *Applying UML and Patterns (3rd ed.)*
by *C. Larman* (2005).

Modelling and Simulation, 2012

Outline

Interaction Diagrams

Traceability

1. Requirements traceability refers to the ability to define, capture and follow the traces left by requirements on other elements of the software development environment and the trace left by those elements on requirements.
2. In the requirements engineering field, traceability is about understanding how high-level requirements - objectives, goals, aims, aspirations, expectations, needs - are transformed into low-level requirements. It is therefore primarily concerned with the relationships between layers of information.

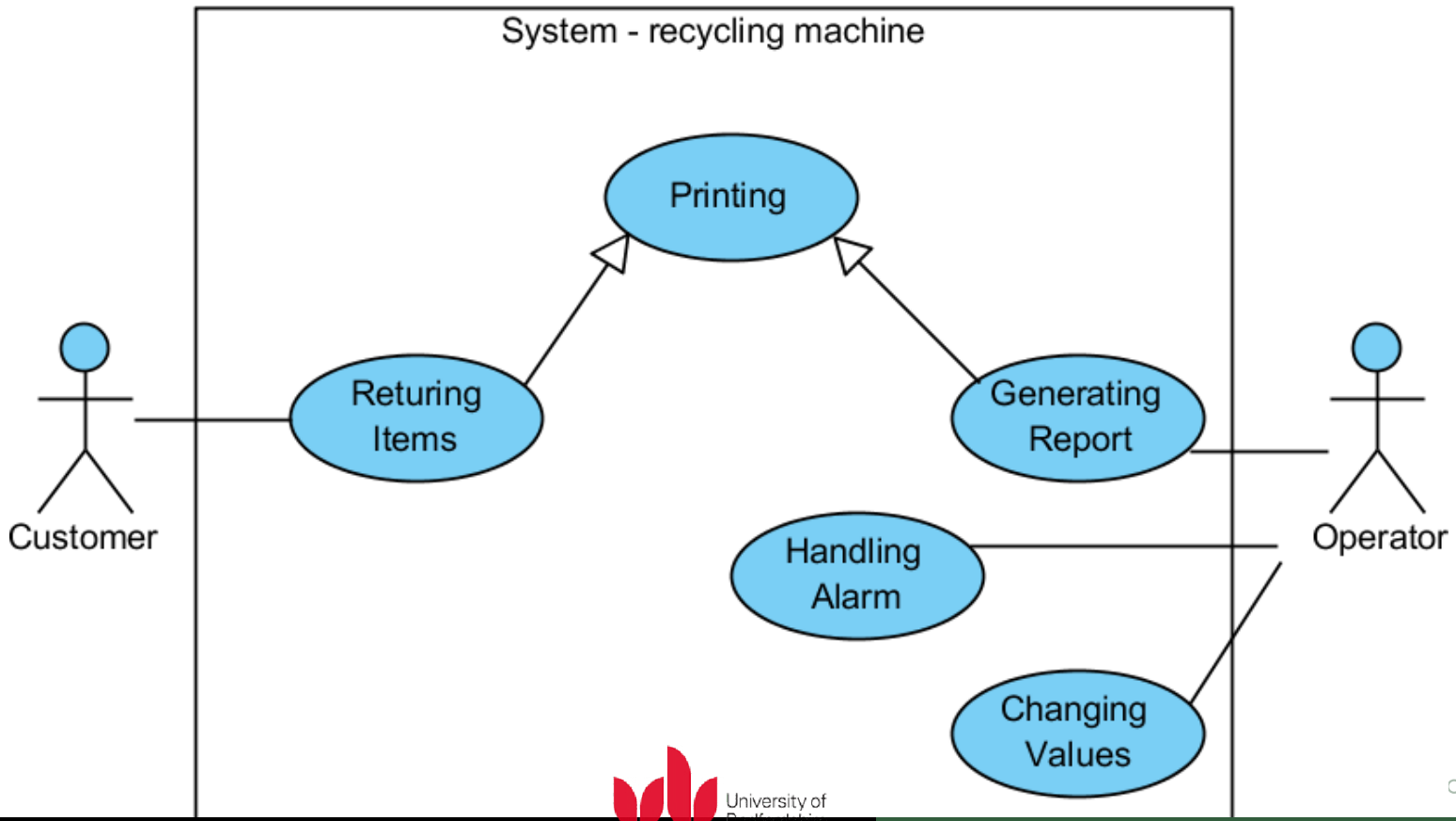
Design Stage

- Design aims to give such a detailed description of a system that actual coding can start based on it.
- Design model is required to represent the description.
- Actual implementation environment need to be taken into account when building up the design model.
- Functional localisation guarantees that any functionality change will not influence large part of a system.

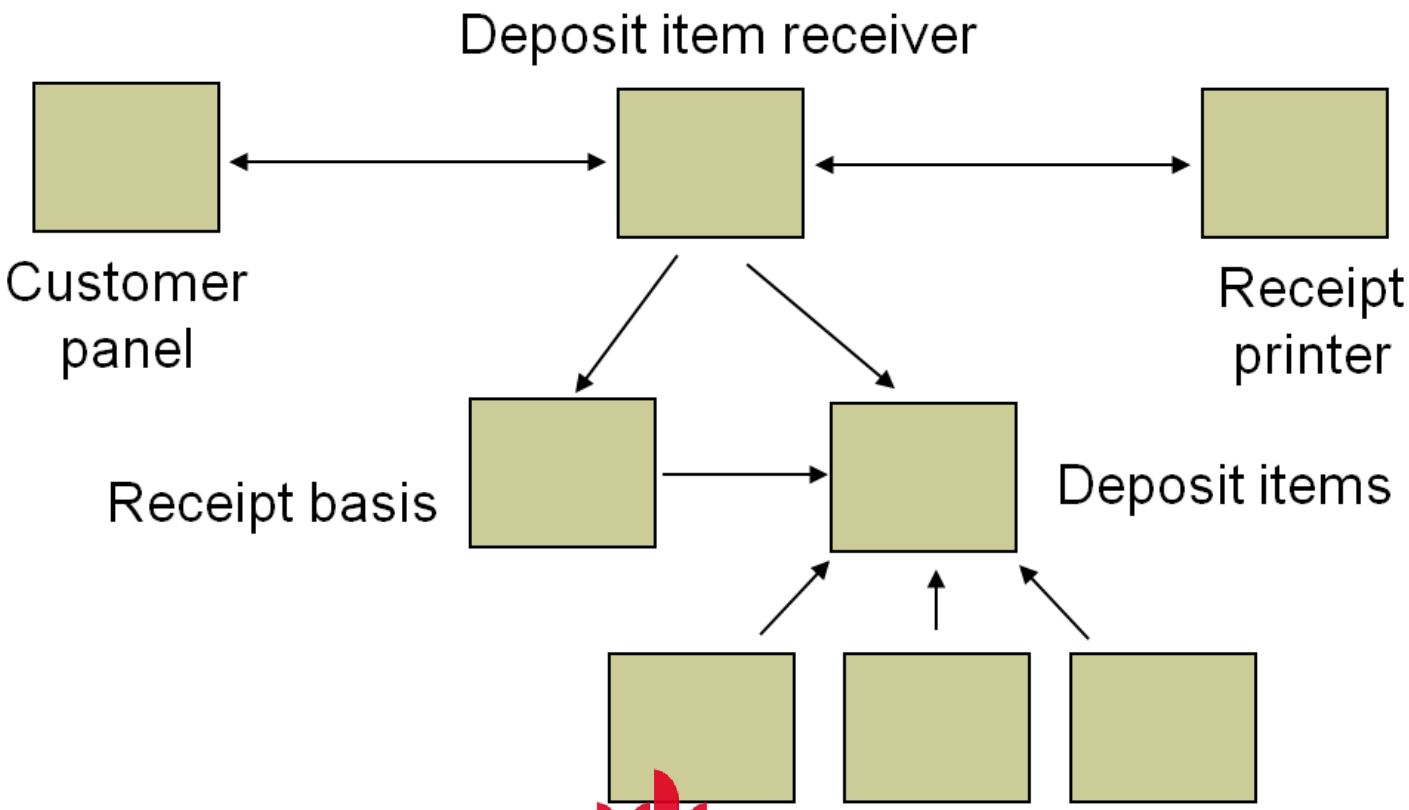
Conversion of OOAM

- The process of constructing a design model starts from OOAM, to guarantee the logic and traceability
- Conversion:
  - Change objects, also known as "lifelines" in Visual Paradigm, in an OOAM to "blocks"
  - Split a block into several blocks or add more blocks if necessary
- Traceability - bridging OOAM and OODM enables one to see how OODM "inherits" system's logic from OOAM
- Functional localisation - functionality is restricted in each block

Return Item Use Case in recycling machine - example



Return Item Use Case in recycling machine - example



What next?

- Details, then, need to be given for each block.
- BUT, inputs and outputs of a block MUST be known before thinking about the details of a block.
- Sequence diagram helps to find out the inputs and the outputs.

Sequence Diagrams

Sequence Diagrams

- Sequence diagram shows how blocks (objects) within a used case cooperate with each other. It shows the occurrence of a sequence of events as
- the responses to stimuli, which is called stimuli-responses relationship or the internal logic of a system.
- A stimulus comes from a block and goes to another block. It triggers a sequence of events in that block.
- The receiver may also send "results" (feedback) back to the sender.
- Events must be known before constructing a sequence diagram.

Sequence Diagram - example

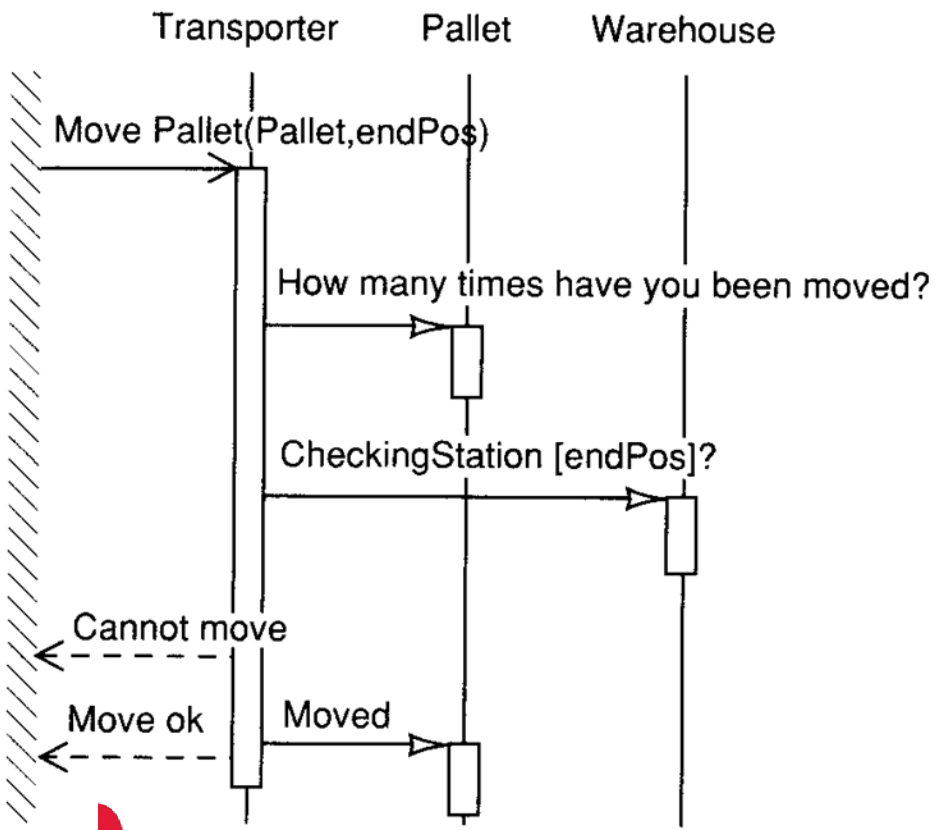
Events

The operator gives a request to move

Check if the Pallet can be moved

IF NrMoved > MaxMove
THEN check if move is to checking station

IF checking station
THEN Execute Move
ELSE
ENDIF
ELSE
Move
ENDIF

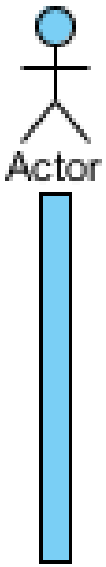


Sequence Diagrams in Visual Paradigm

The Sequence Diagram models the collaboration of objects based on a time sequence. It shows how the objects interact with others in a particular scenario of a use case. With the advanced visual modeling capability, you can create complex sequence diagram in few clicks. Besides, VP-UML can generate sequence diagram from the flow of events which you have defined in the use case description.

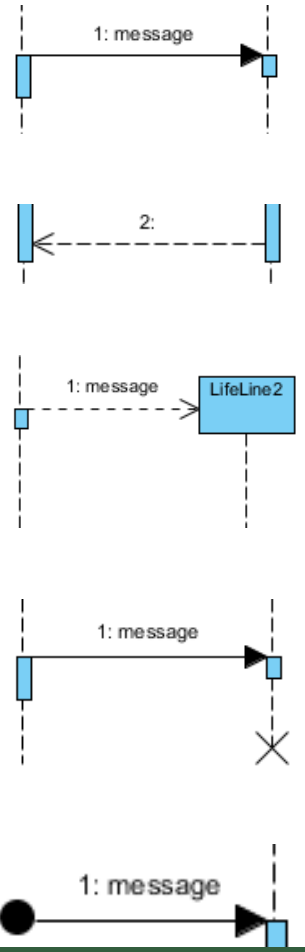
Notation/syntax - Actor

- An Actor models a type of role played by an entity that interacts with the subject but which is external to the subject.
- Actors may represent roles played by human users, external hardware, or other subjects; some entity that is relevant to the specification of its associated use cases.
- A single physical instance may play the role of several different actors and, conversely, a given actor may be played by multiple different instances.



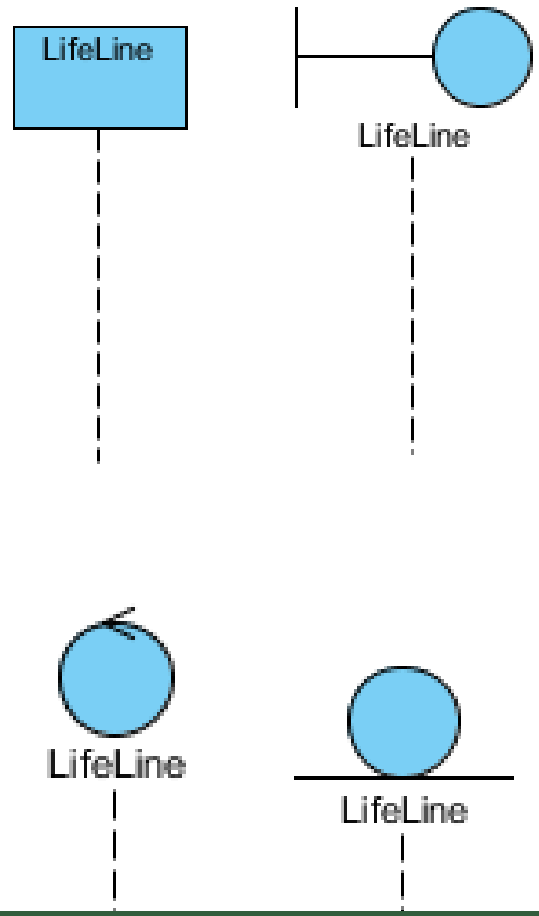
Notation/syntax - Message

- Defines a communication between Lifelines of an Interaction.
  1. *Call message* - invocation of operation of target lifeline.
  2. *Return message* - pass of information back to the caller of a corresponded former message.
  3. *Create message* - instantiation of (target) lifeline.
  4. *Destroy message* - request of destroying the lifecycle of target lifeline.
  5. *Found message* - receiving event occurrence is known, but there is no (known) sending event occurrence



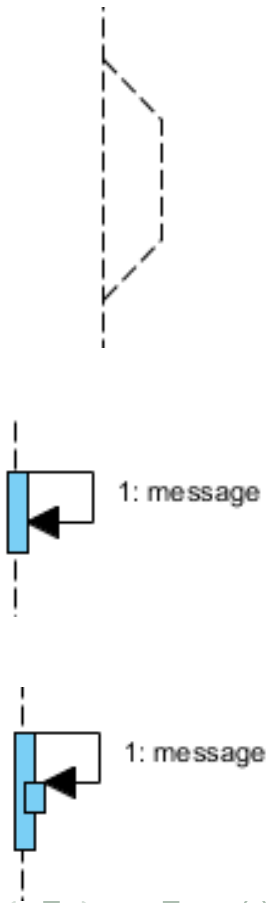
Notation/syntax - Lifeline

- A lifeline represents an individual participant in the Interaction.
  - Entity object models information. It holds information and some operations that naturally related to the information.
  - Boundary/interface object models input and output information and operations that process the information.
  - Control object models

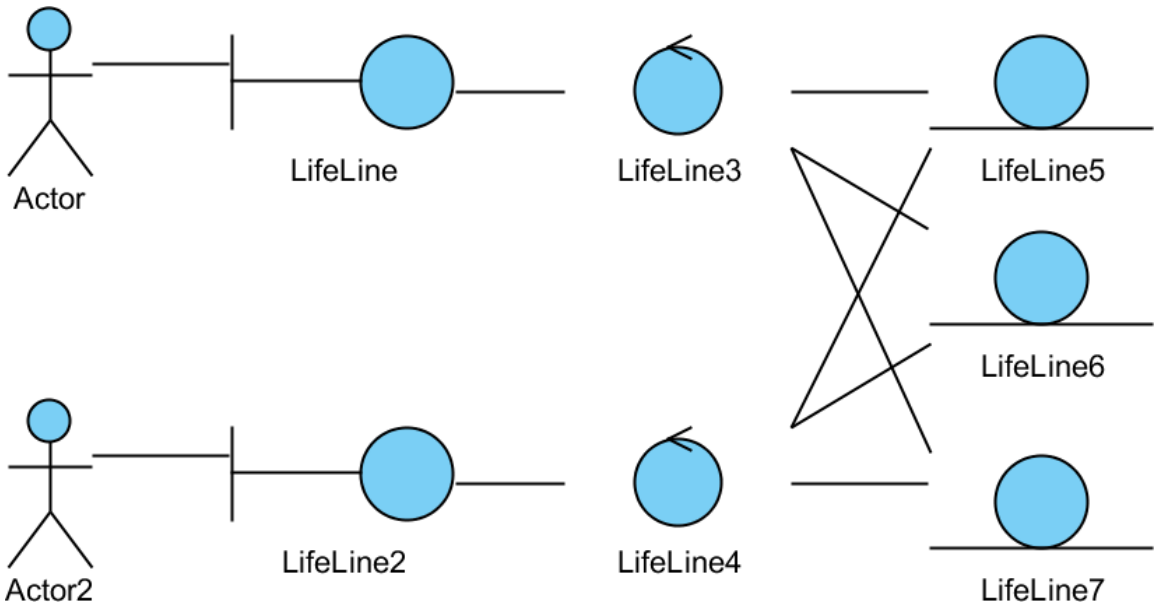


Notation/syntax - Miscellanea

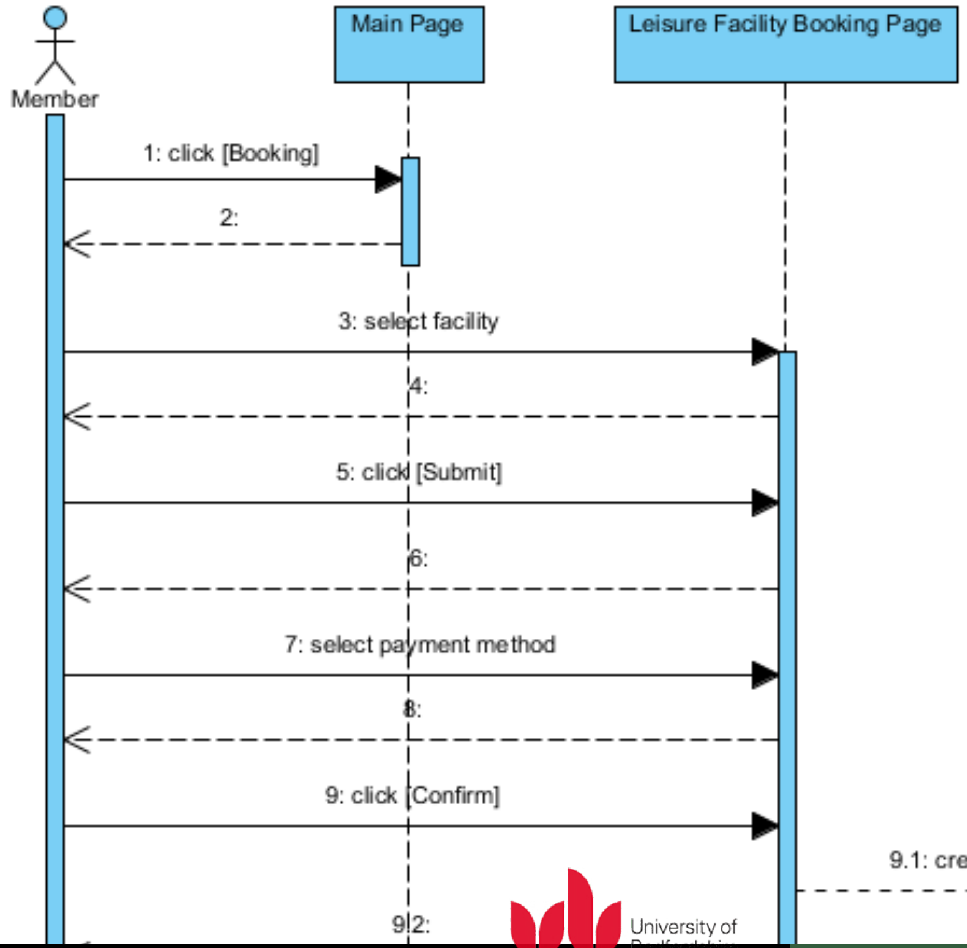
- Miscellaneous notation of interest:
  - A Concurrent represents a session of concurrent method invocation along an activation. It is placed on top of an activation.
  - Self message is a kind of message that represents the invocation of message of the same lifeline.
  - Recursive message is a kind of message that represents the invocation of message of the same lifeline. It's target points to an activation on top of the activation where the message was invoked from.



Stereotype/Analysis Classes



Visual Paradigm - example

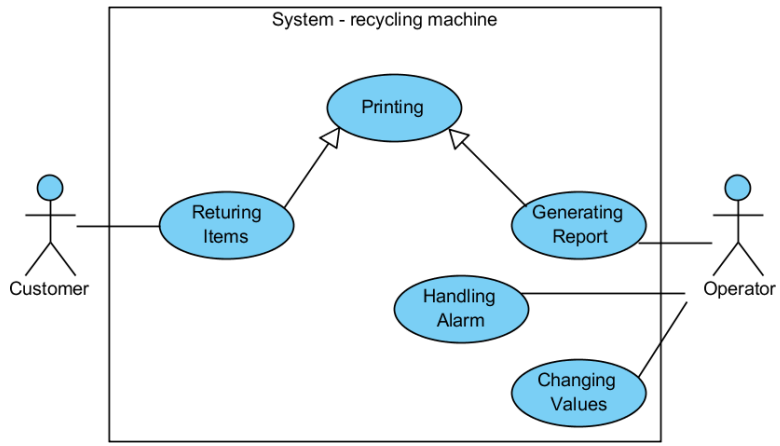


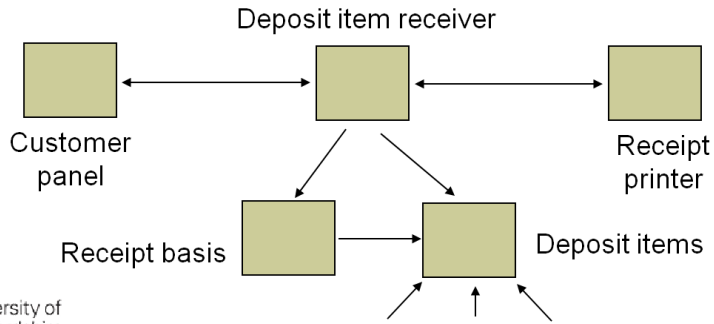
Pragmatics

- List all blocks right-hand side of the system border,
- Draw a life line for each block,
- List all events on the left-hand side of the system border,
- Draw vertical bars on the corresponding life lines to represent the events.
- Identify and draw signals,
- Identify and draw messages,
- Identify and draw feedback signals.

Example: (focusing on Returning item use case)

- Blocks:
  - Customer panel (interfacing)
  - Deposit item receiver (measurement)
  - Receipt basis (values)
  - Deposit item (categories)
  - Receipt printer (printing)





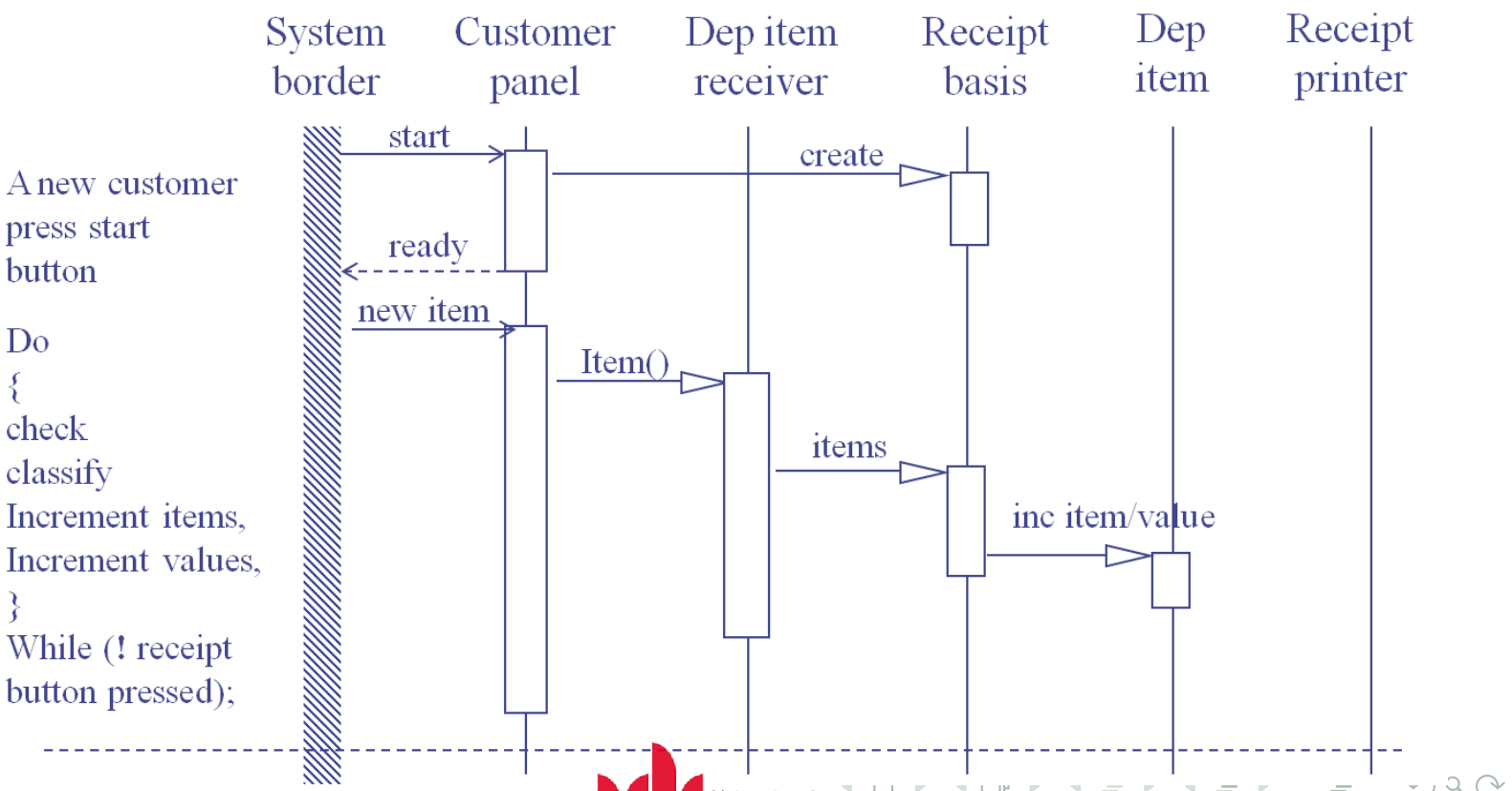
Events (when a customer insert an item)

```
If (customer is new)
{
        Create a new account;
}
Do
{
        If (returned item is acceptable == true)
        {
                Classify;
                Increment items;
                Increment values;
        }
        else
```

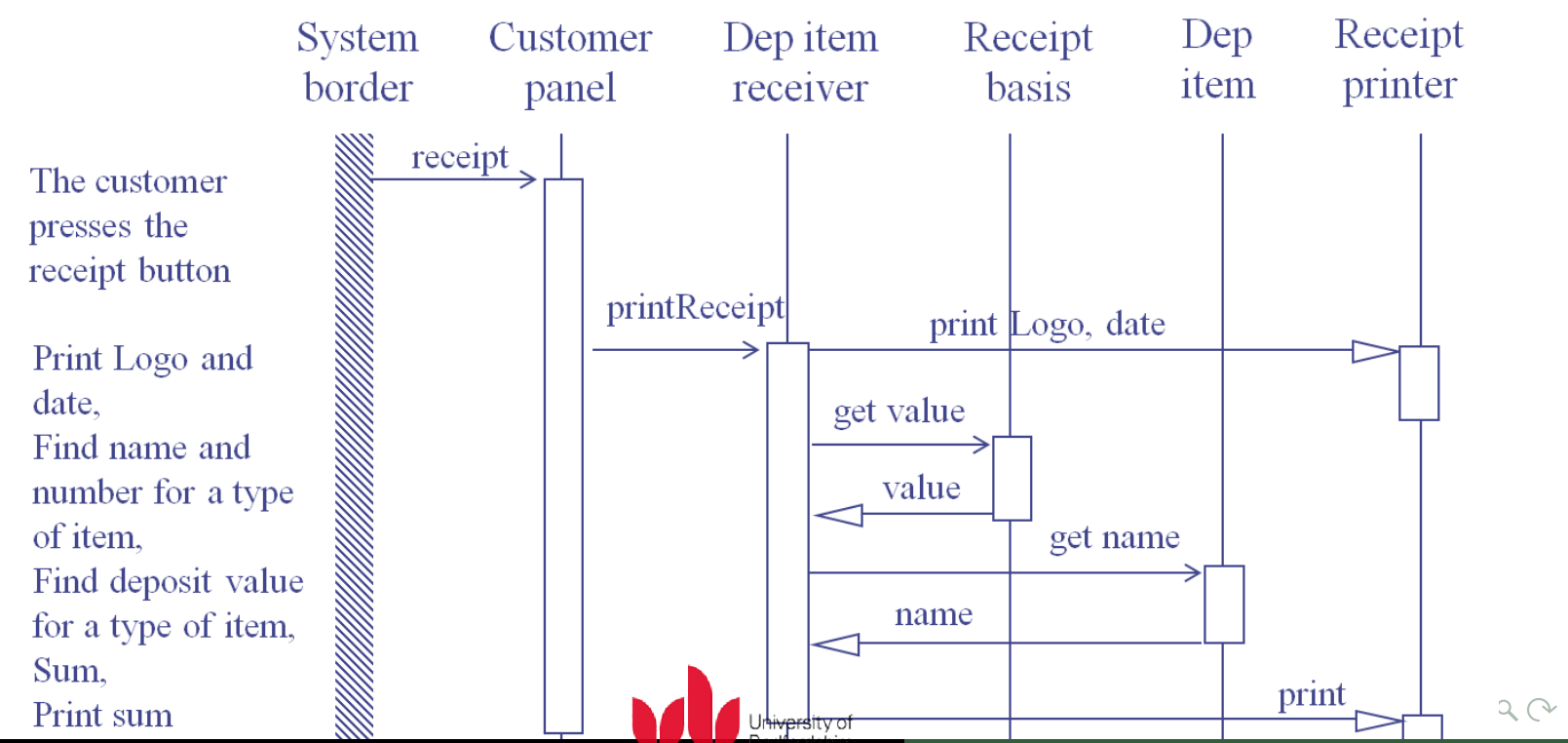
Events (when the customer presses receipt button)

```
Print Logo and date;
for (index = 0; index < 3; index ++)
{
    Find name and number for a type of item;
    Find deposit value for a type of item;
    Sum;
    Print sum;
}
Ready for the next customer;
```

SEQUENCE DIAGRAM # 1 (WHEN A CUSTOMER INSERT AN ITEM)



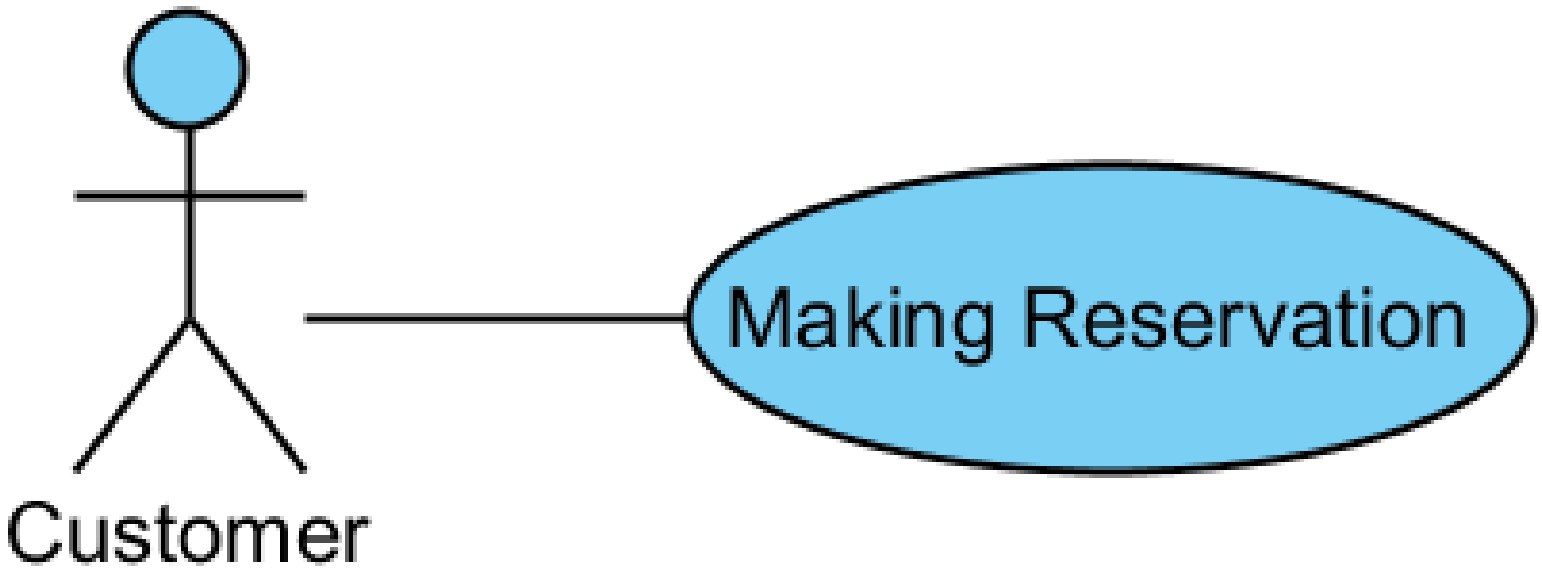
SEQUENCE DIAGRAM # 2 (WHEN THE CUSTOMER PRESSES RECEIPT BUTTON)



Case Study

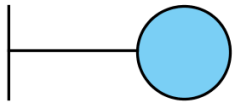
EuroStar is a vehicle hire company. It provides a variety of vehicles for its customers. Each vehicle has a record in the company's database, including Register Number, Maker, Model, Colour, Engine Size, etc.
The company has a manager and a number of Register Staff. The register staff takes a customer's reservation and searches for a vehicle that matches the order by sending a query to the company's computerised database. When the requested type of vehicle is available, the staff confirms the reservation. If it is not available, the staff gives suggestions of alternative vehicle.
Customers are required to return vehicles to the company before a Return Time. They are liable to a fine if they return

Use case diagram



Analysis modeling

- Console - interface between the system and the user, an interface object.

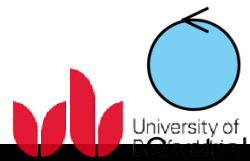


Boundary

- Database - vehicle records and availability, an entity object.

Entity

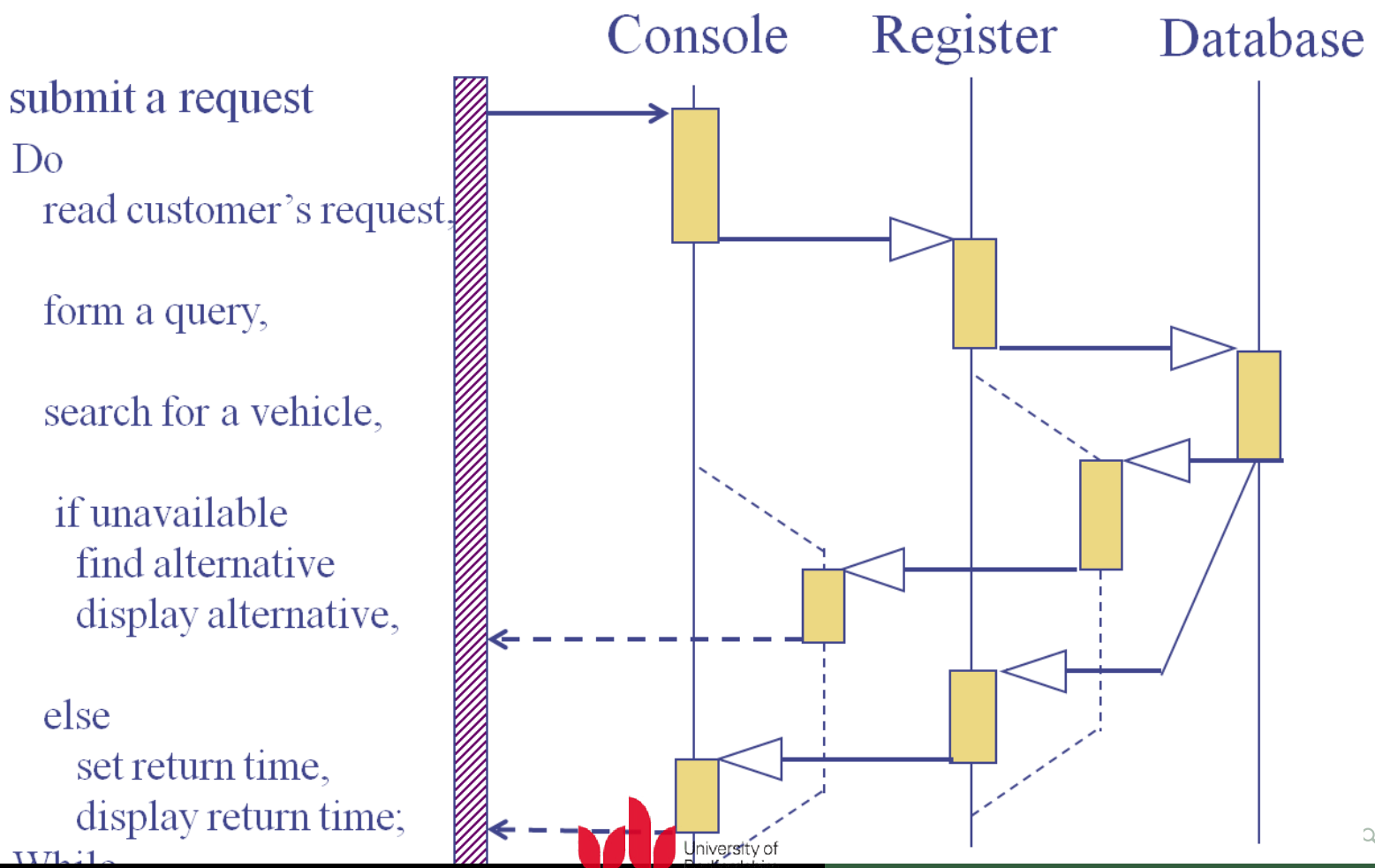
- Register - Control of data flow, a control object.



Sequence diagram

- Events:
  - read customer's request,
  - form a query,
  - search for a vehicle,
  - find alternative if the request vehicle is unavailable,
  - display alternative vehicle,
  - set return time,
  - display return time.

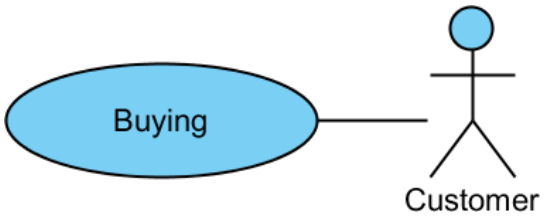
SEQUENCE DIAGRAM #1

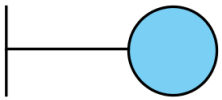


Selection in Sequence Diagram

- If. . . else. . .
- Condition repression.
- Branch of control in the receiving block's lifeline.

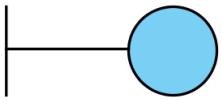
A soda machine example
Customers buy soda drinks from a soda machine





- FRONT Boundary

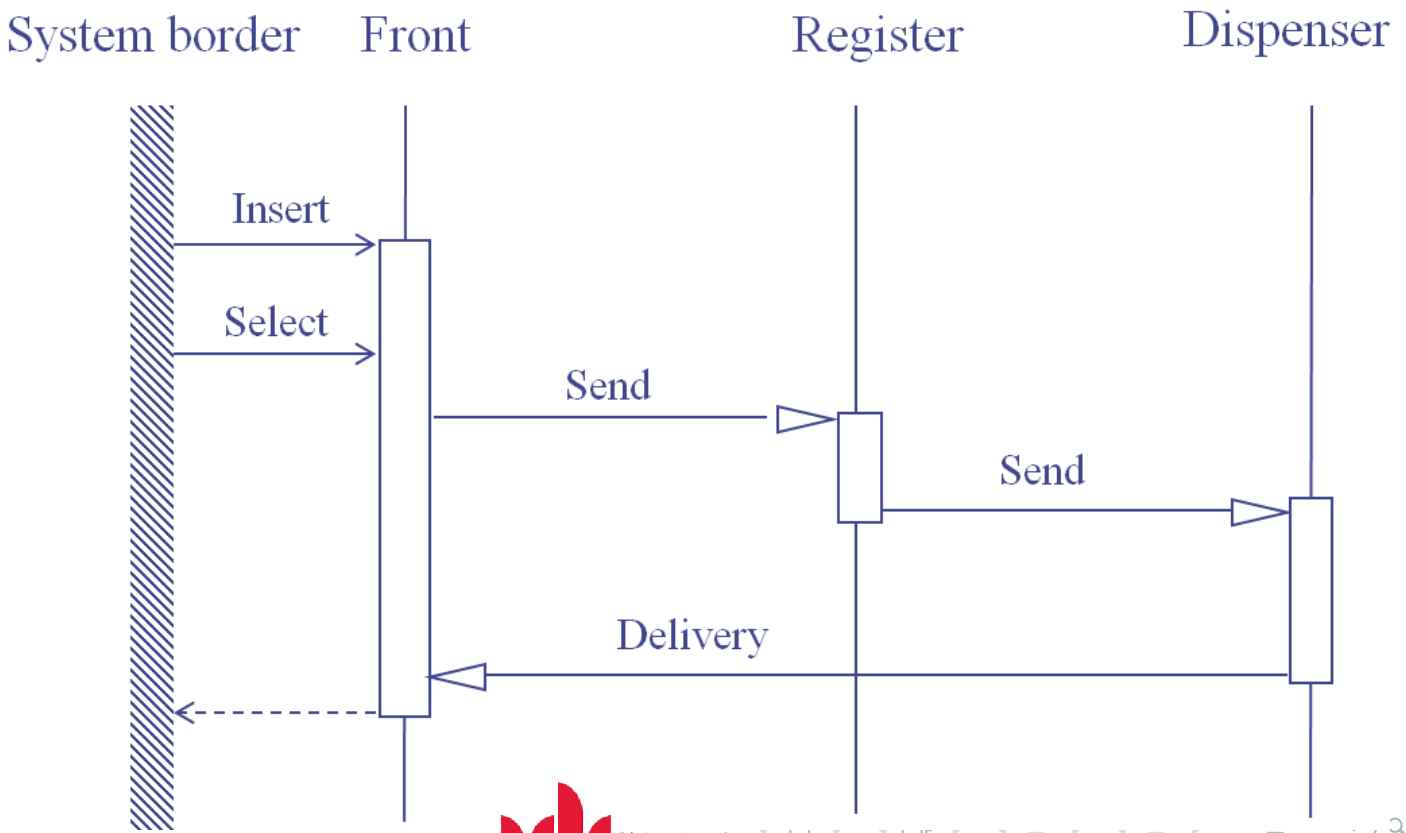
- REGISTER Control



- DISPENSER Boundary

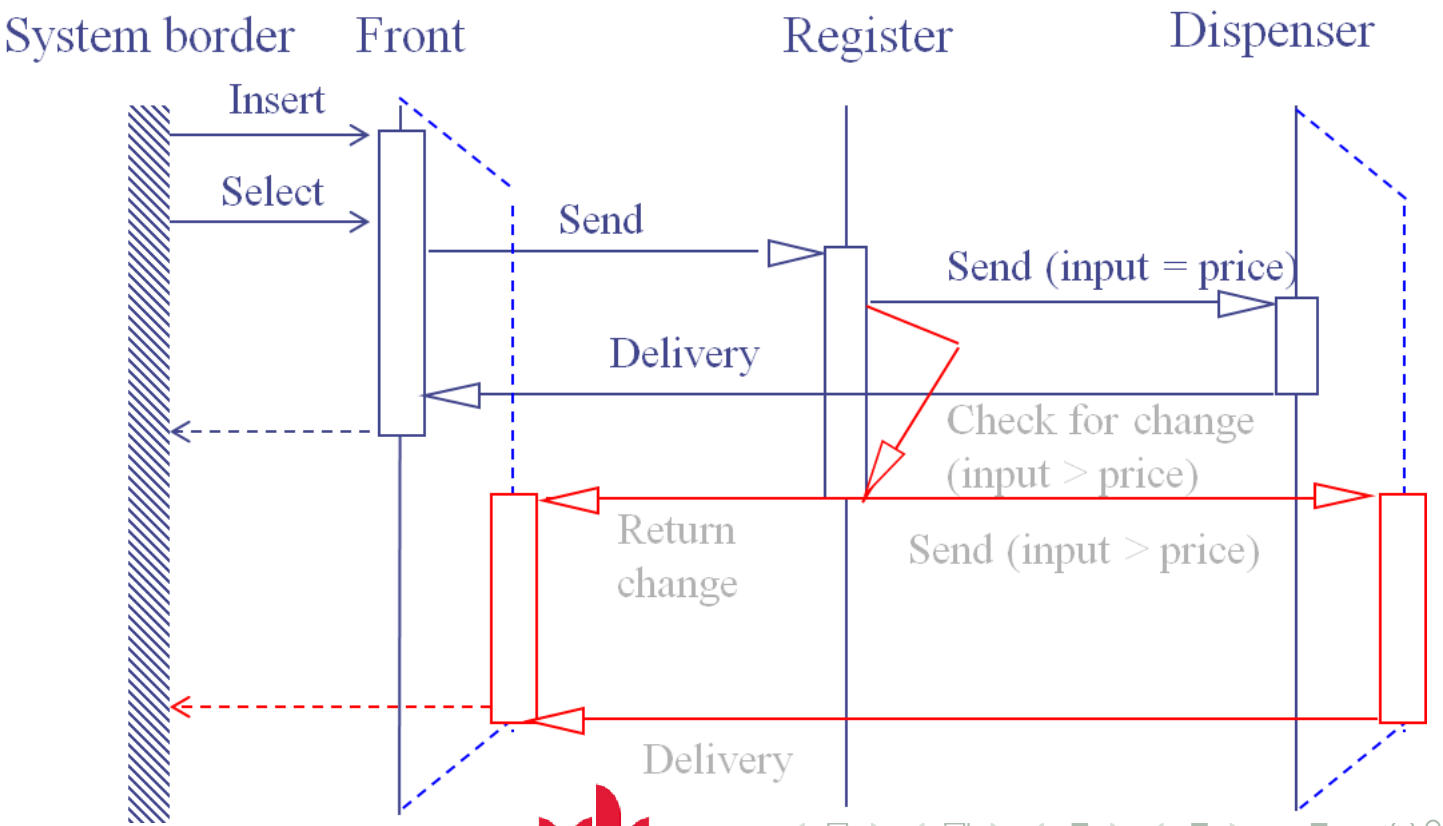
Selection in Sequence Diagram

When input (inserted coins) = price



Selection in Sequence Diagram

When input (inserted coins) > price

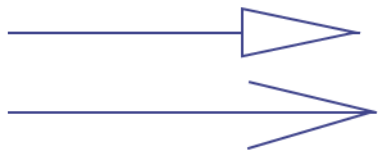


Synchronous and asynchronous

- A synchronous message/signal is a control which has to wait for an answer before continuing. Hence, the sender passes the control to the receiver and cannot do anything until the receiver sends the control back.
  - A bank teller might send a credit request to the bank manager for approval and must wait for a response before further serving the customer.
- An asynchronous message is a control which does not need to wait before continuing. Hence, the sender actually does not pass the control to the receiver. The sender and the receiver carry on their work concurrently.
  - A bank customer could apply for credit but can receive banking information over the phone or request money from an ATM, while waiting to hear about the credit

Synchronous and asynchronous - syntax

- asynchronous



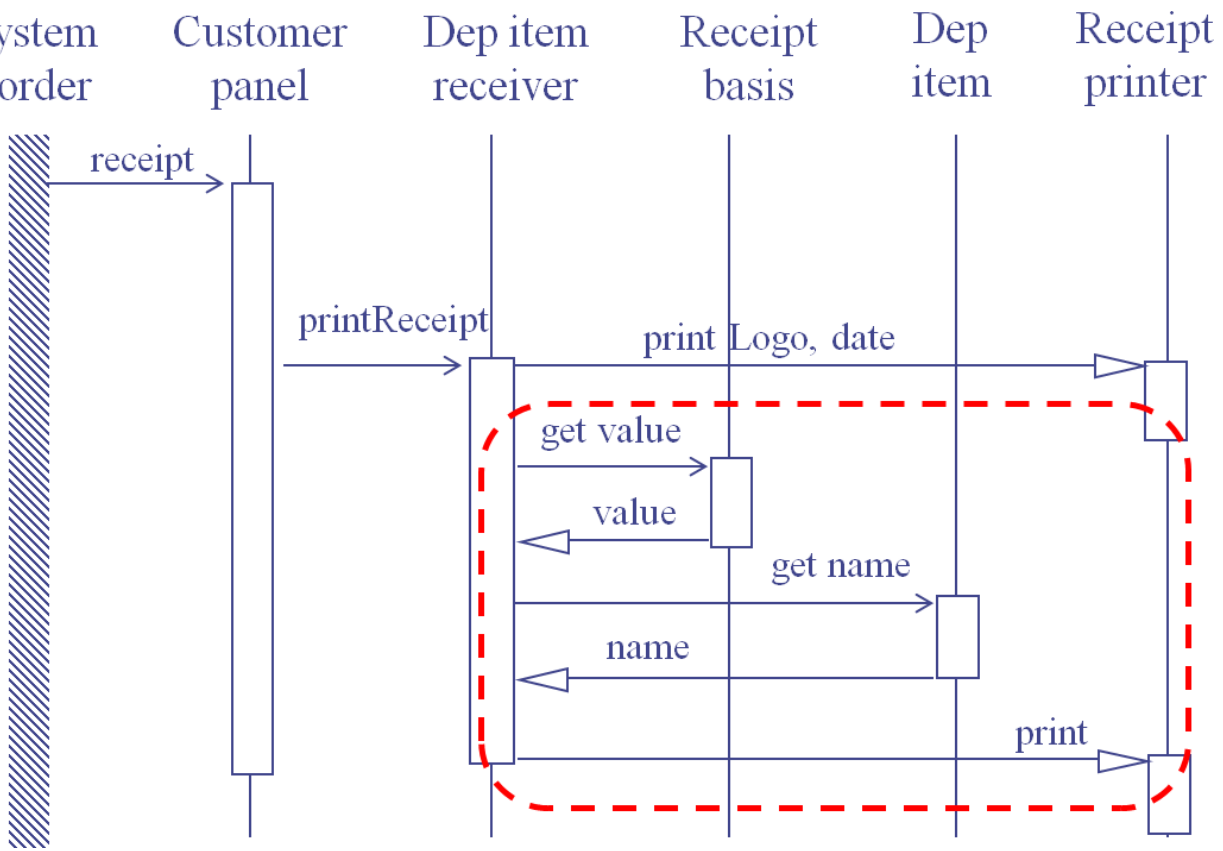
synchronous

- Example: the "return item" use case in the recycling machine.
  - Concentrate on the rounded rectangle and answer the question:
    - "Should getName and getValue be synchronous or asynchronous?"

Selection in Sequence Diagram

The customer presses the receipt button

Print Logo and date,
Find name and number for a type of item,
Find deposit value for a type of item,
Sum,
Print sum



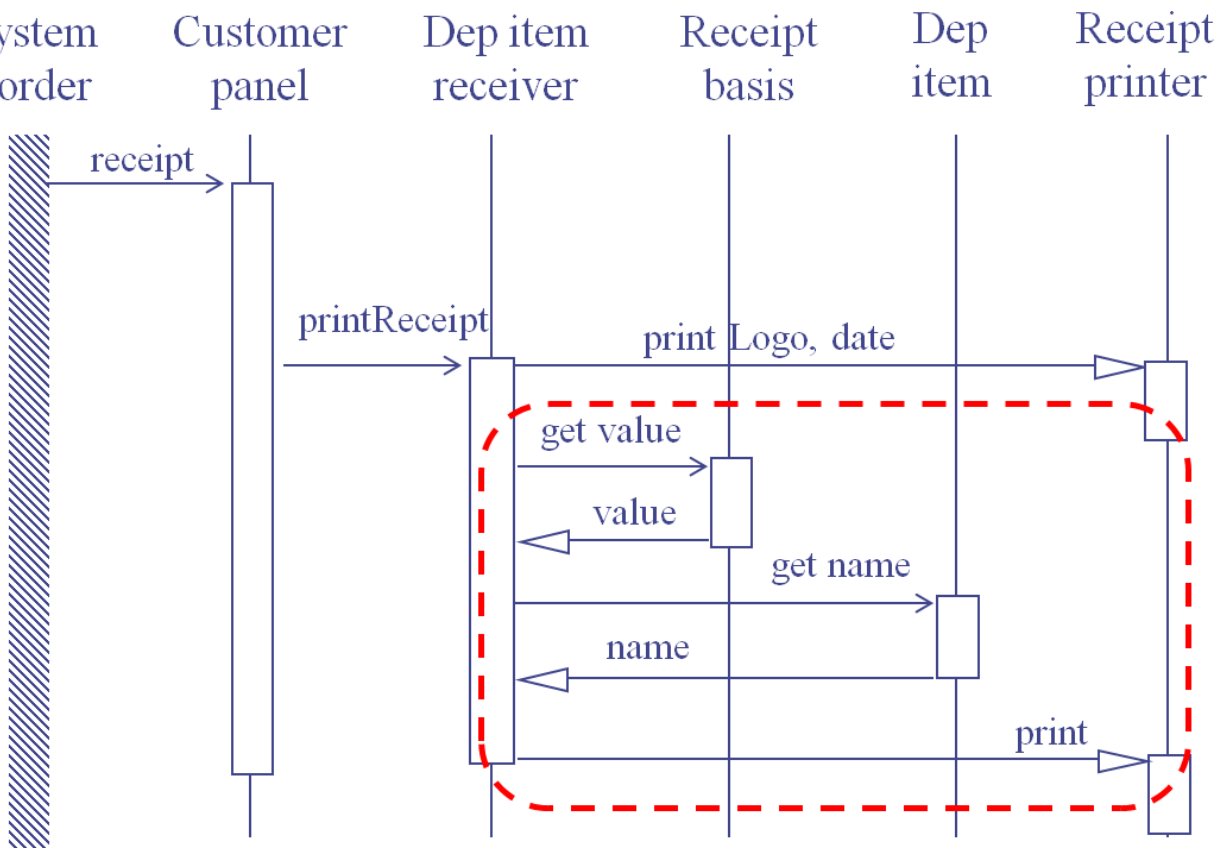
Synchronous or Asynchronous?

- On one hand, getValue signal and getName signal can be asynchronous as the system does not need to wait for values before requesting name.
- On the other hand, "print" message could be sent out to the printer if getValue and getName were asynchronous.
- For this reason, getValue signal and getName signal must be synchronous.

Selection in Sequence Diagram

The customer presses the receipt button

Print Logo and date,
Find name and number for a type of item,
Find deposit value for a type of item,
Sum,
Print sum



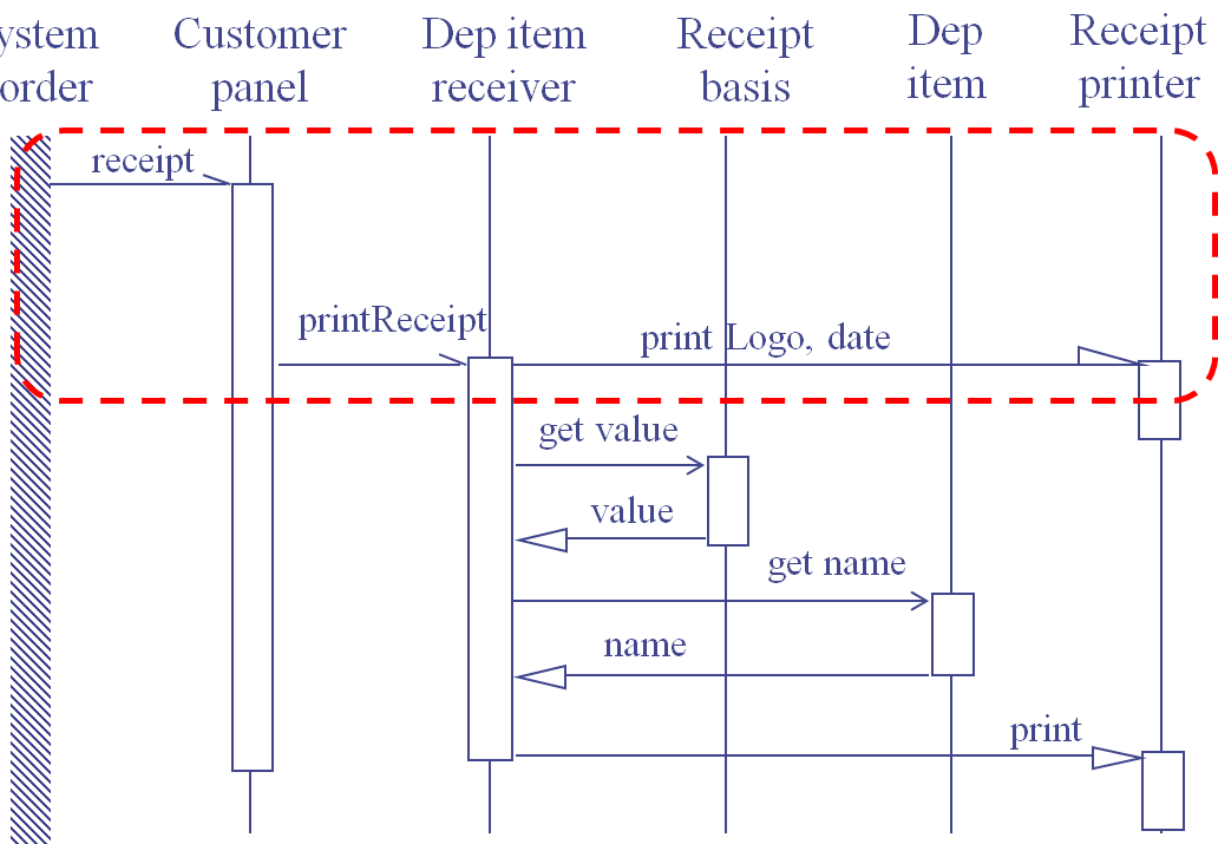
Synchronous or Asynchronous?

- Concentrate on other signals and messages.
  - receipt signal: it does not even need a feedback. Therefore, it can asynchronous.
  - printReceipt signal: it can be asynchronous, too.
  - printLogo, date message: it can also be asynchronous, as no feedback is required.

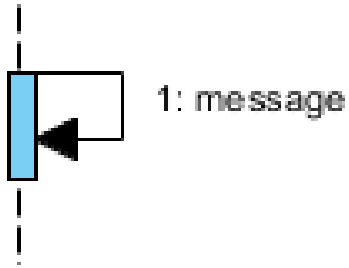
Selection in Sequence Diagram

The customer presses the receipt button

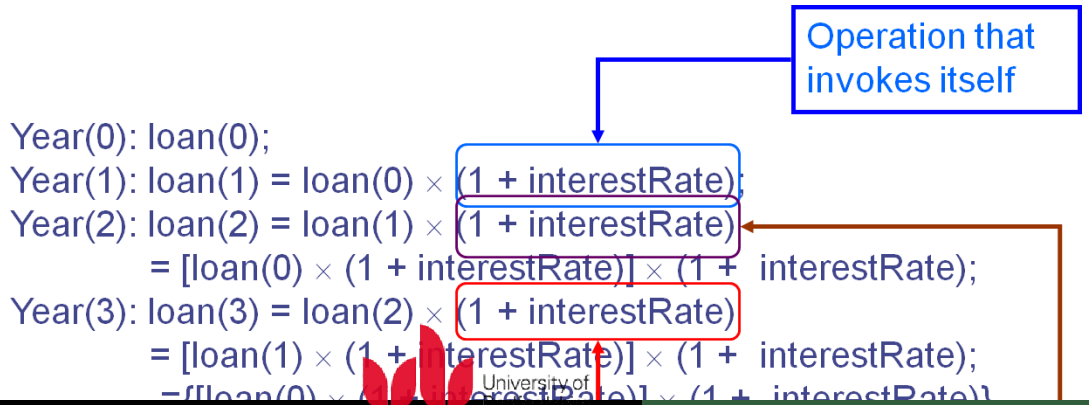
Print Logo and date,
Find name and number for a type of item,
Find deposit value for a type of item,
Sum,
Print sum



Recursion in Sequence Diagrams

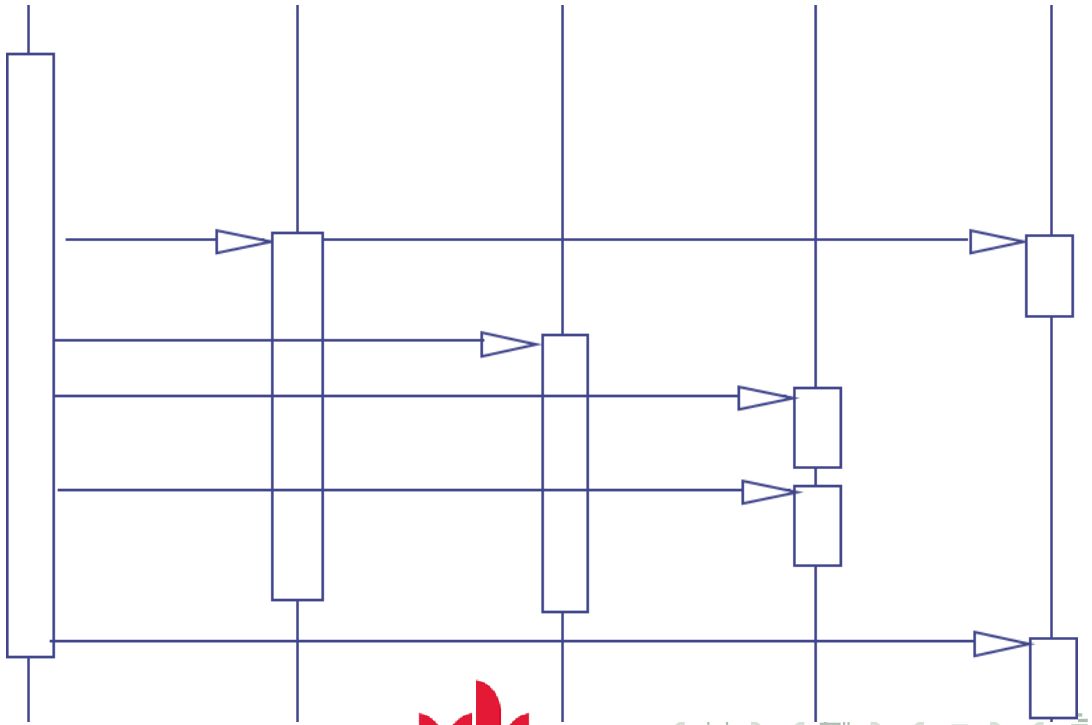


- Recursion or self-call - an object/block may have an operation that invokes itself.
- Example (personal loan):



Structure in Sequence Diagrams

- Centralised structure - Fork
  - Everything is handled and controlled by the left-most block.



Structure in Sequence Diagrams

- Decentralised structure - Stair
  - There is no a central control block.