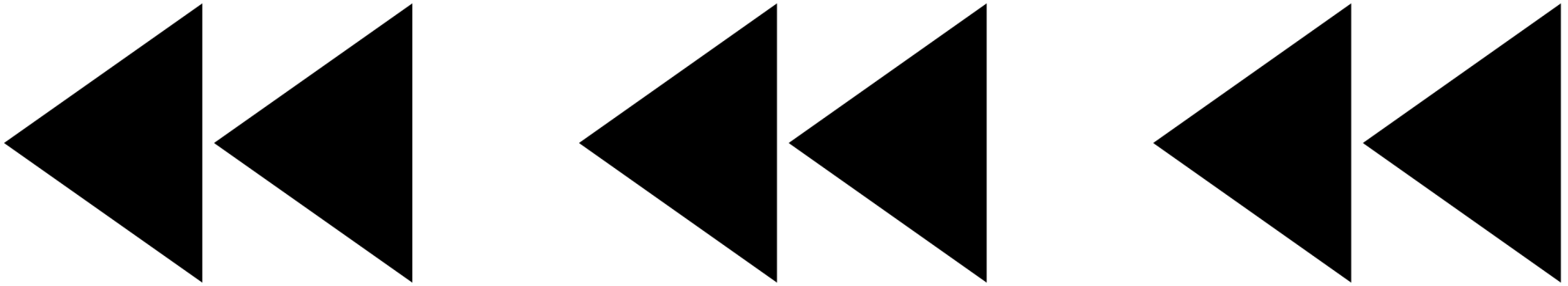
CS261
Software Engineering

Topic 4: System Design 1

James Archbold
James.Archbold@warwick.ac.uk

In association with Deutsche Bank

Hold tight, rewind

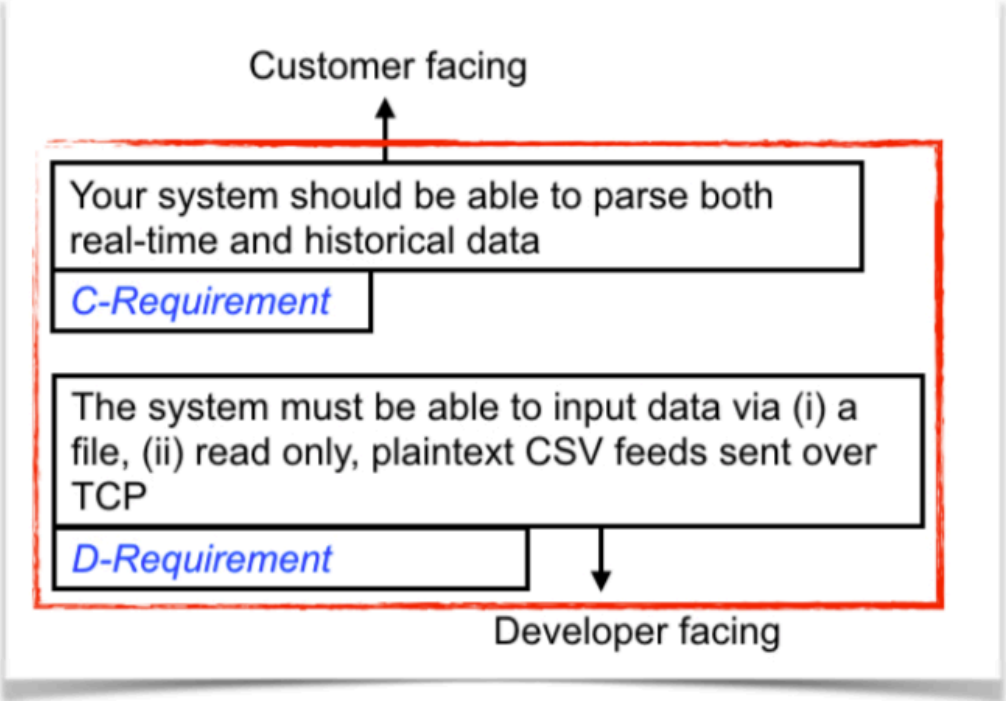


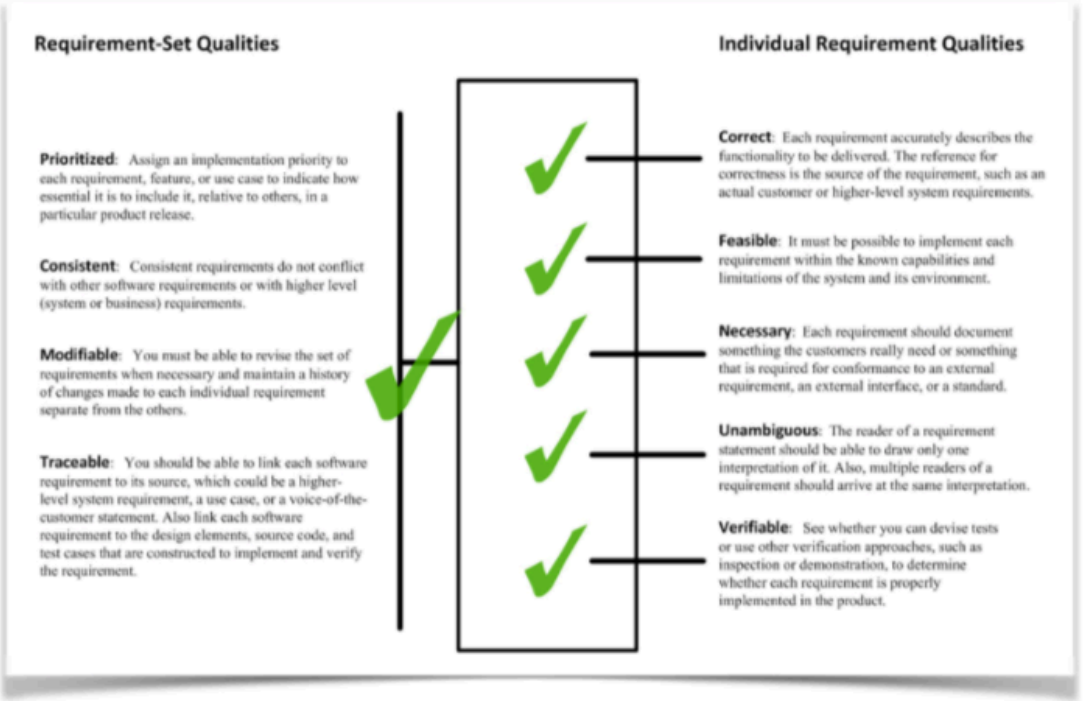
- Last week, we considered Requirements Analysis
- We classified requirements into two types:
  - *Functional Requirements* - Describe what the system should do
  - *Non-functional Requirements* - Constraints on the services or functions offered by the system e.g. availability, performance

Hold tight, rewind

A single requirement can bridge from customer to developer

Considered criteria for good requirements and good sets of requirements

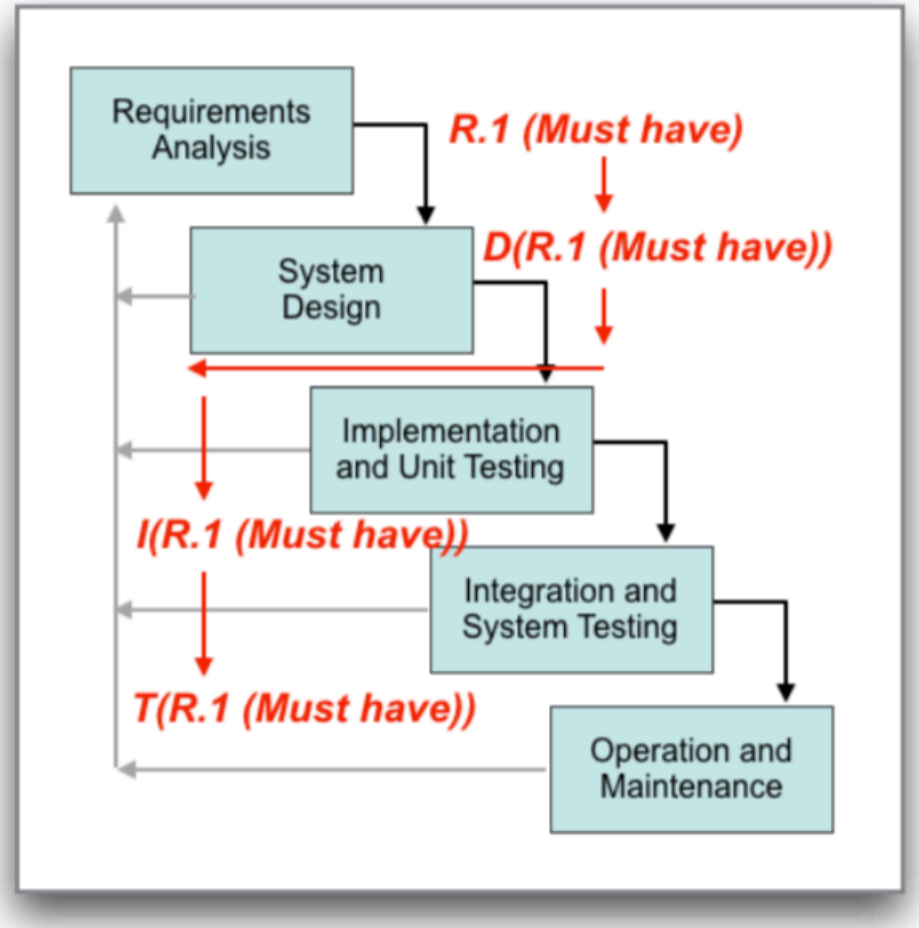




Hold tight, rewind

Considered prioritising requirements and also tracing them through the different processes in your software methodology.

Requirement	Must	Should	Could	Won't	Consensus
Req A	6	1			M
Req B	1	5	1		S
Req C	3	1	2	3	?
Req D			1	6	W
Req E	7				M
Req F		1	6		C
Req G		3	1	3	?
...					



System Modelling

Sommerville Chapter 5

Session Outline

- System modelling goals
- Context models
- Interaction models
- Structural models
- Behavioural models
- Model-driven engineering

System Modelling Goals

- *Requirements Analysis* - building a contract between customer and developer
  - **Customer** - natural language-based, supported by diagrams
  - **Developer** - must be completely unambiguous, more mathematical
- System design is supported by **system modelling** - developing mathematical-like models to:
  - help **clarify** functionality
  - provide a **basis** for development
  - **inform** design approaches and component-level decisions
- UML developed in the 1990s as general purpose modelling language; most recent updated (2.5) was released in 2015.

System Modelling - Perspectives

External

Model the context of the system

Interaction

Model the interactions between the system and its environment, or between components of a system.

Structural

Model the organisation of a system or the structure of the data being processed.

Behavioural

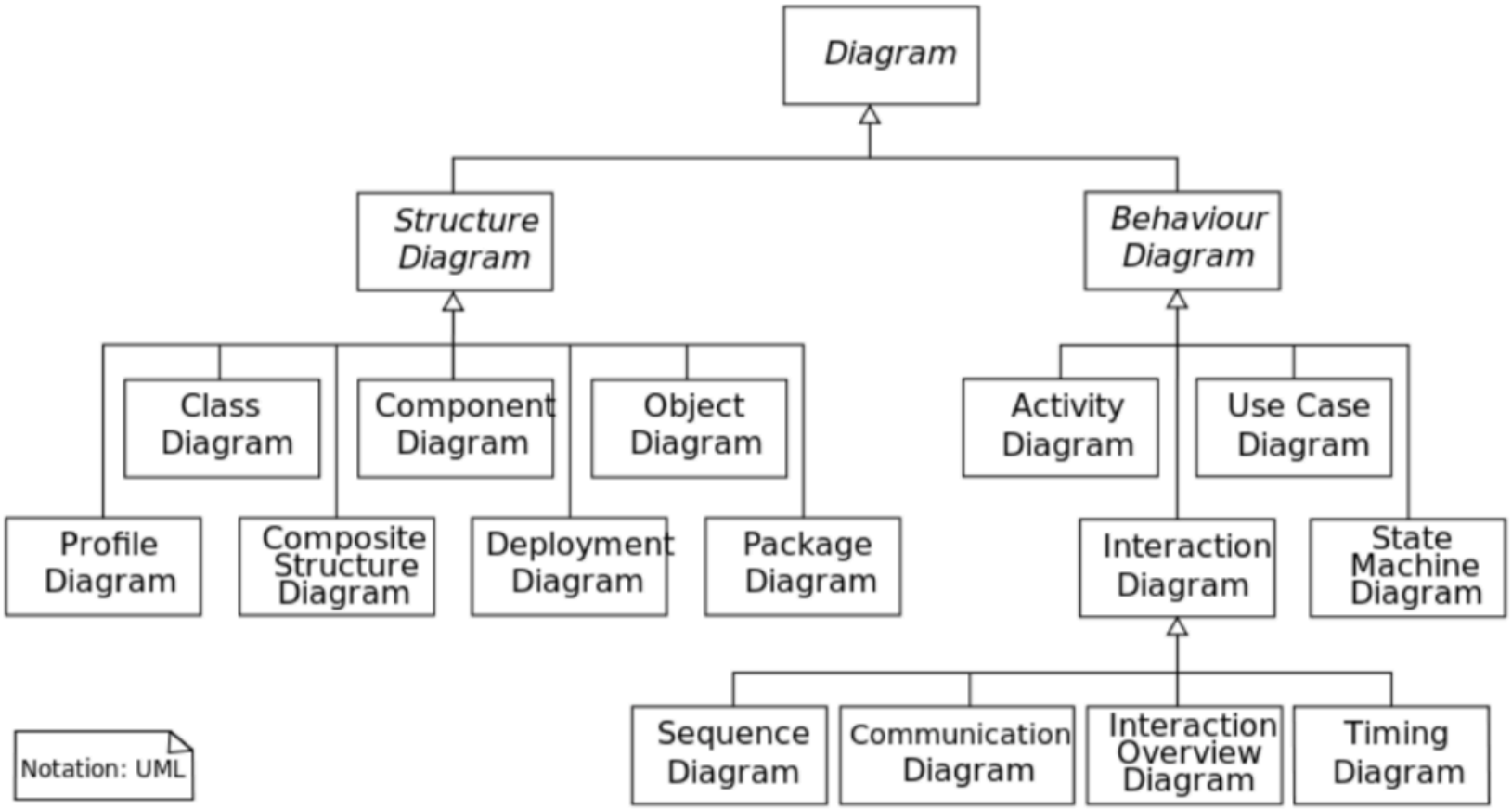
Model dynamic behaviour of the system

Unified Modelling Language (UML)

- We will explore a modelling technique which takes a description and expresses it formally (mathematically)
- The modelling language that we will use is called UML
- We will use UML to represent 2 different views of a system:
  - *Static/structural view* - static structure of the system (objects and their attributes and operations)
  - *Dynamic/behavioural view* - the dynamic behaviour of the system (collaboration among objects and their changing state)

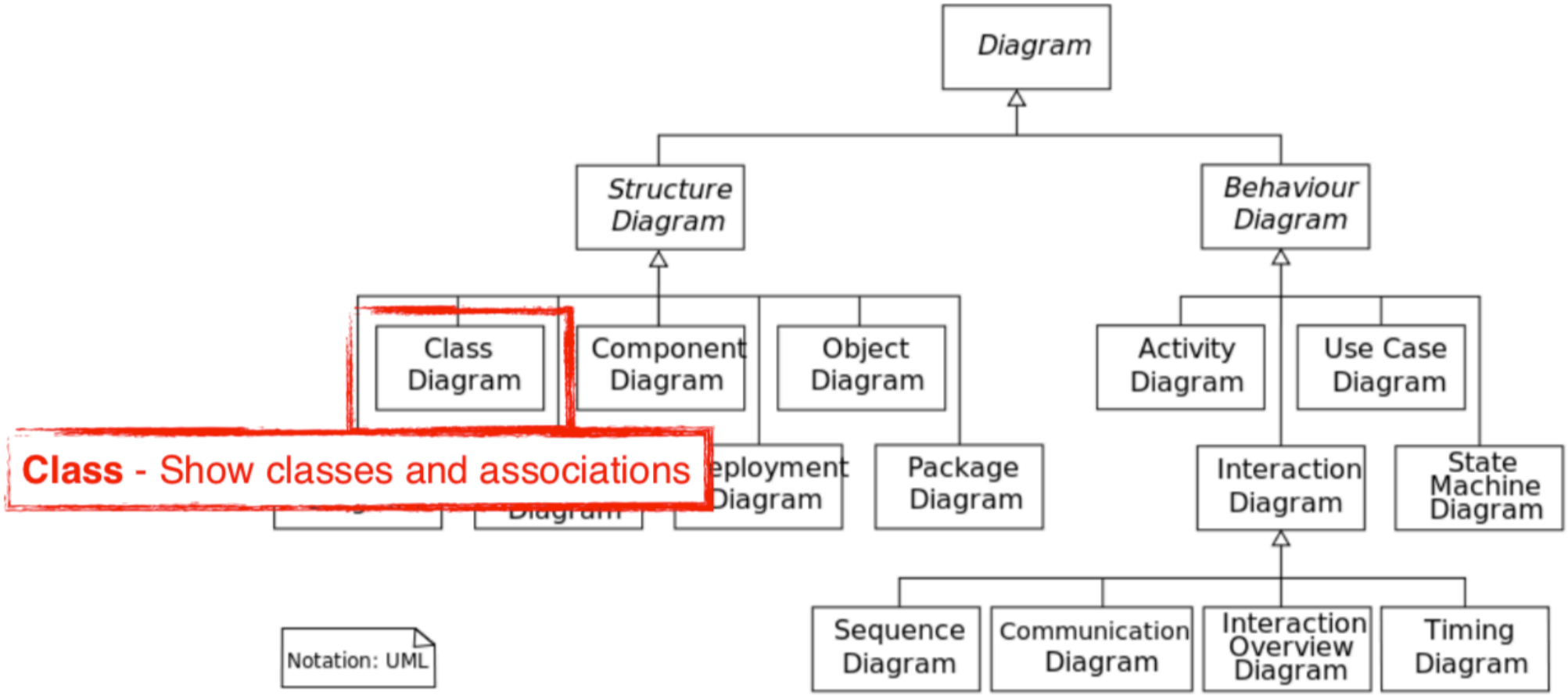
Unified Modelling Language (UML)

Taxonomy of diagrams (model types) that UML provides



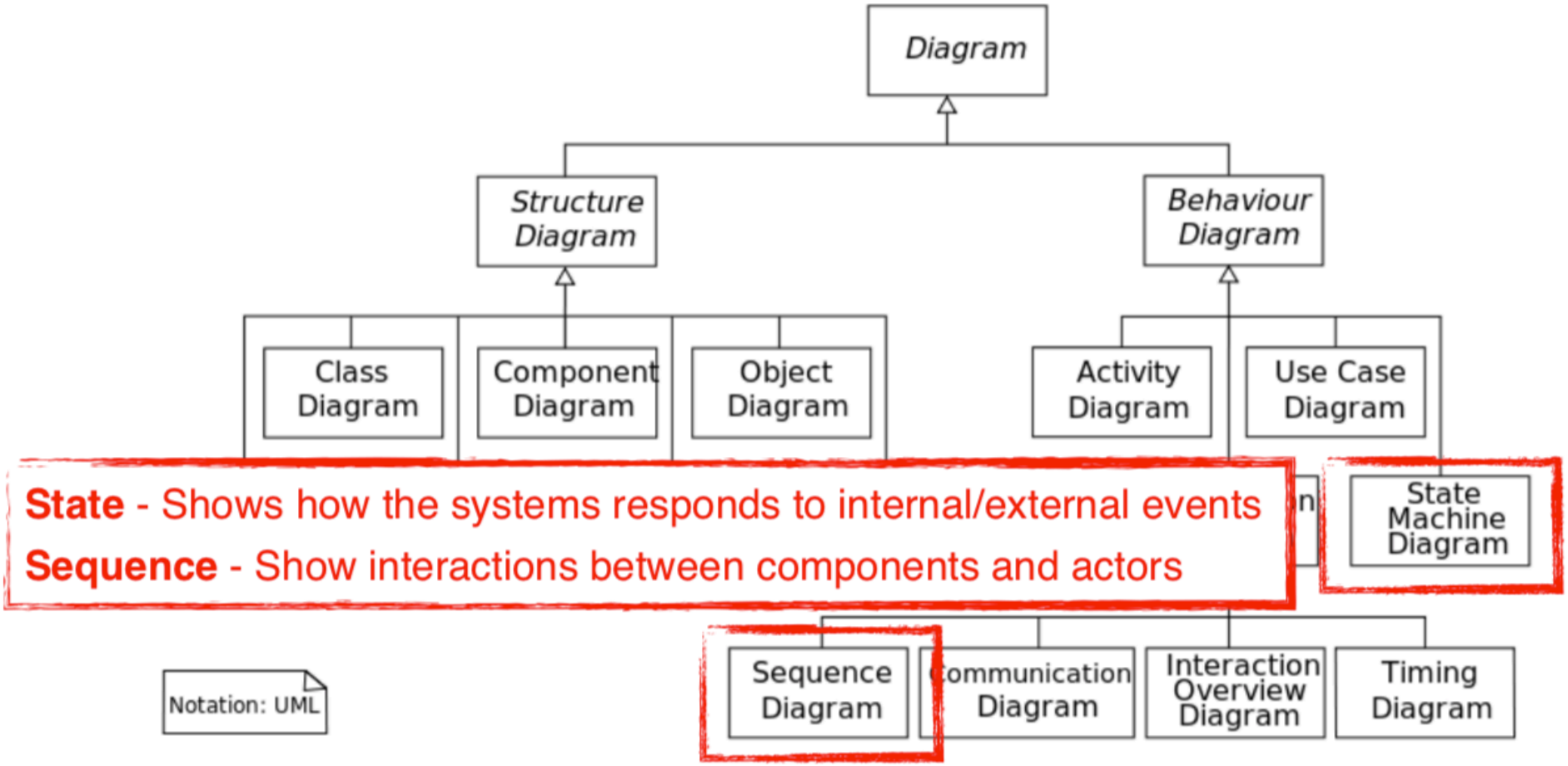
Unified Modelling Language (UML)

Static/structural view - static structure (objects, their attributes and operations)



Unified Modelling Language (UML)

Dynamic/behavioural view - dynamic behaviour (collaboration among objects and their changing state)



UML - Class Diagrams

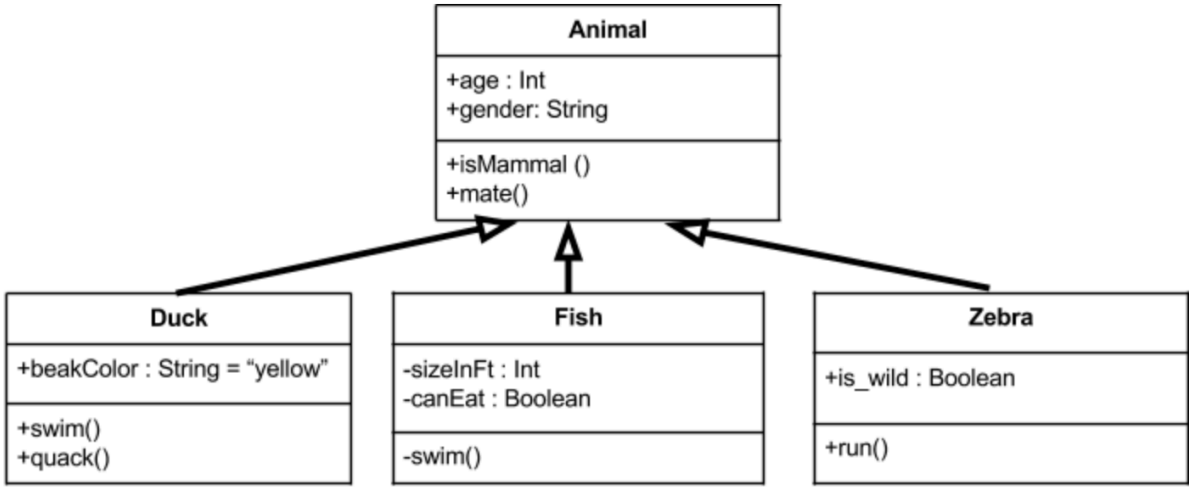
Creating a static/structural view of a system requires identifying entities (objects)

Grammatical approach based on a natural language description of the system.

Identification of tangible things in the application domain.

Behavioural approach to identify objects based on what participates in what behaviour.

Scenario-based approach. The objects, attributes and methods in each scenario are identified.



Grammatical Approach

Your task is to build a Trader ChatBot that will (i) provide answers to queries on the FTSE 100, (ii) provide access to financial news digests, (iii) employ some AI in order to become a personalised virtual trading assistant.

There are several aspects to this project and these are deconstructed as follows:

1. Your ChatBot must be able to access and interface with data on the FTSE 100.
2. Your ChatBot must be able to receive and respond to queries from a trader.
3. Your ChatBot must employ some AI so that it exhibits traits consistent with a personalised virtual trading assistant.

Grammatical Approach

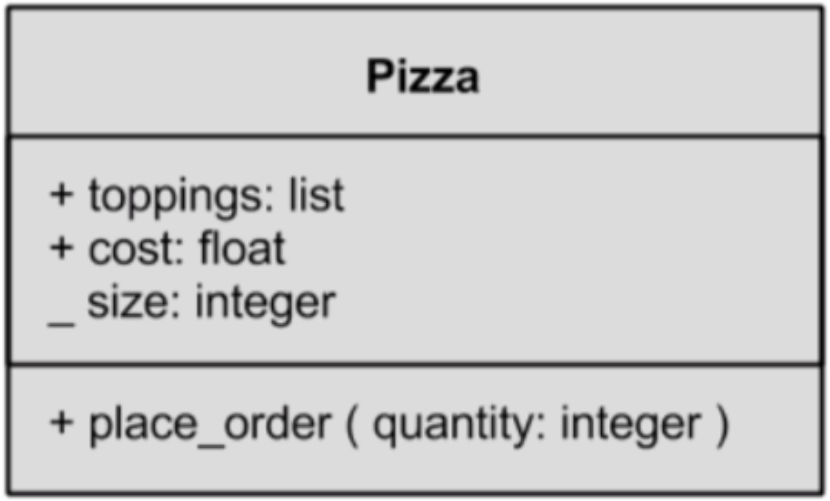
Your task is to build a **Trader ChatBot** that will (i) provide answers to queries on the **FTSE 100**, (ii) provide access to **financial news digests**, (iii) employ some AI in order to become a personalised virtual trading assistant.

There are several aspects to this project and these are deconstructed as follows:

1. Your **ChatBot** must be able to access and interface with data on the **FTSE 100**.
2. Your **ChatBot** must be able to receive and respond to queries from a **trader**.
3. Your ChatBot must employ some AI so that it exhibits traits consistent with a personalised virtual trading assistant.

UML - Class Diagram

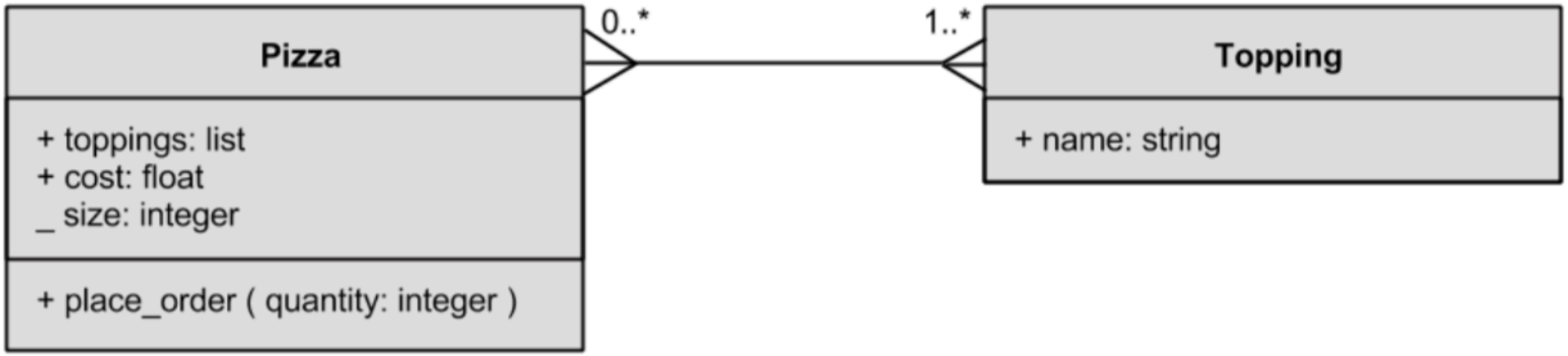
- A Class Diagram in UML shows system classes and their relationships
- UML formal notation moves requirements closer to a mathematical description (which in turn allows us to resolve ambiguities)



- This forces us to think carefully about our language in our D-requirements
- This completes the design of one entity in our system, making it clear to the developer the requirements (the ‘what’, not the ‘how’)

UML - Class Diagram

We can then start to link entities in our system together



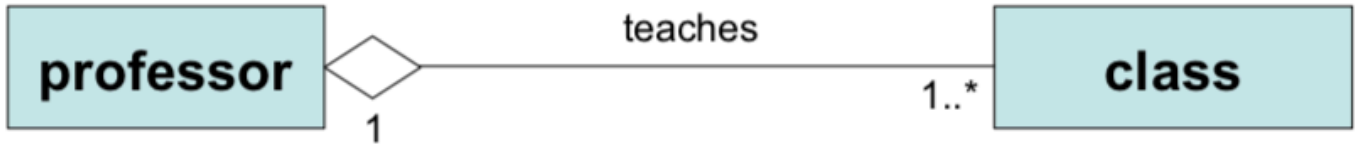
Y has between 0 and 10 X
X has 1 Y

* = no limit

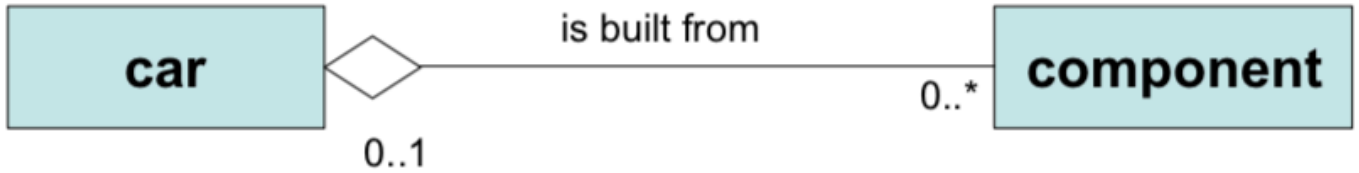
UML - Class Diagram

- **Association**
  - Indicates that two classes are associated in some way

- **Aggregation**
  - A “has a” relationship, more specific than association



- **Composition**
  - An “owns a” relationship, more specific than aggregation
  - If the container is destroyed, all instances that it contains are destroyed along with it



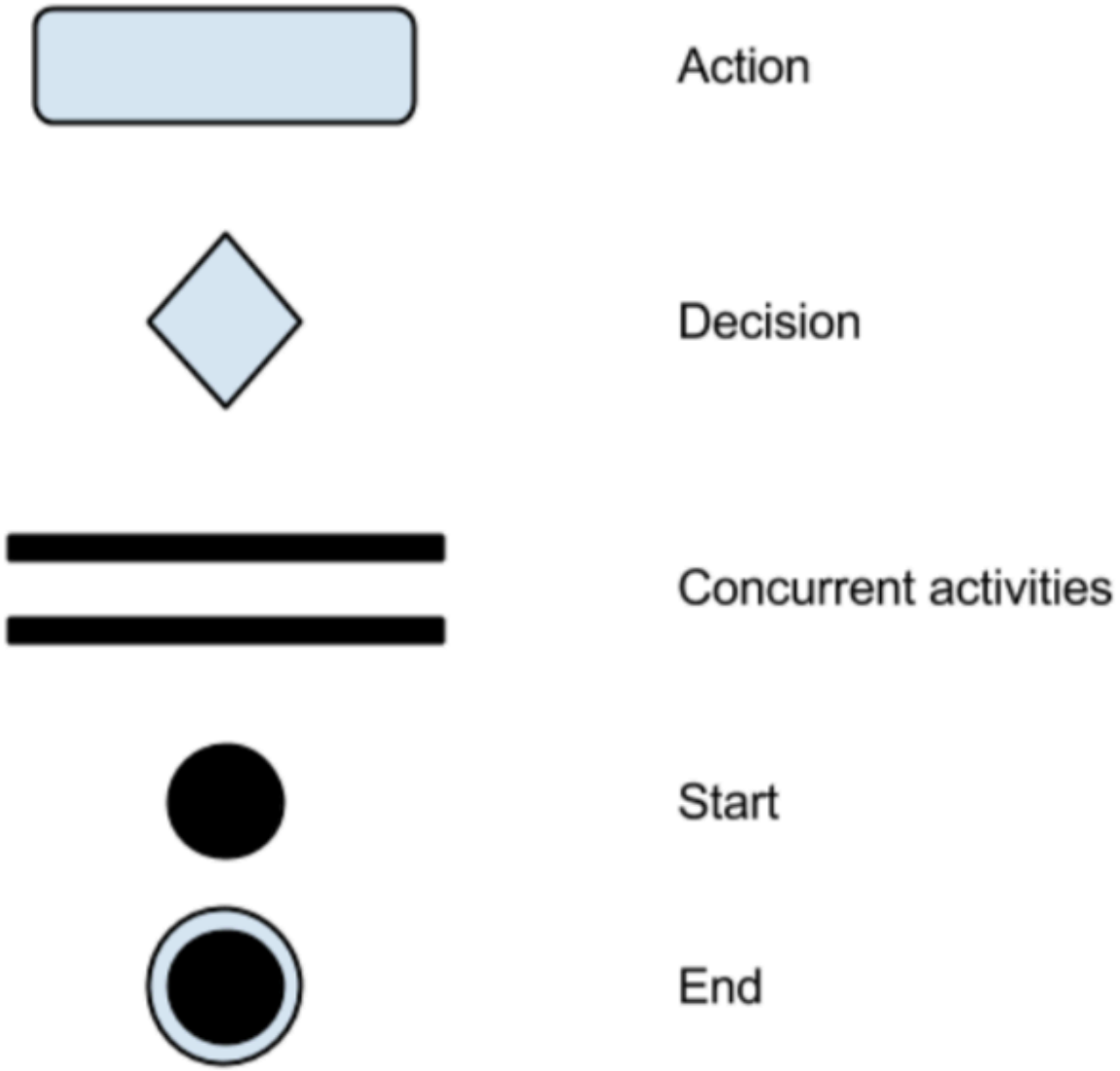
Structural Models

- Structural (static) SE models show the organisation of components within the system
- UML class diagrams are one example of these
- We are studying what we call *Formal Methods*, these are based on foundations which are now familiar to you
  - *logic calculi*
  - *type systems and algebraic data types*
  - *program semantics*
  - *automata theory*

- Static models show the structure of design, while dynamic models (next) show what happens with (and between) these entities as our system operates

UML - Activity Diagram

We need a **formal language** that allows us to **represent workflows** of stepwise activities

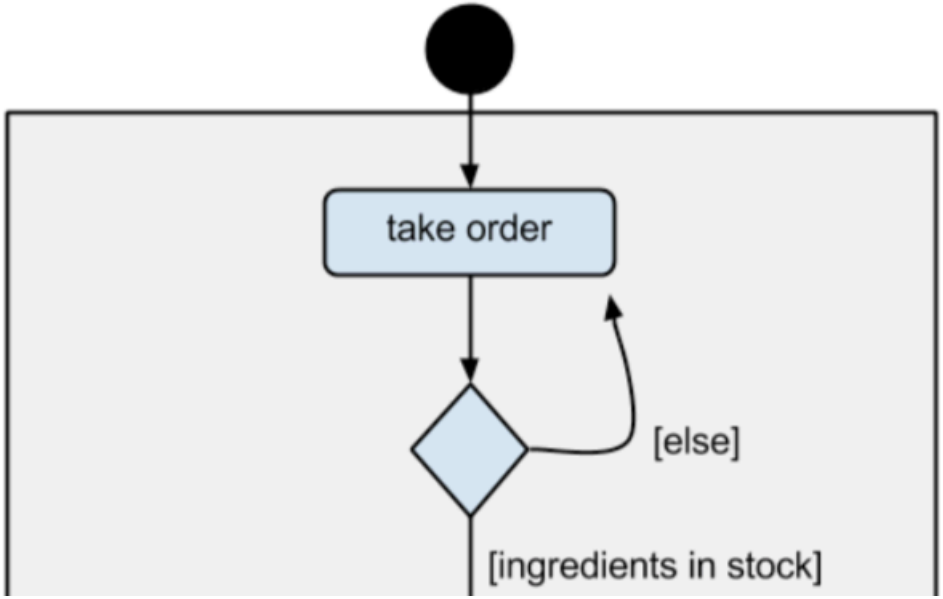


UML - Activity Diagram

- Let’s go back to the pizza example
- We want to order a pizza
- How do we begin?

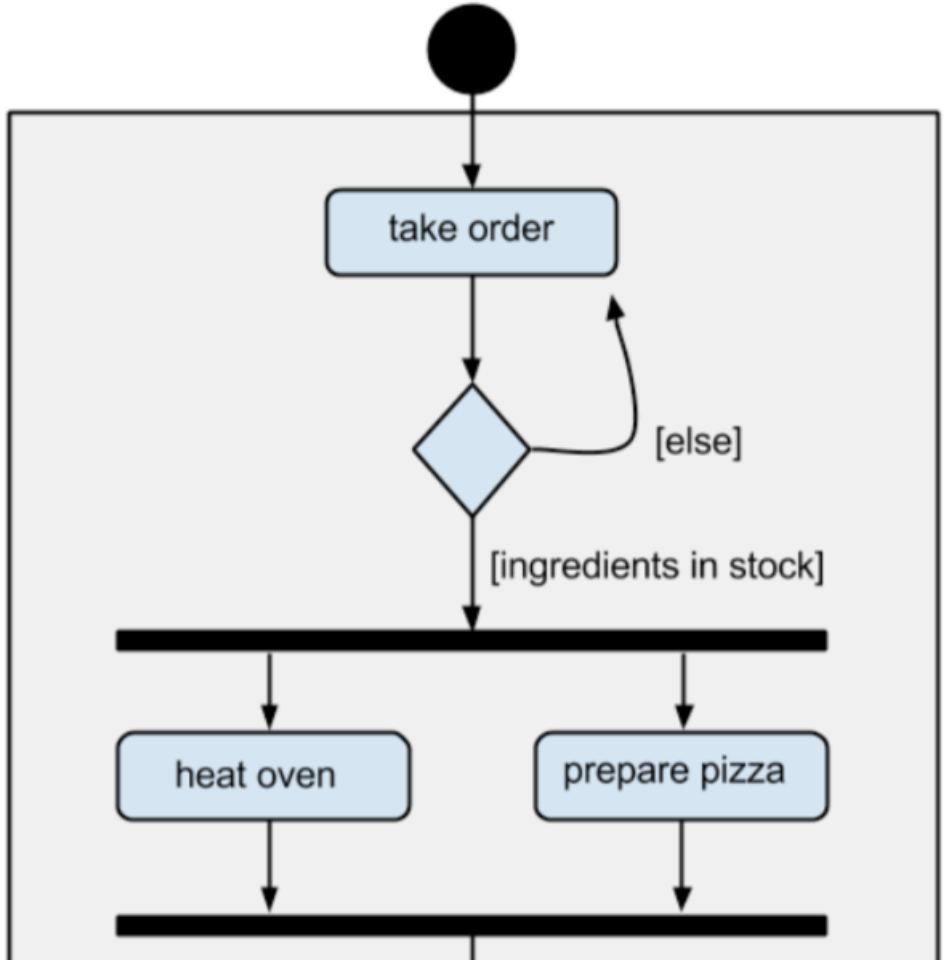
UML - Activity Diagram

- We invoke an action - *take order*
- A decision box is employed



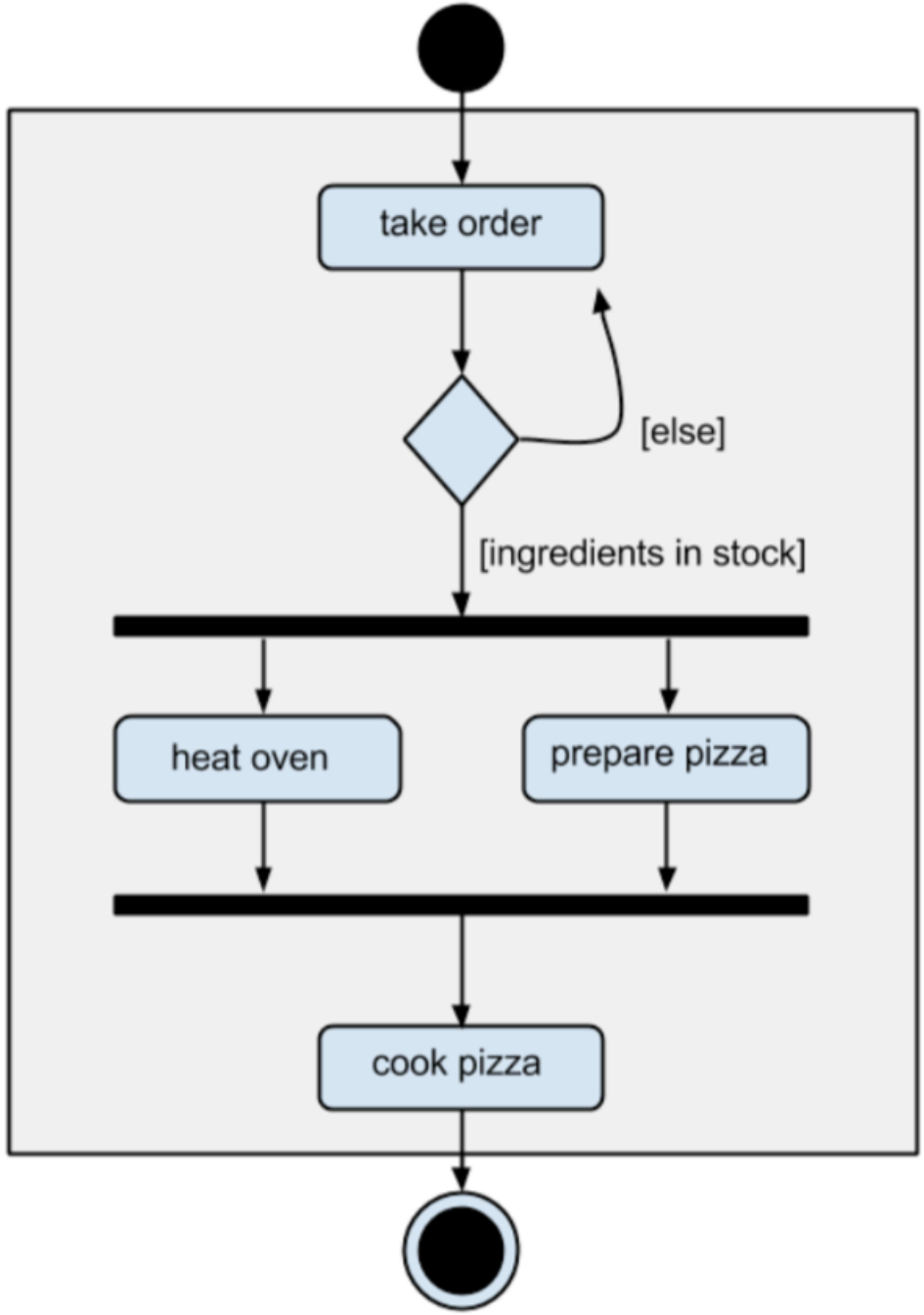
UML - Activity Diagram

- We invoke an action - *take order*
- A decision box is employed
- We introduce some parallelism to speed things up

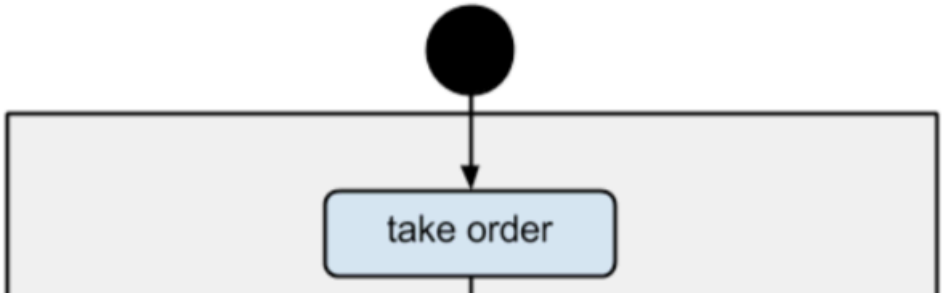


UML - Activity Diagram

- We invoke an action - *take order*
- A decision box is employed
- We introduce some parallelism to speed things up
- A further sequential action is taken - *cook pizza.*



Interaction Models



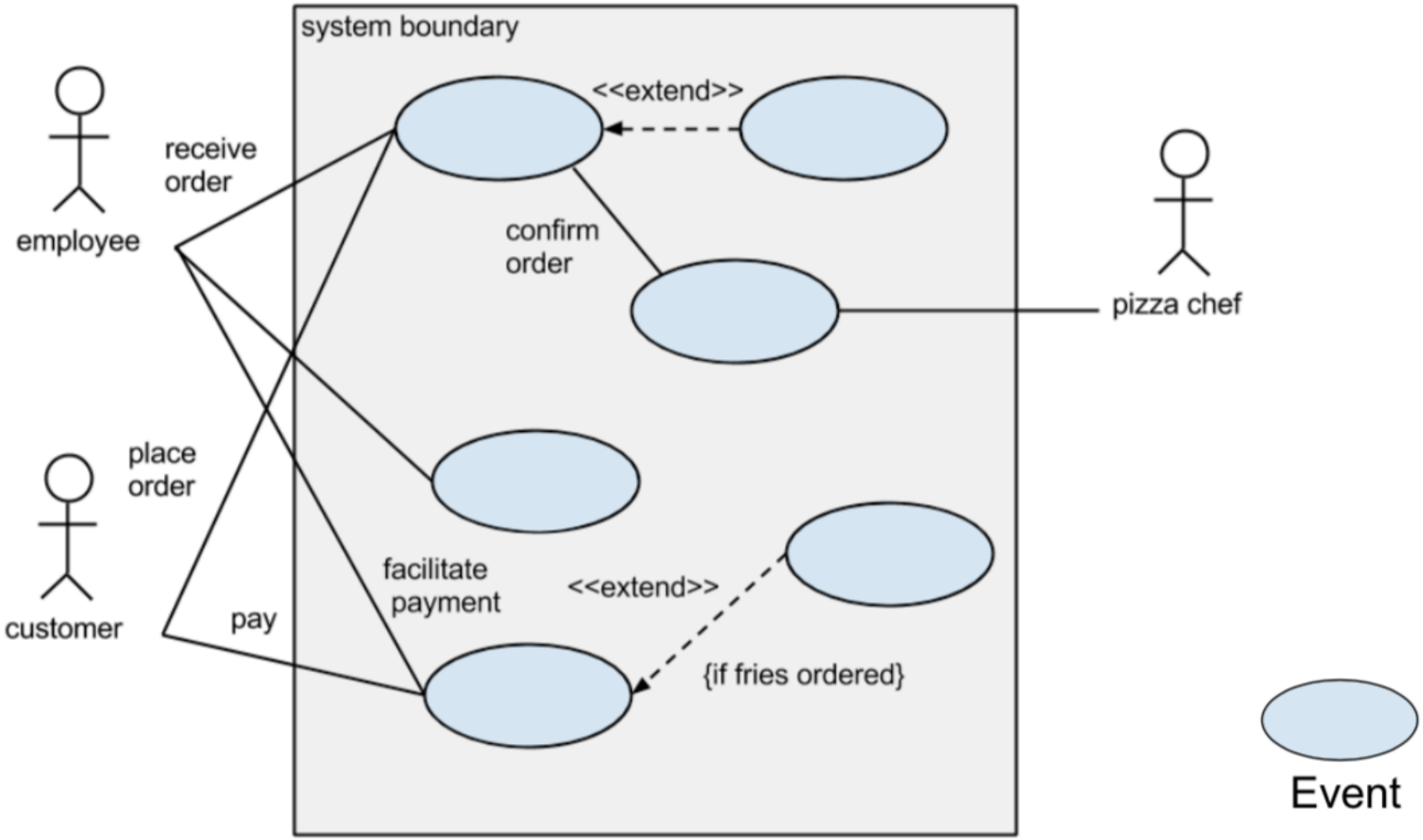
Interaction models show user interaction with the system

In UML these include *Use Case Diagrams* and *Sequence Diagrams*

Shows how **components interact with users** of the system (called actors in UML)

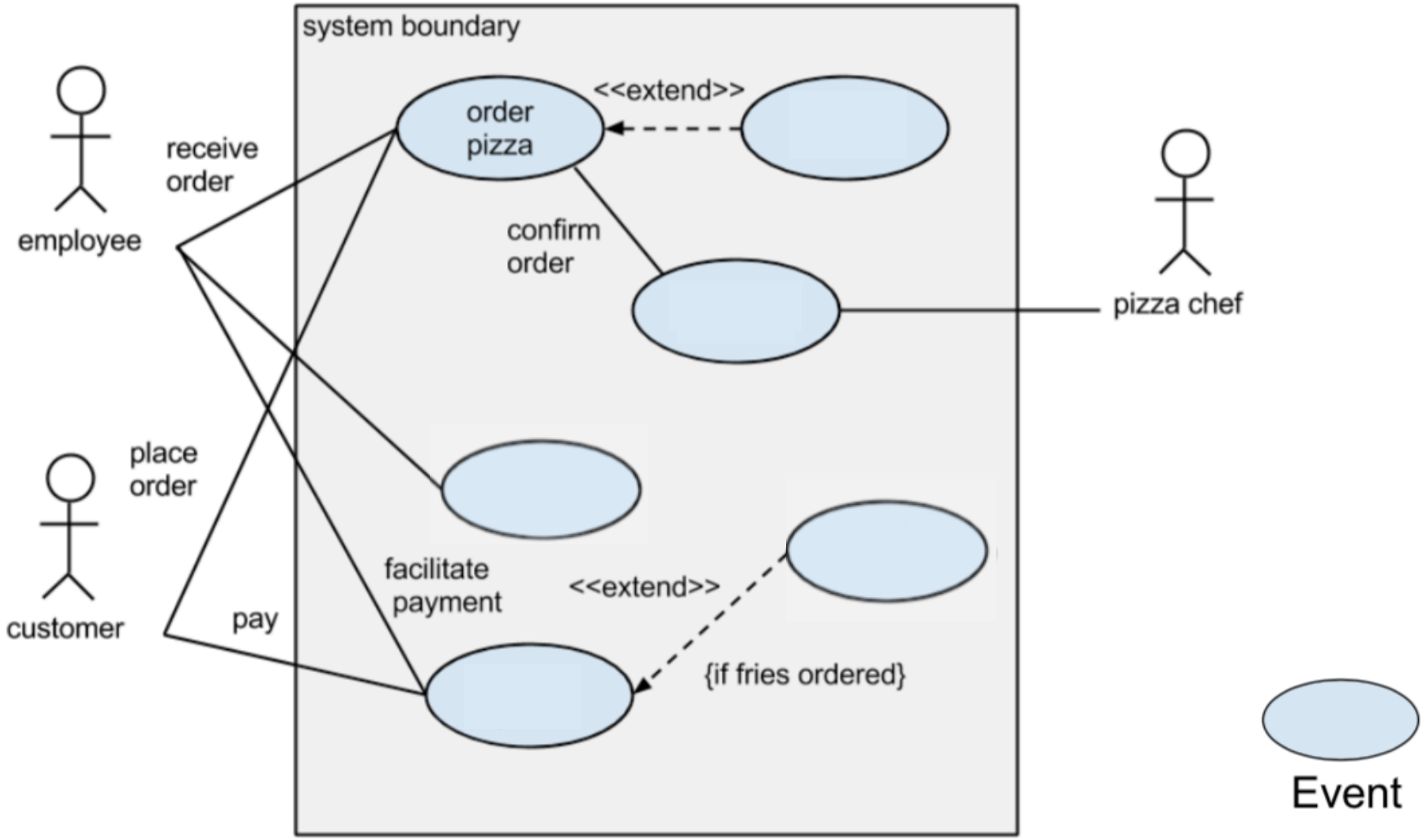
UML - Use Case Diagram

Represents user's interactions with the system



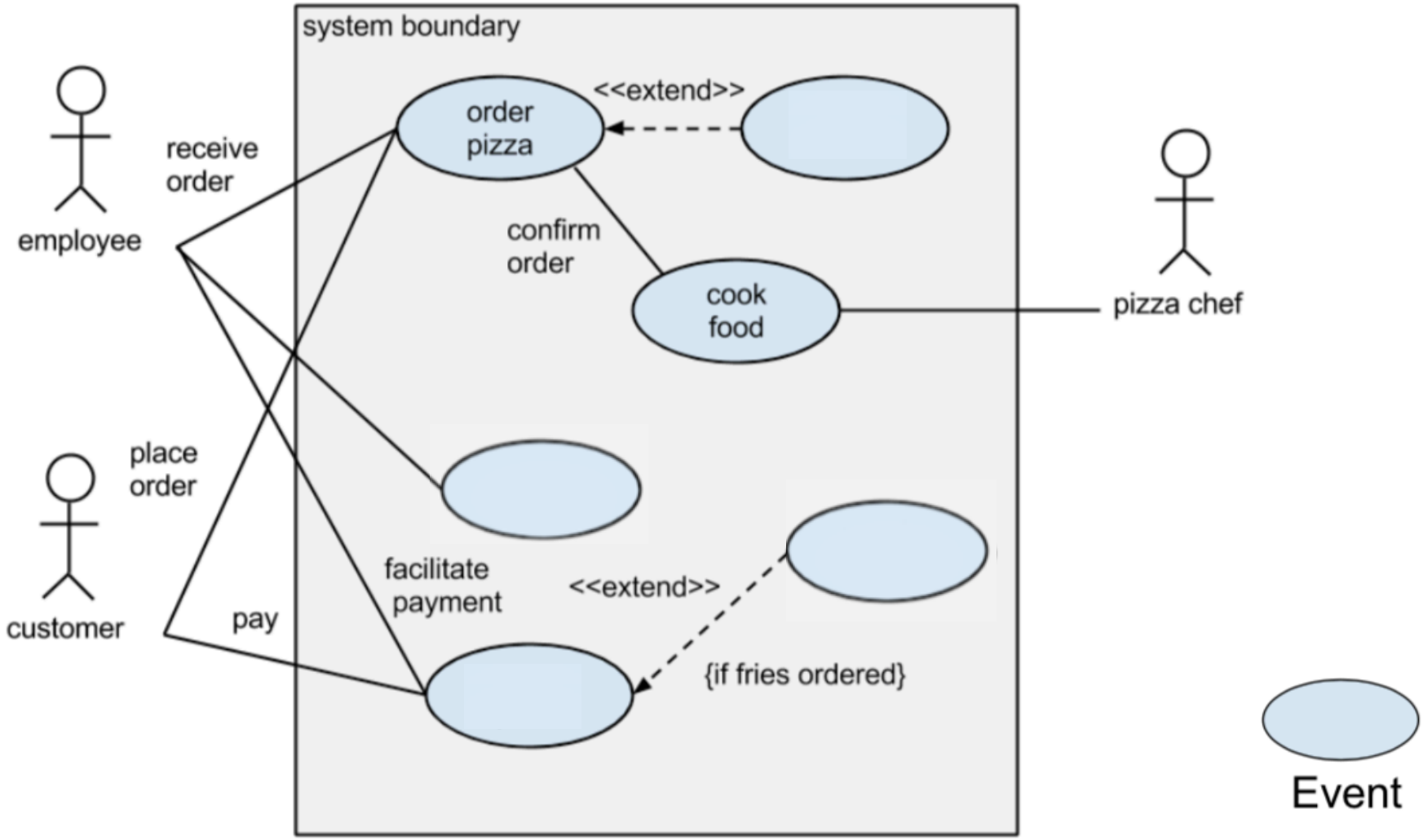
UML - Use Case Diagram

Represents user's interactions with the system



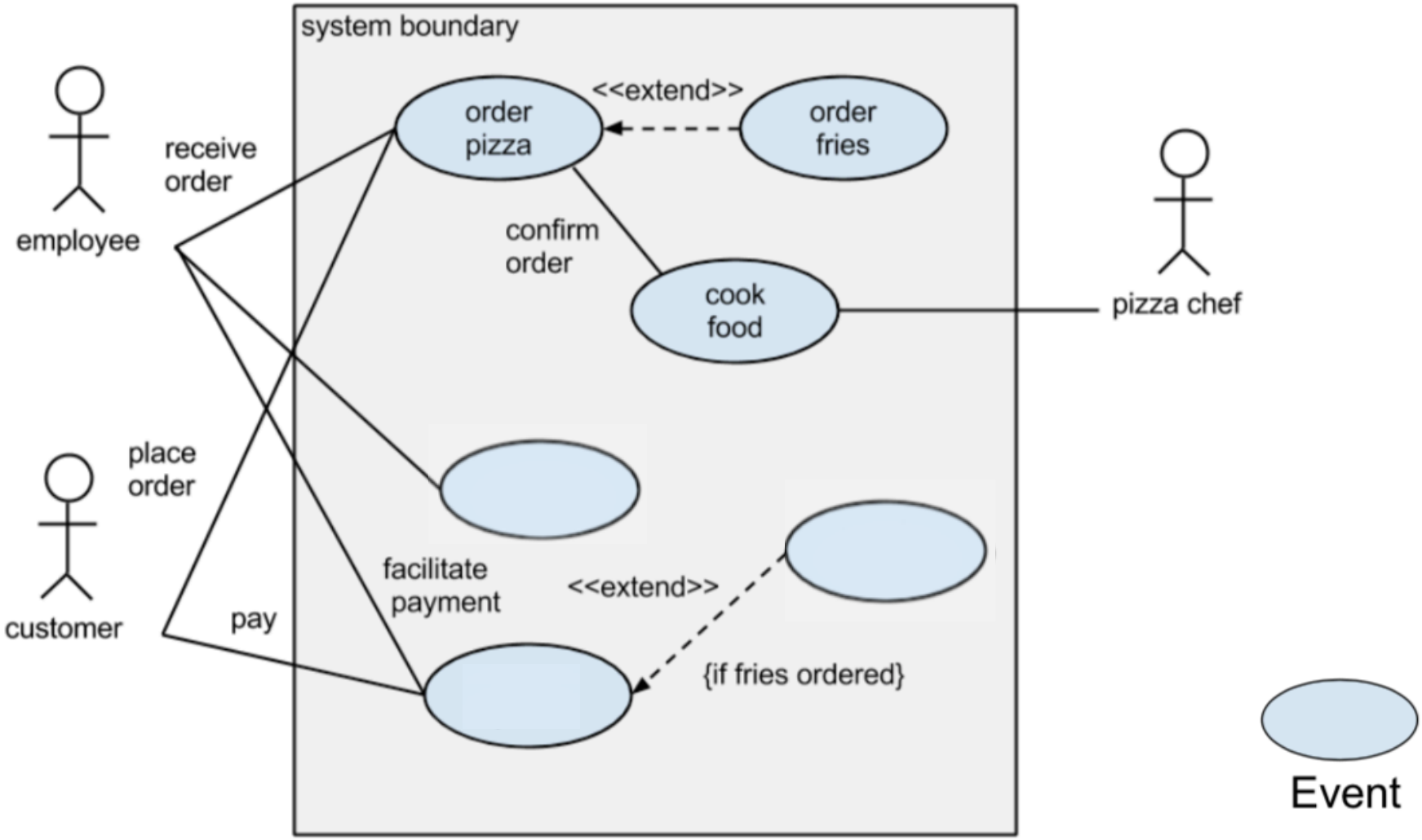
UML - Use Case Diagram

Represents user's interactions with the system



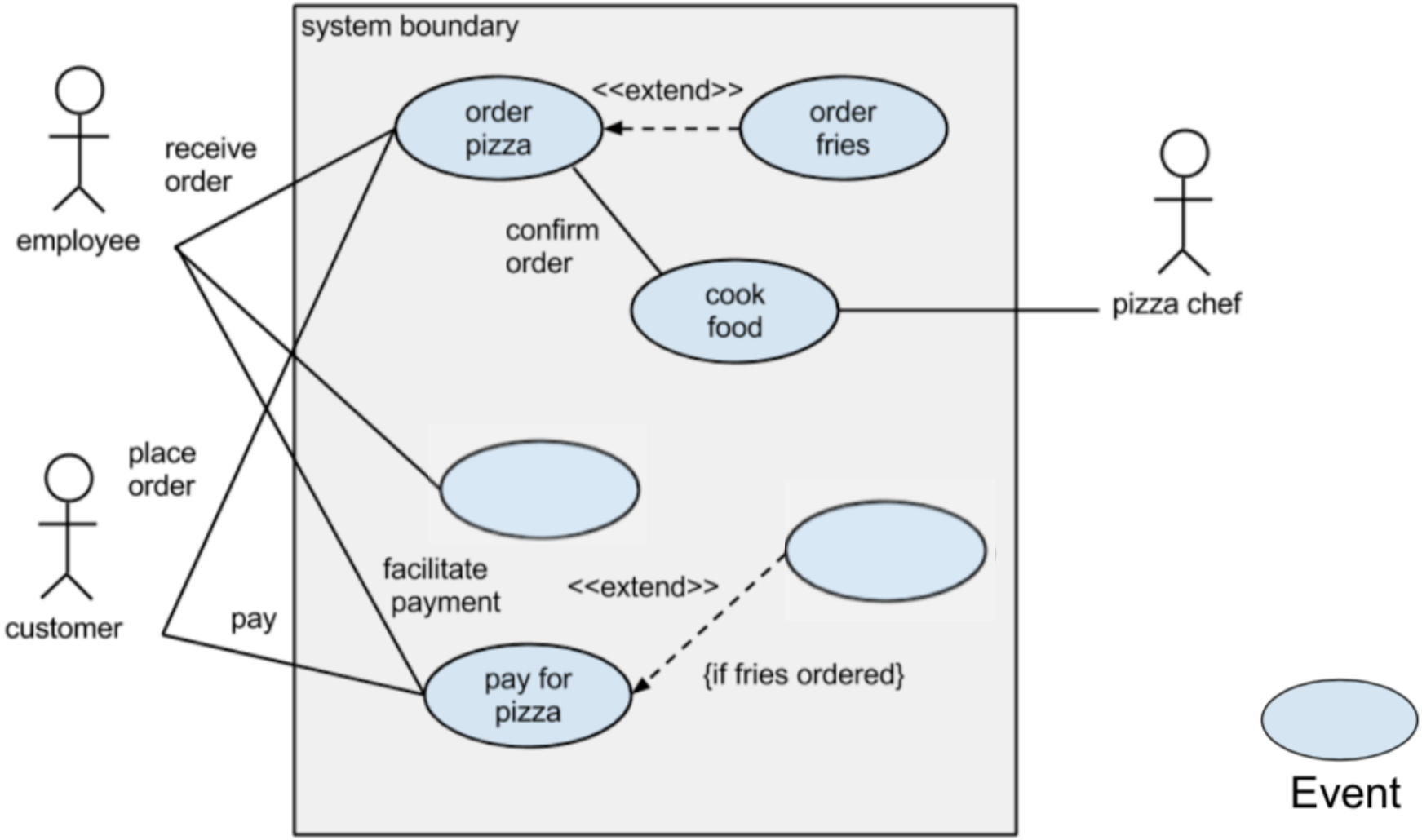
UML - Use Case Diagram

Represents user's interactions with the system



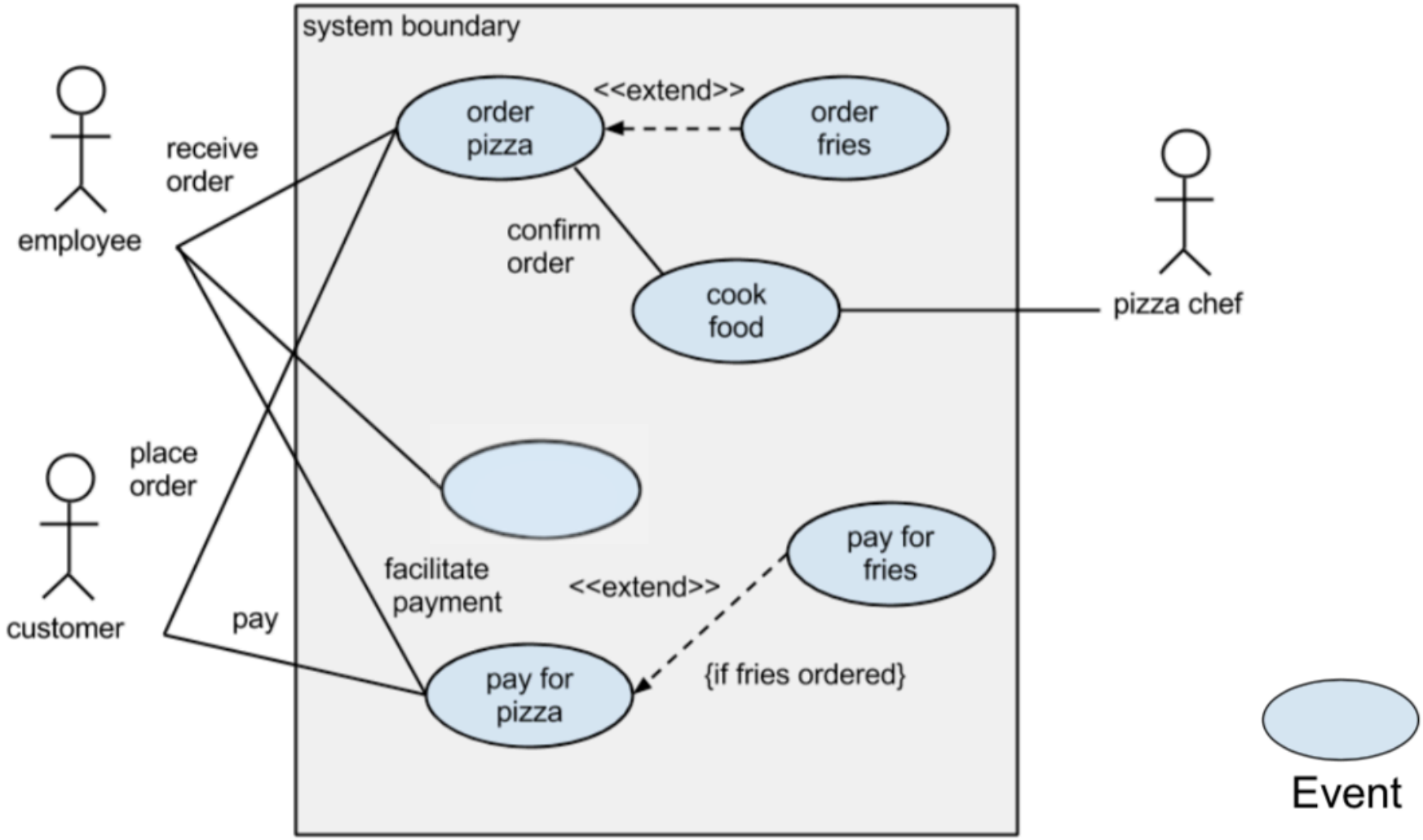
UML - Use Case Diagram

Represents user's interactions with the system



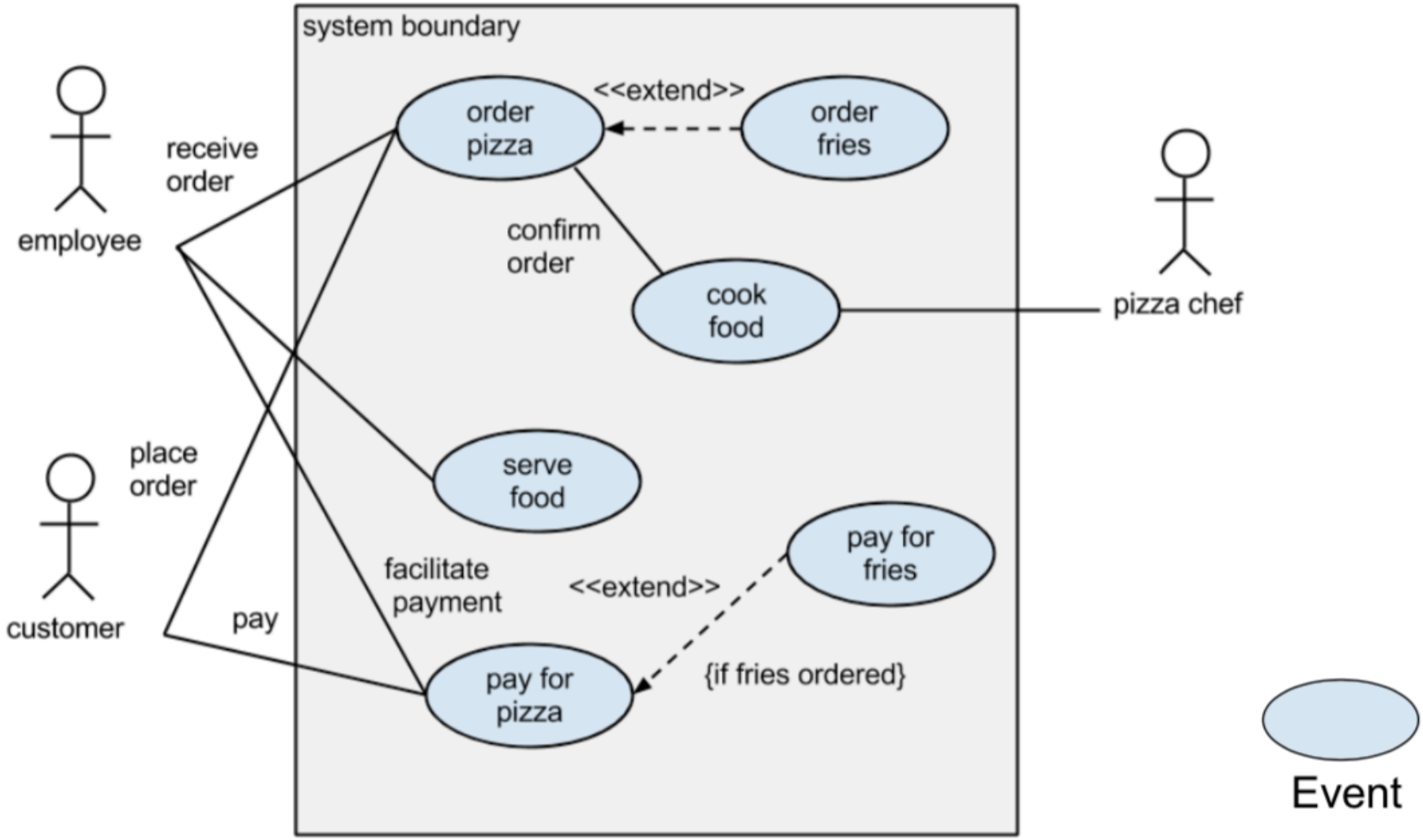
UML - Use Case Diagram

Represents user's interactions with the system



UML - Use Case Diagram

Represents user's interactions with the system

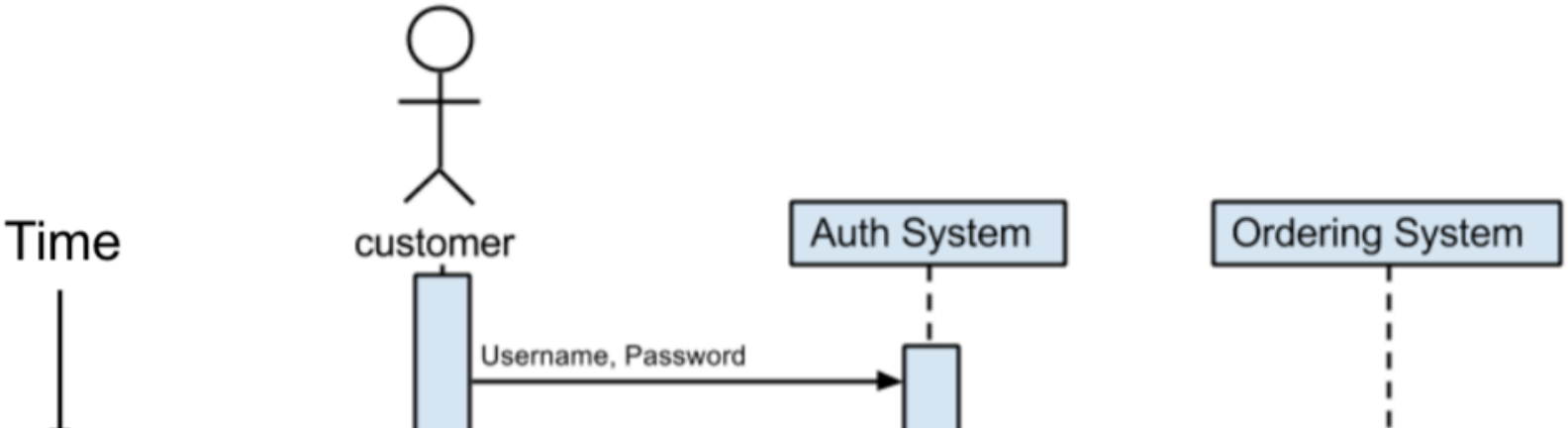


UML - Sequence Diagram

Shows temporal interaction between processes

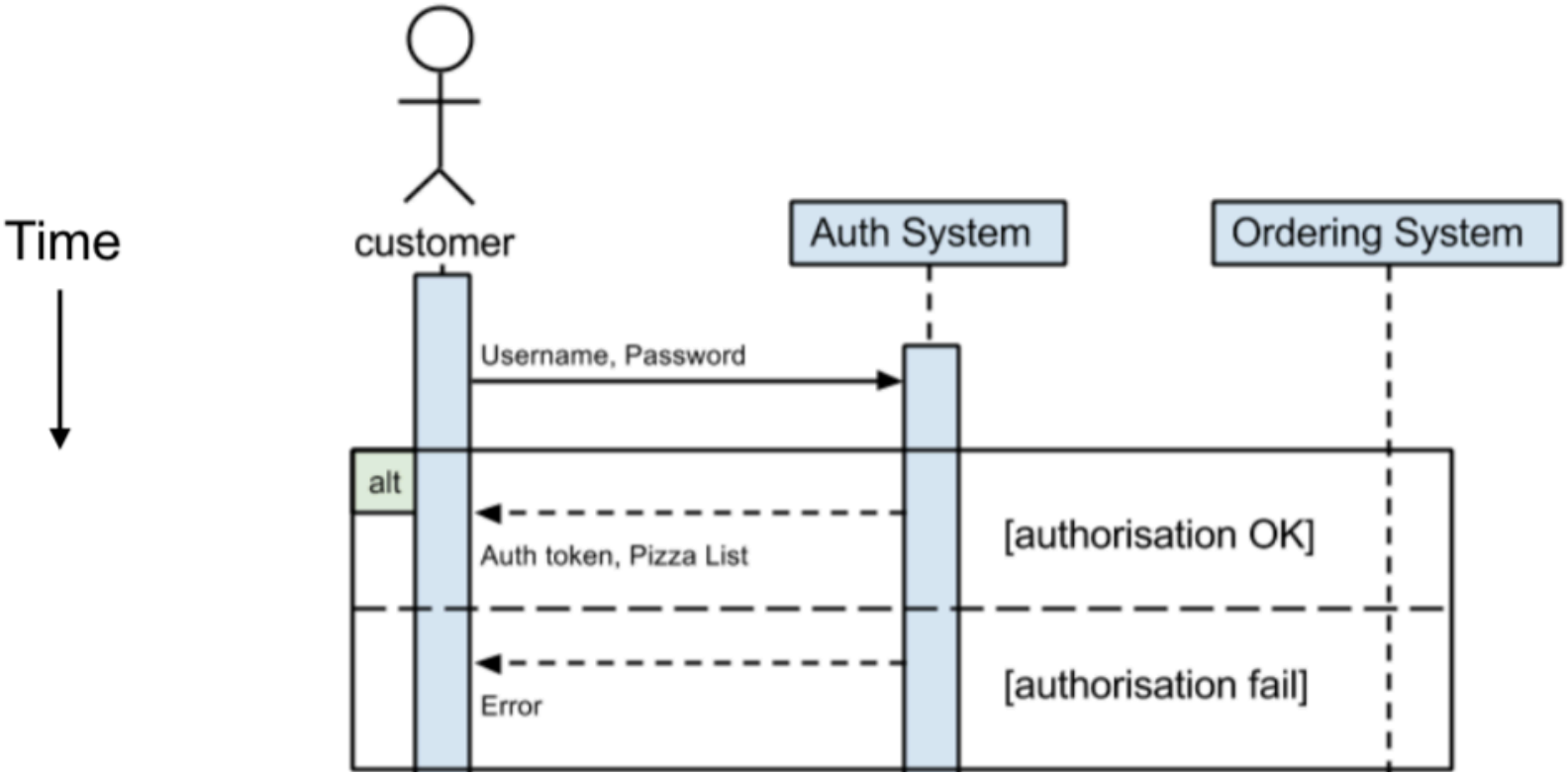
UML - Sequence Diagram

Shows temporal interaction between processes



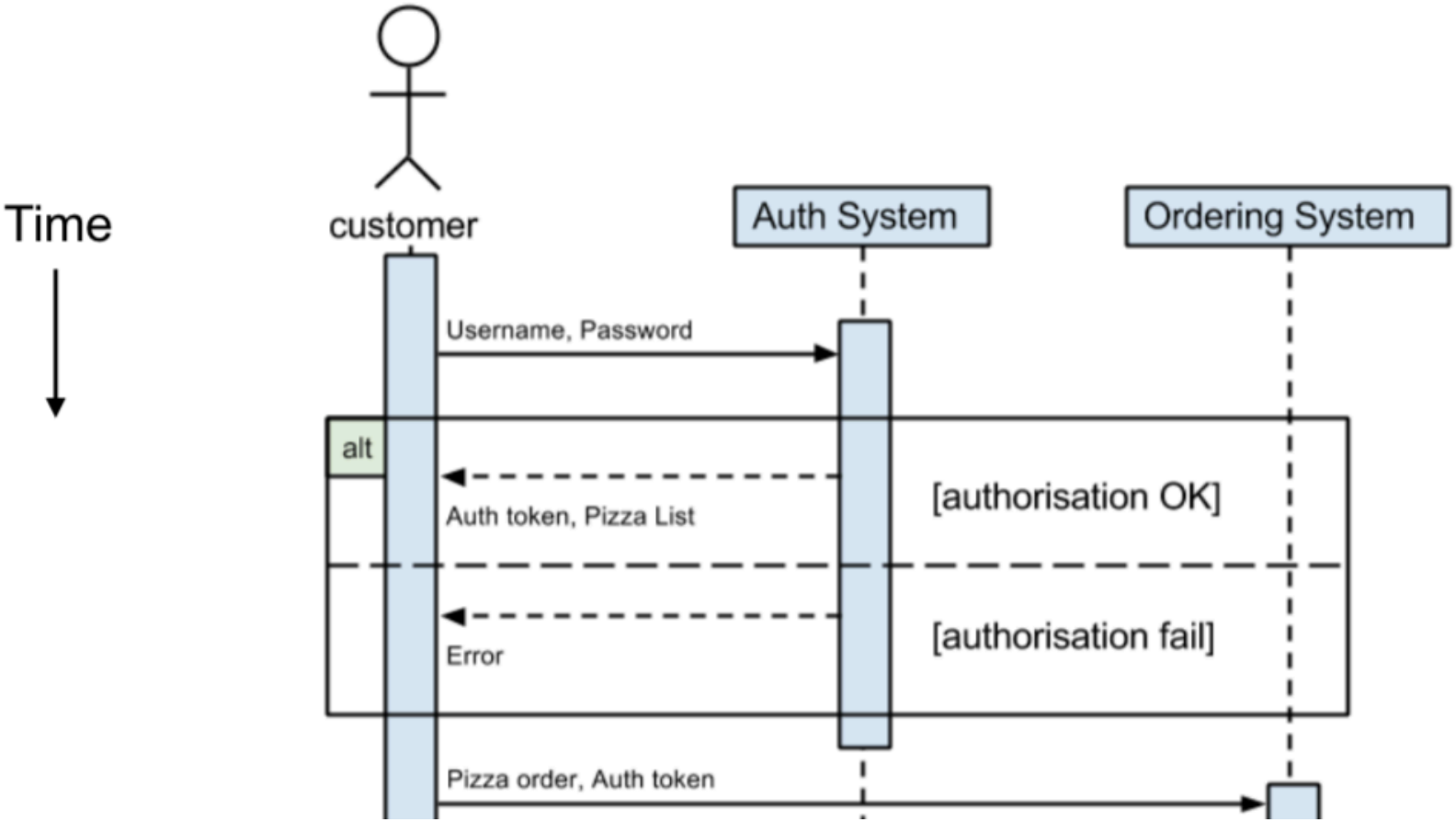
UML - Sequence Diagram

Shows temporal interaction between processes



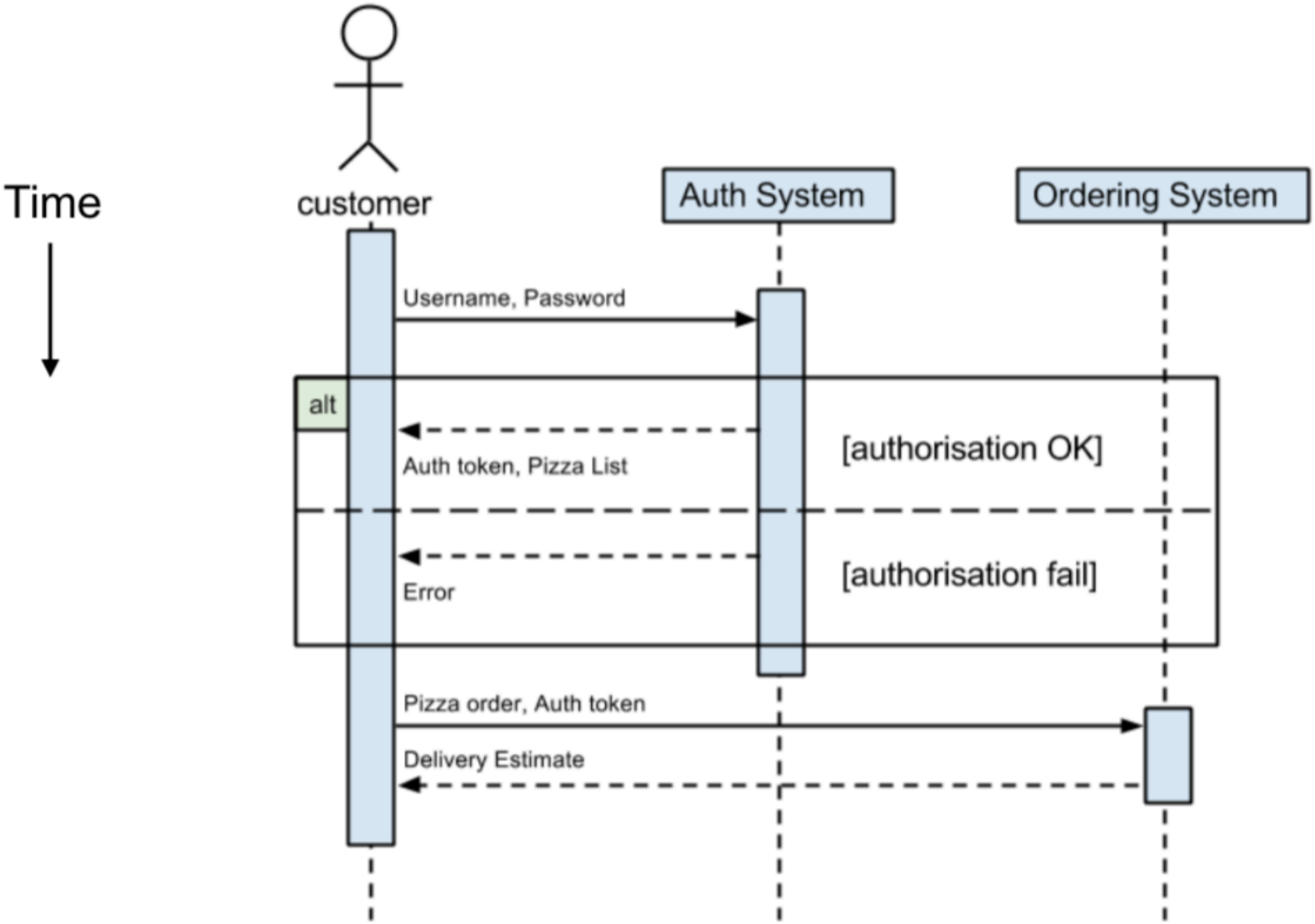
UML - Sequence Diagram

Shows temporal interaction between processes



UML - Sequence Diagram

Shows temporal interaction between processes



Nearly There

- We have gathered C-requirements and mapped these onto D-requirements
- From the D-requirements we need to extract the design
  - We want to extract the entities (*class diagram*)
  - Isolate the interaction between these entities (activity diagram)
  - And also identify how users (use case diagram) might interact with the system (sequence diagram)
- System modelling allows us to capture this behaviour