Introduction to Software Engineering

Lecture 05: System Modeling

System Modeling

- System modeling is the process of developing abstract models of a system.
- Each model presenting a different view or perspective of that system.
- System modeling helps the analyst/developer to understand the functionality of the system
- System models are used by developers to discuss possible design and implementation options
- In addition, some models are used to communicate with customers during requirement discovery.

System Perspectives/Views

- An external perspective, where you model the context or environment of the system.
- An interaction perspective, where you model the interactions between a system and its environment, or between the components of a system.
- A structural perspective, where you model the organization of a system or the structure of the data that is processed by the system.
- A behavioral perspective, where you model the dynamic behavior of the system and how it responds to events.

How are software models/architectures described?

Informal description: using natural language and informal notations. In this case the architecture can not formally analyzed (e.g. sketches).

Semi-formal description: using standardized notations and well defined modeling languages. However, the semantic of these notation is not formally defined (e.g. UML, OMT).

Formal description: using formal notations with a precise semantic based on mathematical foundation. (e.g. OCL, Architecture Description Language ADL)

What are the pros and cons of the above methods?

What is UML?

- UML or Unified Modeling Language is a standard modeling language for software systems.
- We can use UML to model:
  - System Requirements
  - System Logical Structure
  - System Workflows
  - System Interactions
- UML diagrams are divided into two categories: Structure diagrams and Behavioral diagrams

UML History

- Many techniques have been proposed to model software design and architecture using diagrams. For example, **Object-Oriented Analysis and Design** (OOAD) and **Object-Modeling Technique** (OMT), **Object-Oriented Software Engineering** (OOSE)
- UML has three architects **Rumbaugh (OMT)**, **Booch (OOAD)**, and **Jacobson (OOSE)**.
- UML was adopted as a standard by the **Object Management Group** (OMG) in **1997**
- UML version 2.0 was released in 2005 and it has 14 different diagrams that we can use to model our design.
- In **2015** UML version **2.5** was released

UML Structure Diagrams

Describe the static architecture of the system. They describe the key building blocks (classes, objects, components) of the model and the relations between these components (dependencies, composition, association, etc)

1. Class Diagram: model system classes, interfaces, types and the relations between them.
2. Object Diagram: describe how the objects within the system would look in a particular scenario.
3. Composite Structure Diagram: model the internal structure of the system classes.
4. Component Diagram: model the key components in the system and the interfaces they used to interact with each other

UML Structure Diagrams

5. Package Diagram: model the hierarchical structure the system classes and components. Group cohesive classes and components and the show the dependencies among packages.
6. Deployment Diagram: model the physical view of the system as collection of software and hardware resources and how they are linked together.
7. Profile Diagram: describe an extension to the UML by defining meta-model, constraints, stereotypes, and tagged values

UML Behavioral Diagrams

Describe the dynamic aspects of the system architecture. They describe the key functions/behaviors of the system during run-time/execution.

1. Use Case Diagram: model interactions between your system and users or other external systems.
2. Activity Diagram: model Sequential and parallel activities within your system.
3. Sequence Diagram: model Interactions between objects. Mainly useful when the order of the interactions is important.
4. State Machine Diagram: model the states of an object during run time and the events that affect these states.

UML Behavioral Diagrams

5. Communication Diagram: model the communications/connections between the objects that are needed to support interactions
6. Timing Diagrams: model the interaction between objects and how these interaction affect the object state. It fuses sequence and and state diagrams
7. Interaction Overview: provide a high-level view of how several interactions work together to implement a system function/behavior. It combines both the activity and the sequence diagram

What is Object Constraint Language (OCL)?

- Developed **by IBM in 1995** as a formal specification extension to UML and adopted by OMG as part of the UML version 1.1
- Proposed to overcome the limitation of UML visual models (**lack of precise semantics**)
- Mainly used to **define constraints on UML models**. But it can be used to queries UML models.
- OCL focuses on **applying constraints on UML Class diagrams** Several tools can extract the OCL constraints from the UML model to perform **constraint checking.**

Object Constraint Language

- OCL is a language it has both well-defined **syntax and semantic**, built -in types and operators.
- OCL is side effect free
- There are three types of constraint that can be applied to UML class using OCL:
  - Invariant: a constraint that must always be true. Defined on class attributes.
  - Precondition: a constraint that is defined on a class method. Usually used to validate input parameters.
  - Postcondition: a constraint that is defined on a class method. Checked after the method executes. Usually used to validate how values were changed by a method.

OCL Common Operators

Group	Operators	Used with types	Example OCL expression
Arithmetic	`+, -, *, /`	Integer, Real	baseCost + tax
Additional Arithmetic	`abs(), max(), min()`	Integer, Real	score1.max(score2)
Comparison	`<, <=, >, >=`	Integer, Real	rate > .75
Equality	`=, <>`	All	age = 65 title <> 'CEO'
Boolean	`and, or, xor, not`	Boolean	isMale and (age >= 65)
String	`concat(), size(), substring(), toInteger(), toReal()`	String	title.substring(1,3)

Figure 1: common operators in OCL expressions

OCL Syntax Structure

Invariant

context < *classifier* > inv [< *constraint name* >] : < *Boolean OCL expression* >

Precondition

context < *classifier* >::< *operation* > (< *parameters* >) pre [< *constraint name* >] : < *Boolean OCL expression* >

Postcondition

context < *classifier* >::< *operation* > (< *parameters* >) post [< *constraint name* >] : < *Boolean OCL expression* >

OCL Examples

Invariant

context *Account* inv *balanceConstraint* : *self*.*blance* >= 0

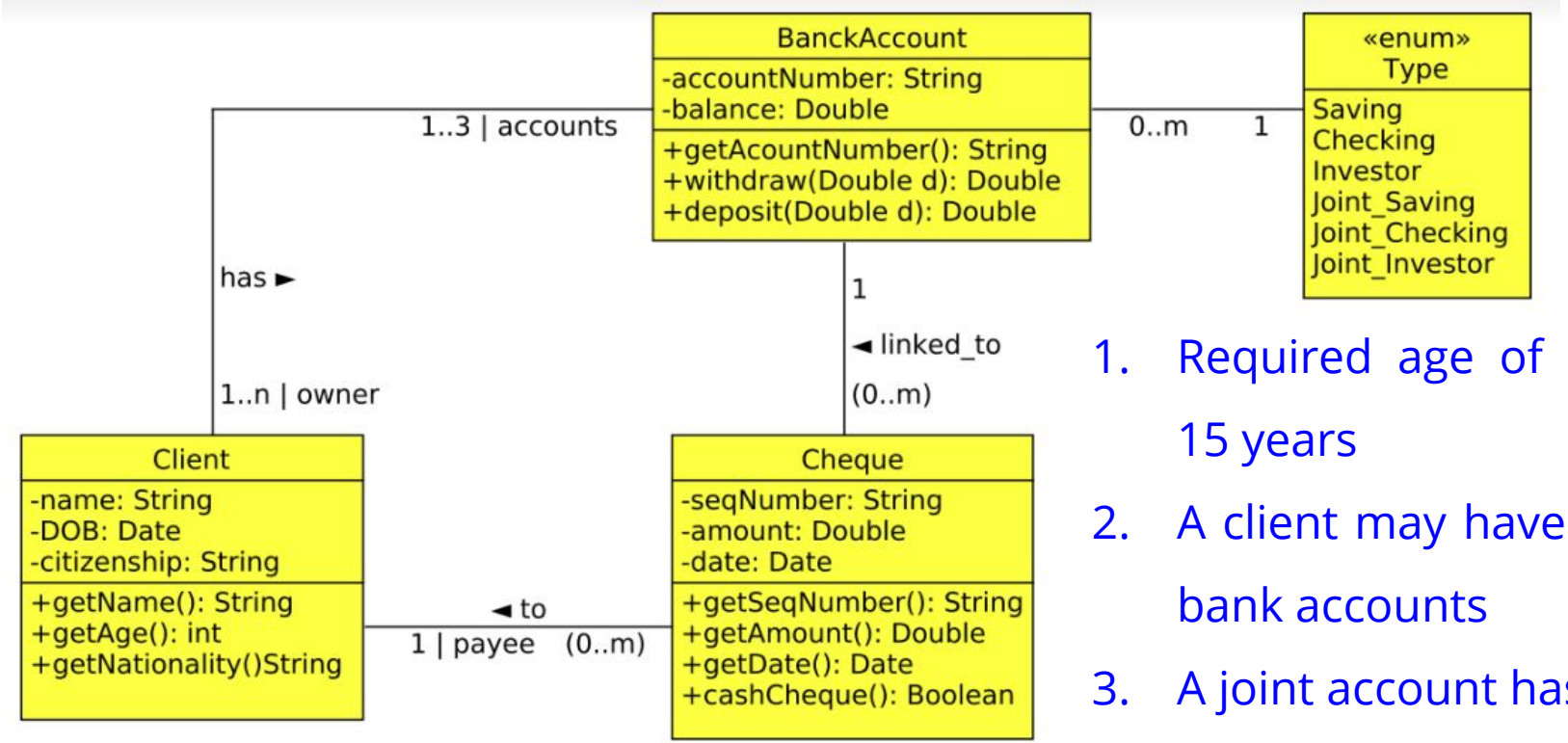
Precondition

context *Meeting* :: *shift*(*days* : *Integer*) pre *shifMeetingRule*
self.*date* <> *self*.*Date*.*today*

Postcondition

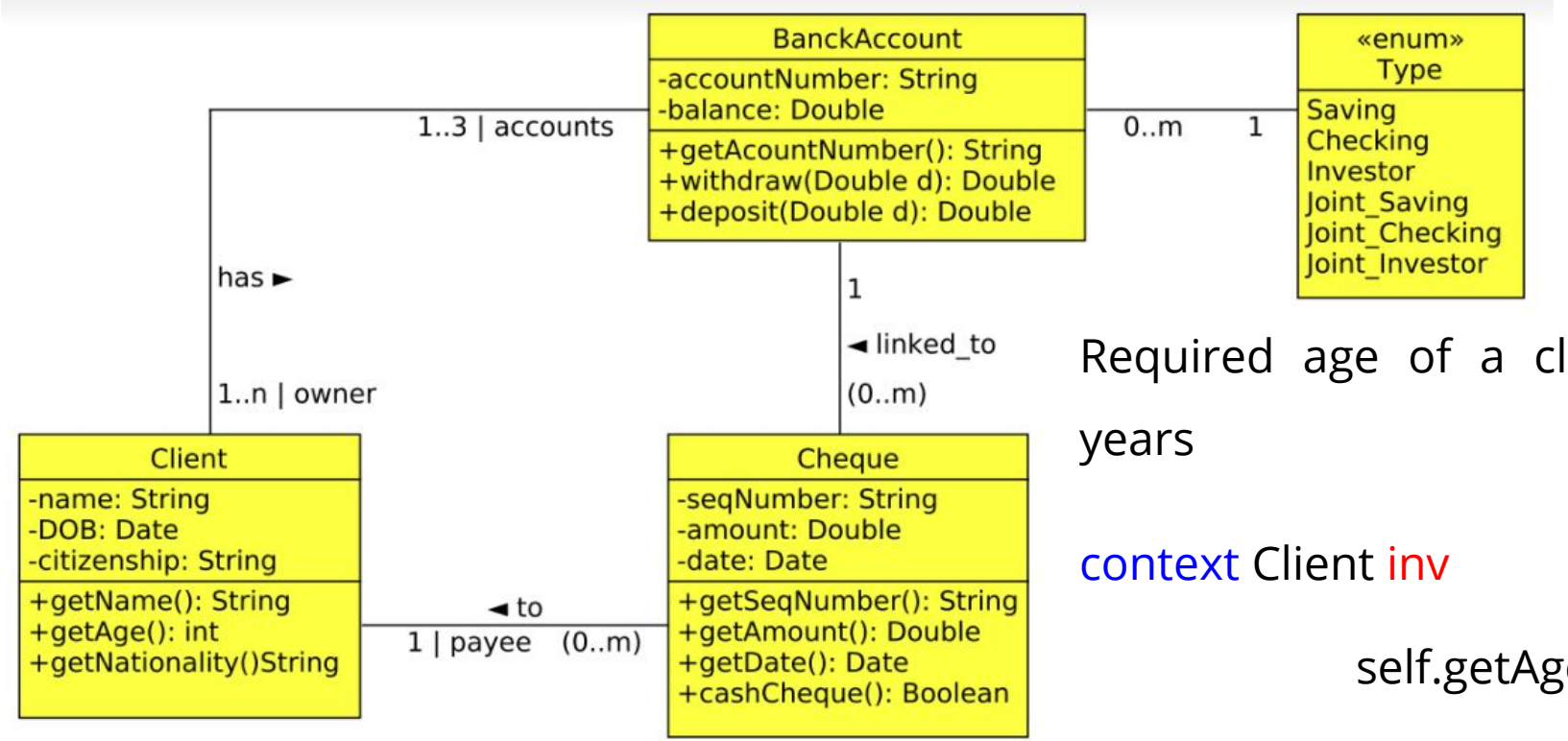
context *Meeting* :: *duration*(*days* : *Integer*) post *checkDuration* result =
self.*endTime* − *self*.*startTime*

OCL Invariant Example



1. Required age of a client is 15 years
2. A client may have at most 3 bank accounts
3. A joint account has at least 2 owners

OCL Invariant Example

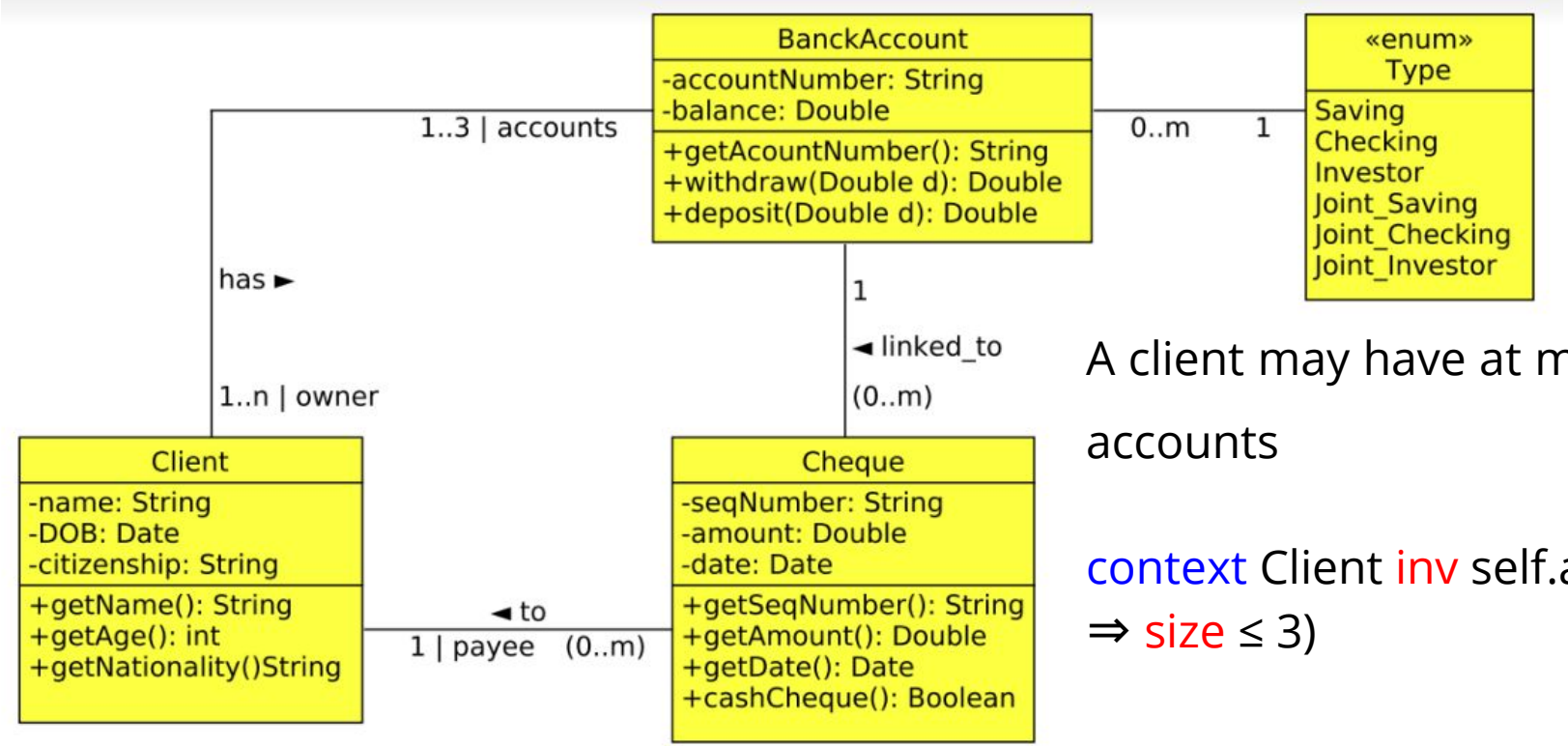


Required age of a client is 15 years

context Client inv

self.getAge() ≥ 15

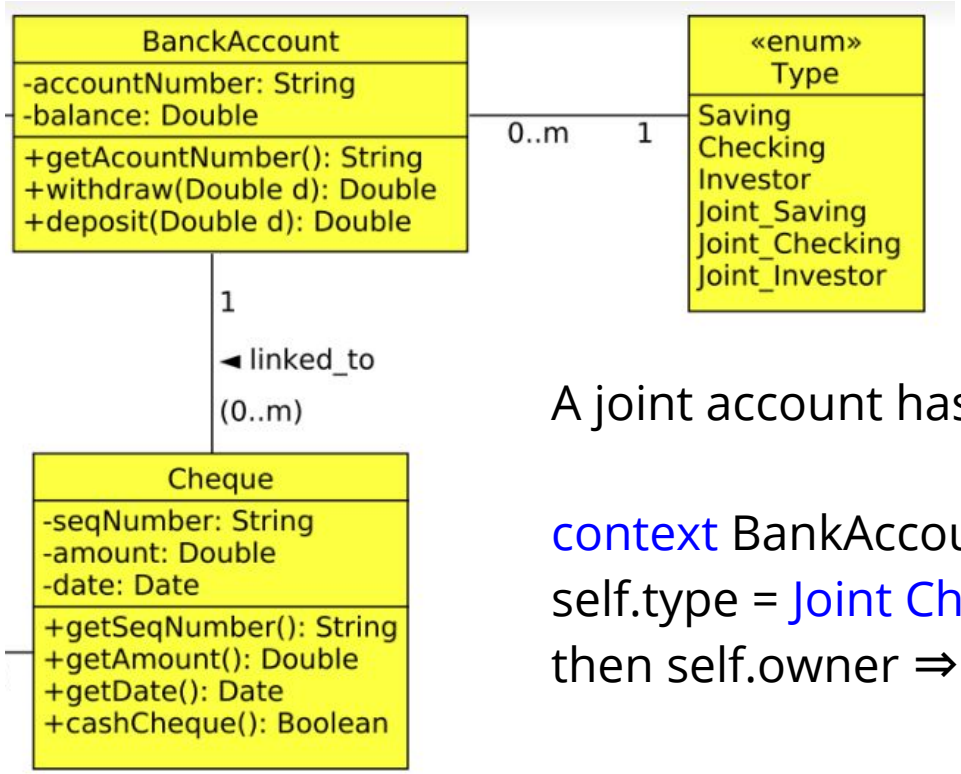
OCL Invariant Example



A client may have at most 3 bank accounts

context Client inv self.accounts ⇒ size ≤ 3)

OCL Invariant Example



A joint account has at least 2 owners

context BankAccount inv if (self.type = Joint_ Saving or self.type = Joint Checking or self.type =Joint Investor then self.owner ⇒ size ≥ 2)

Context Models

Context Models

- Context models are used to illustrate the operational context of a system
- They show what lies outside the system boundaries.
- Context models simply show the other systems in the environment, not how the system being developed is used in that environment.
- Context models focus on defining system boundaries
- System boundaries are established to define **what is inside and what is outside the system**.
- The position of the system boundary has a **profound effect on the system requirements**.

Context Diagram

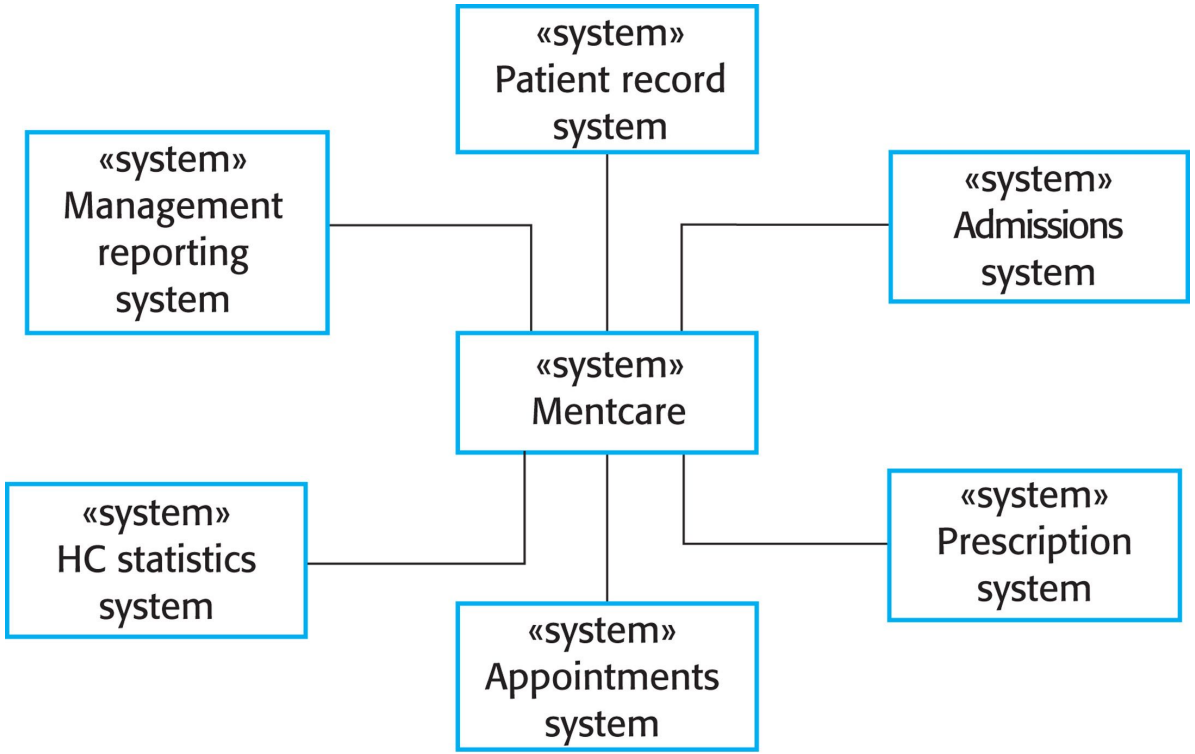
- A high-level diagram that sets the scene of your system including key system dependencies and actors.
- Defines the boundary between the system, or part of a system, and its environment, showing the entities that interact with it.
- Represent all external entities that may interact with a system.
- The objective of the context diagram is to focus attention on external factors and events that should be considered while developing your system.

Context Diagram

By looking at the context diagram we should be able to answer the following questions:

- What is the software system that we are building?
- Who is using it?
- How does it fit in with the existing IT environment?

Context Diagram



Context Diagram

The context diagram consist of:

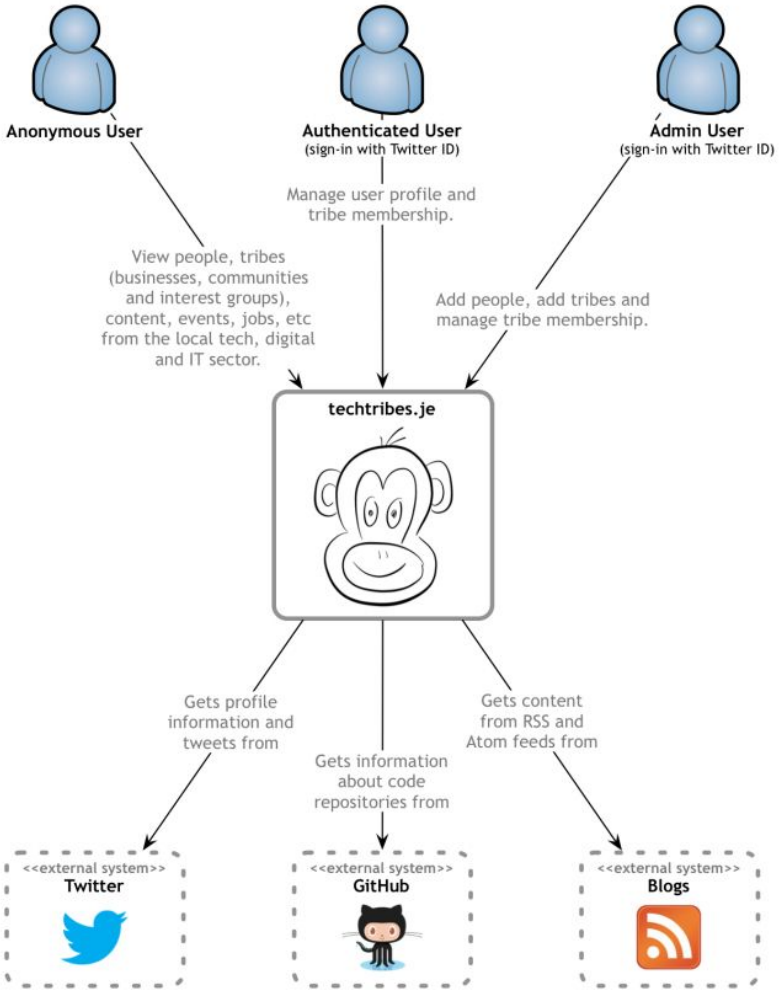
1. One box in the center representing the your system with no details of its interior structure.
2. Users, actors, roles, or personas
3. External Entities: other IT systems that interact with your system
4. Relationships: labelled lines between the entities and the system describe the Interactions between them.

TechTribes: Example

The techtribes website provides a way to find people, tribes (businesses, communities, interest groups, etc) and content related to the tech, IT and digital sector. At the most basic level, it’s a content aggregator for local tweets, news, blog posts, events, talks, jobs and more

Context Diagram

Compare this context diagram with the previous one. Which one is more informative?



Structural Model

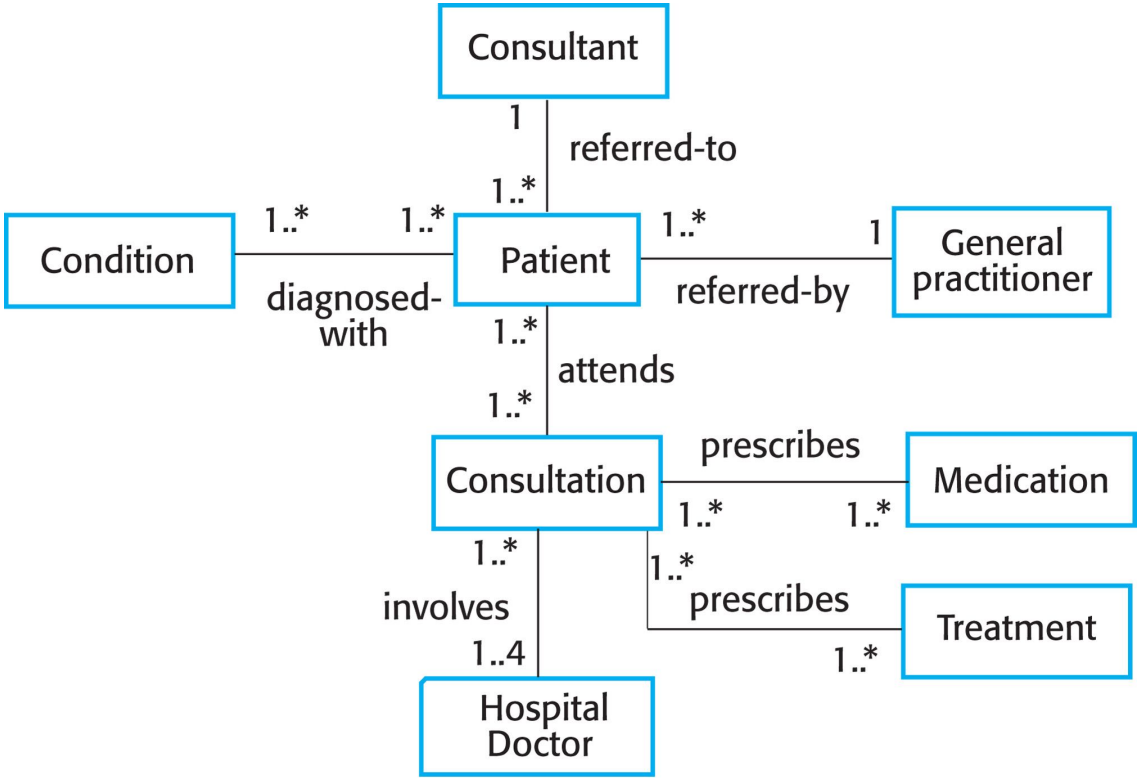
Structure Models

- Structural models of software display the organization of a system regarding the components that make up that system and their relationships.
- You create structural models of a system when you are discussing and designing the system architecture.
- Class Diagram, Component Diagram, and Containers Diagrams are the most common diagrams used for structure models.

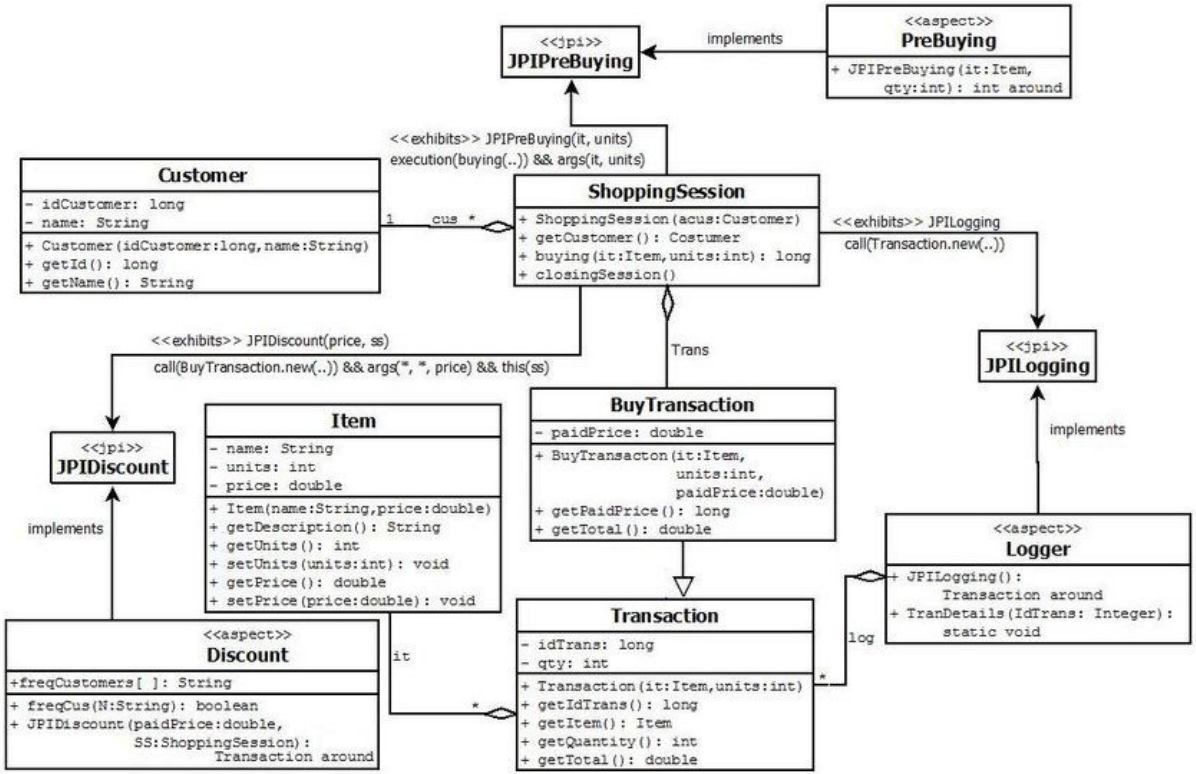
Class Diagram

- Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
- An object class can be thought of as a general definition of one kind of system object.
- An association is a link between classes that indicates that there is some relationship between these classes.
- When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.

Class Diagram Example 1



Class Diagram Example 2



Component Diagram

- Identify the **major logical components** and their interactions.
- Partitioning the functionality implemented by a software system into a number of distinct components, services, and subsystems.
- A component diagram helps you answer the following questions
  - What components/services is the system made up of?
  - It is clear how the system works at a high-level?

Component Diagram

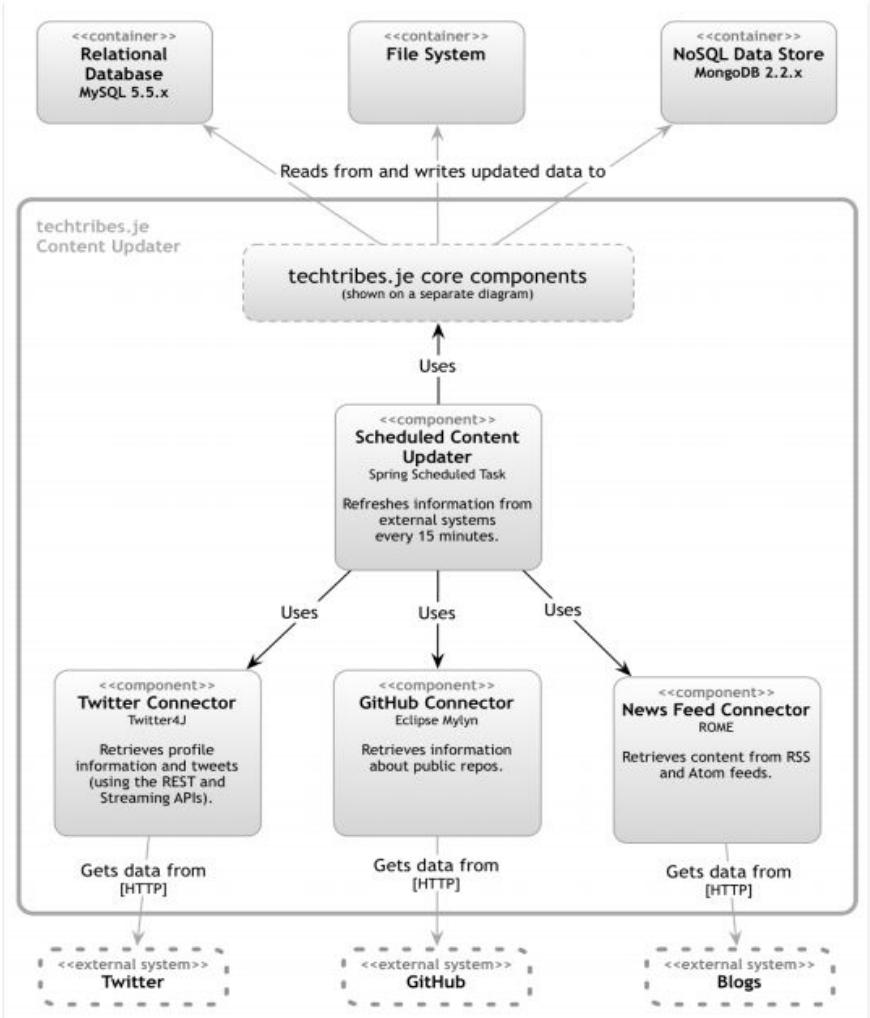
For each of the components drawn on the diagram, you could specify:

- Name: The name of the component (e.g. “Risk calculator”, “Audit component”, etc).
- Technology: The technology choice for the component (e.g. Plain Old [Java|C#|Ruby|etc] Object, Enterprise JavaBean, Windows Communication Foundation service, etc).
- Responsibilities: A very high-level statement of the component’s responsibilities (e.g. either important operation names or a brief sentence describing the responsibilities)

Component Diagram

It is important to describe the interactions between the system components, mainly you need to clearly identify

- **The purpose of the interaction** (e.g. “uses”, “persists trade data through”, etc)
- **Communication style,** how the components communicate (e.g. synchronous, asynchronous, batched, two-phase commit, etc)



Container Diagram

- A container represents something in which components are executed or where data resides.
- Help to illustrate the high-level technology choices in your system.
- It makes the high-level technology choices explicit.
- It shows where there are relationships between containers and how they communicate.

Container Diagram

A container diagram help answering the following questions

- What is the overall shape of the software system?
- What are the high-level technology decisions?
- How are responsibilities distributed across the system?
- How do containers communicate with one another?
- As a developer, where do I need to write code in order to implement features?

Container Diagram

For each of the container drawn on the diagram, you could specify:

- Name: The logical name of the container (e.g. “Internet-facing web server”, “Database”, etc)
- Technology: The technology choice for the container (e.g. Apache Tomcat 7, Oracle 11g, etc)
- Responsibilities: A very high-level statement or list of the container’s responsibilities. You could alternatively show a small diagram of the key components that reside in each container, but I find that this usually clutters the diagram

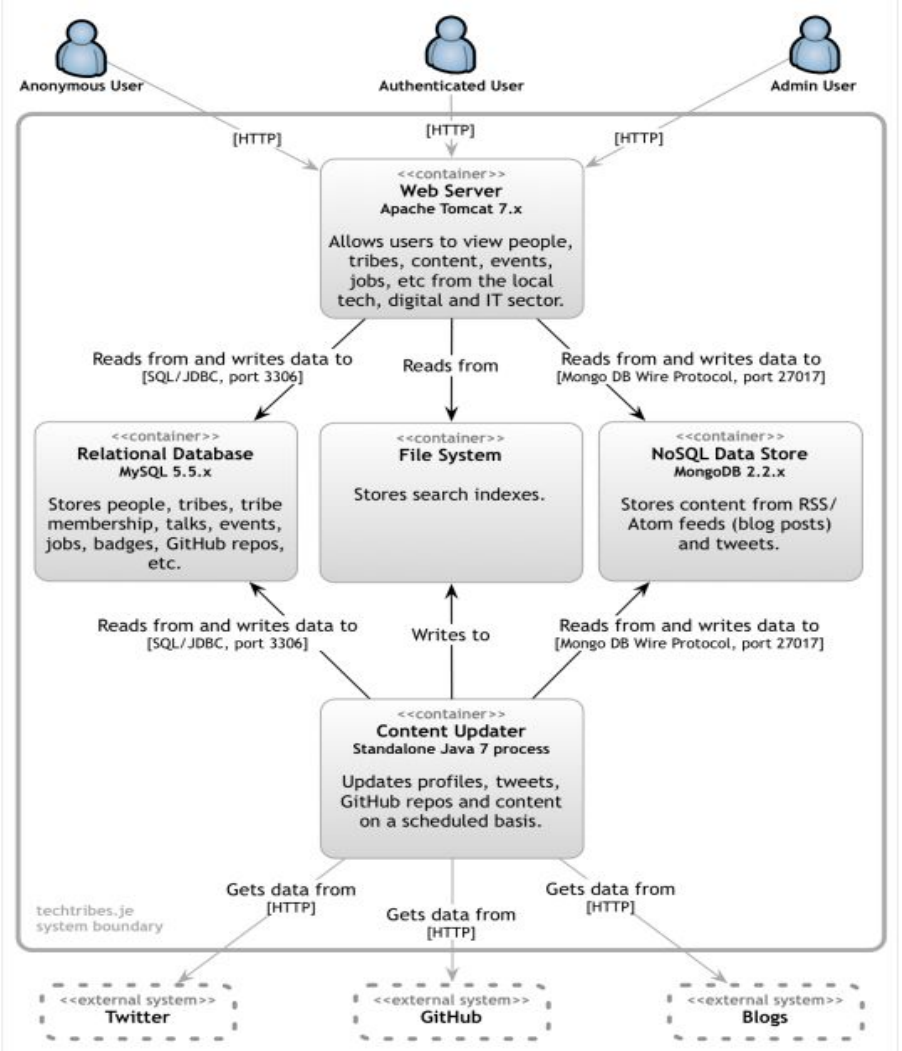
Container Diagram

Describe the interactions between the system

- The purpose of the interaction (e.g. “reads/writes data from”, “sends reports to”, etc).
- Communication method (e.g. Web Services, REST, Java Remote Method Invocation, Windows Communication Foundation, Java Message Service). • Communication style (e.g. synchronous, asynchronous, batched, two-phase commit, etc)
- Protocols and port numbers (e.g. HTTP, HTTPS, SOAP/HTTP, SMTP, FTP, RMI/IIOP, etc).

Container Diagram

Note container diagram does not explain the actual deployment



Interaction & Behavioral Models

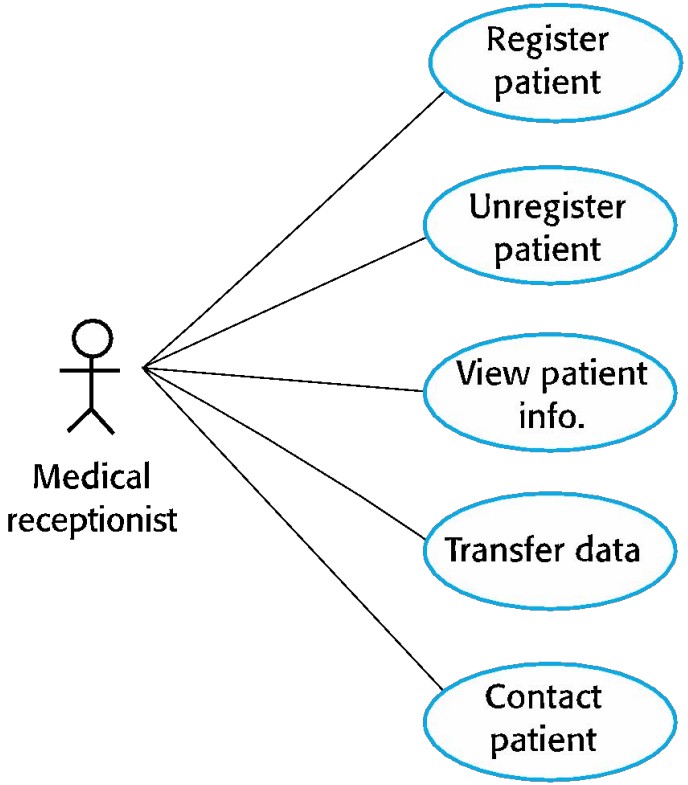
Interaction Models

- Modeling user interaction is important as it helps to identify user requirements.
- Modeling system-to-system interaction highlights the communication problems that may arise.
- Modeling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.
- Use case diagrams and sequence diagrams may be used for interaction modeling.

Use Case Diagram

- Use cases were developed originally to support requirements elicitation and now incorporated into the UML.
- Each use case represents a discrete task that involves external interaction with a system.
- **Actors** in a use case may be **people or other systems**.
- Represented diagrammatically to provide an overview of the use case and in a more detailed textual form.

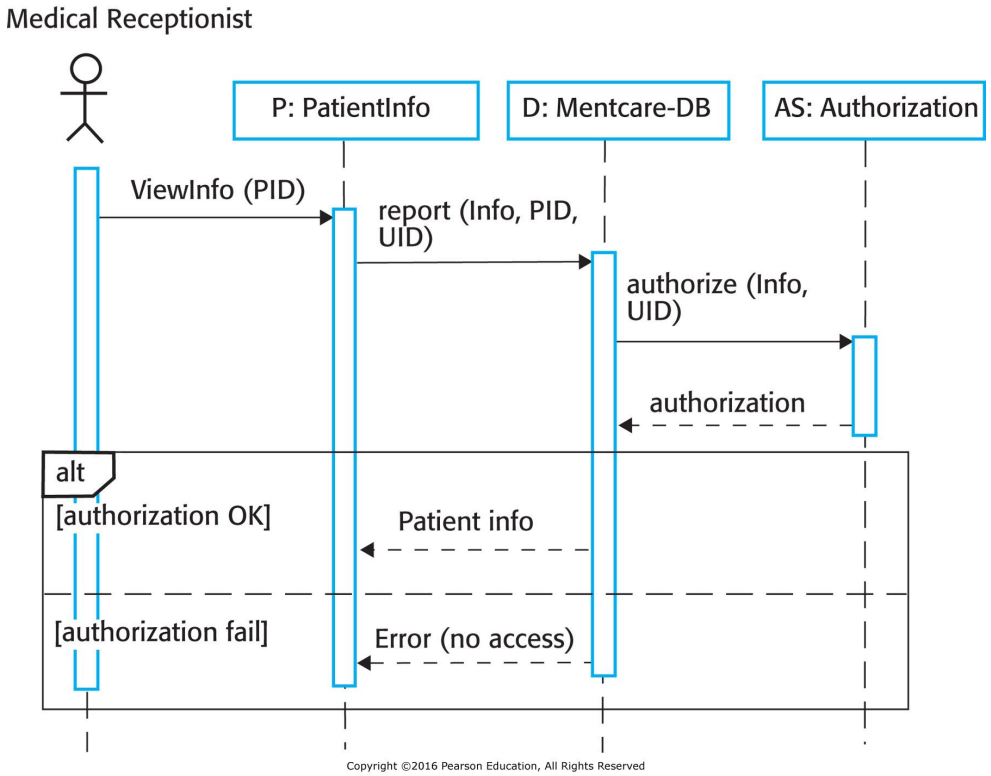
Use Case Diagram



Sequence Diagram

- Sequence diagrams are part of the UML and are used to model the **interactions between the actors and the objects within a system**.
- A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance.
- The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these.
- Interactions between objects are indicated by annotated arrows.

Sequence Diagram



Behavioral Models

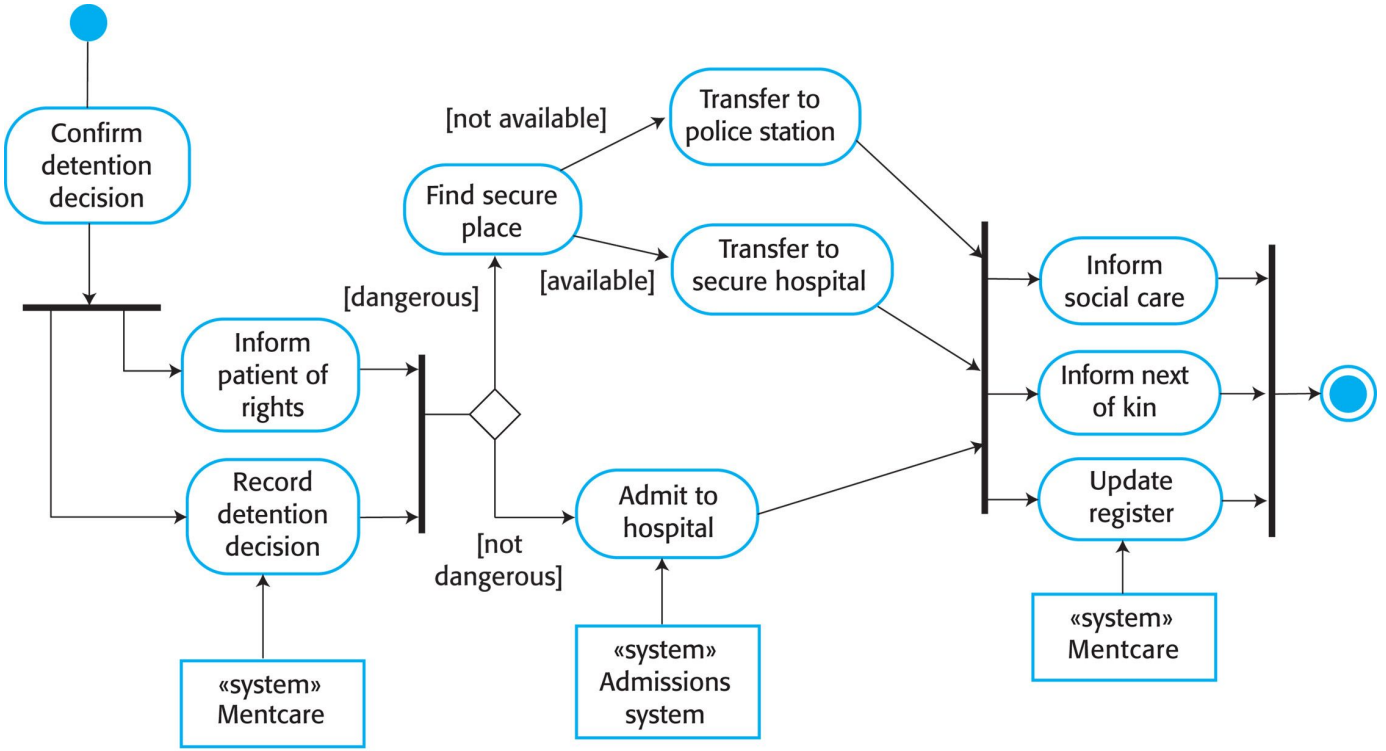
- Behavioral models are models of the dynamic behavior of a system as it is executing.
- They show what happens or what is supposed to happen when a system responds to a stimulus from its environment.
- You can think of these stimuli as being of two types:
  - **Data** Some data arrives that has to be processed by the system.
  - **Events** Some event happens that triggers system processing. Events may have associated data, although this is not always the case.

Activity Diagram

Activity diagrams present a number of benefits to users. Consider creating an activity diagram to:

- Demonstrate the logic of an algorithm.
- Describe the steps performed in a UML use case.
- Illustrate a business process or workflow between users and the system.
- Simplify and improve any process by clarifying complicated use cases.
- Model software architecture elements, such as method, function, and operation.

Activity Diagram

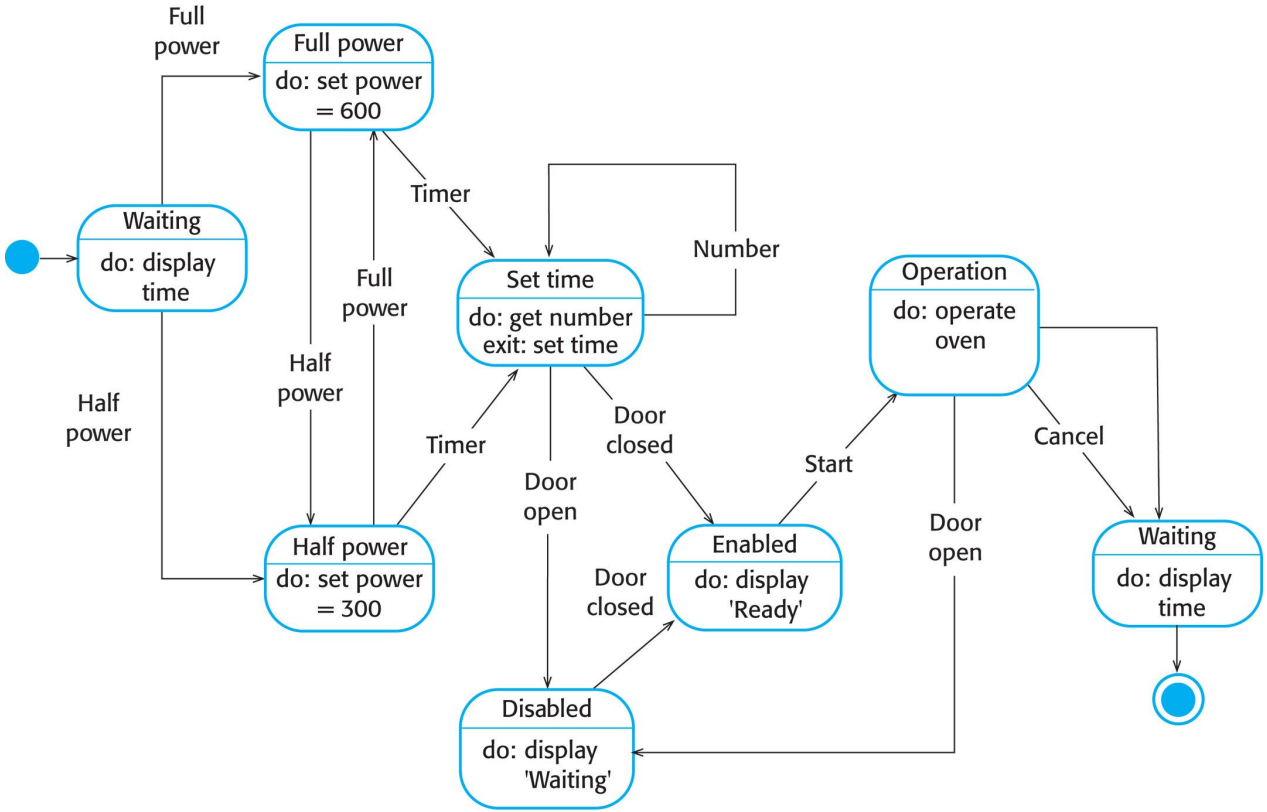


State Diagram

State diagrams present a number of benefits to users. Consider creating a state diagram to:

- Depicting event-driven objects in a reactive system.
- Illustrating use case scenarios in a business context.
- Describing **how an object moves through various states** within its lifetime.
- Showing the overall behavior of a state machine or the behavior of a related set of state machines.

State Diagram



Questions

References

- Chapter 5, Software-Engineering-10th-Ian-Sommerville
- Software architecture for developers by simon brown