ID2207
Modern Methods in Software Engineering

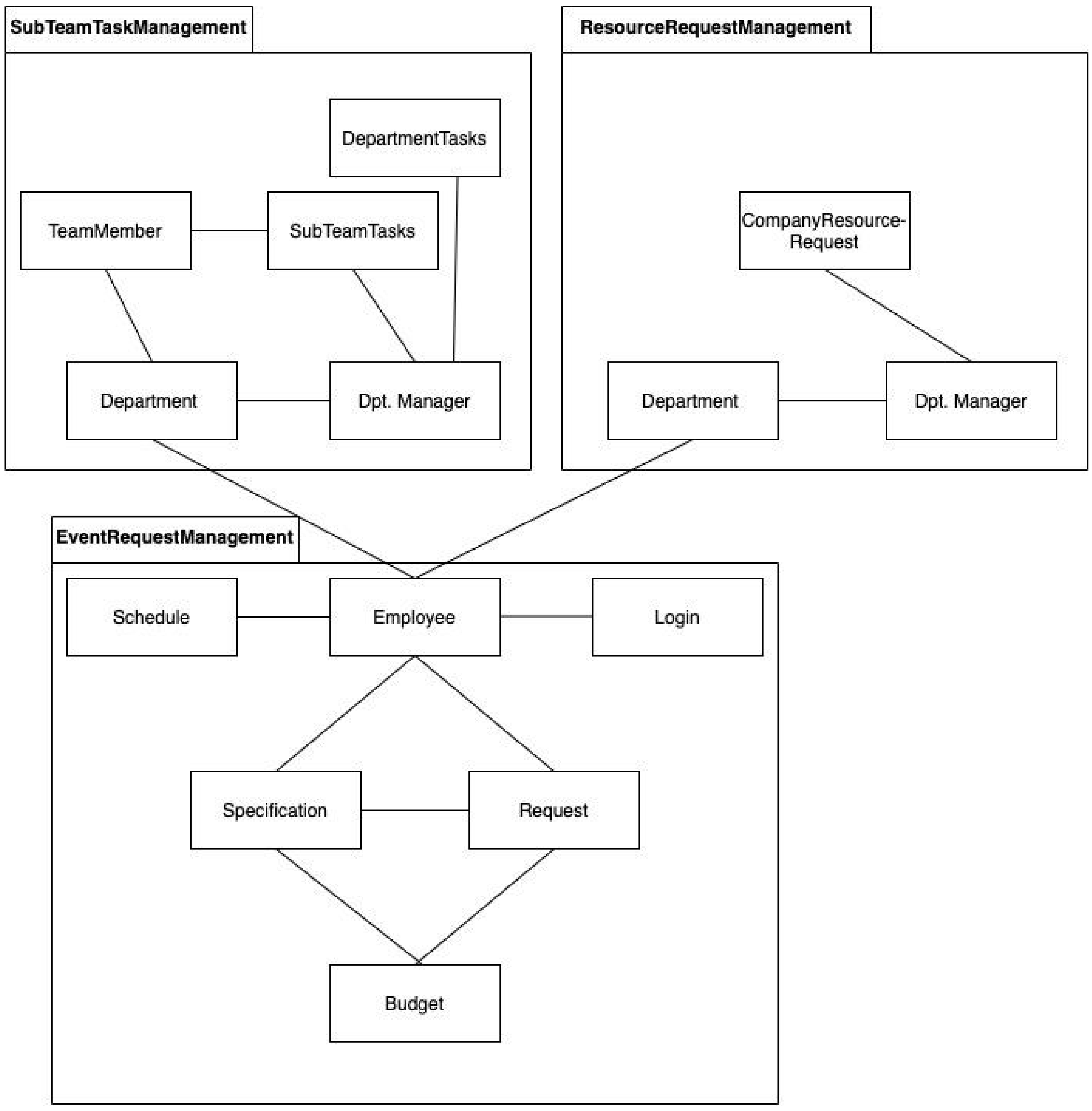
Homework 4 - Group 19

October 2021
Royal Institute of Technology

Aksel Uhr
Abdullah Abdullah

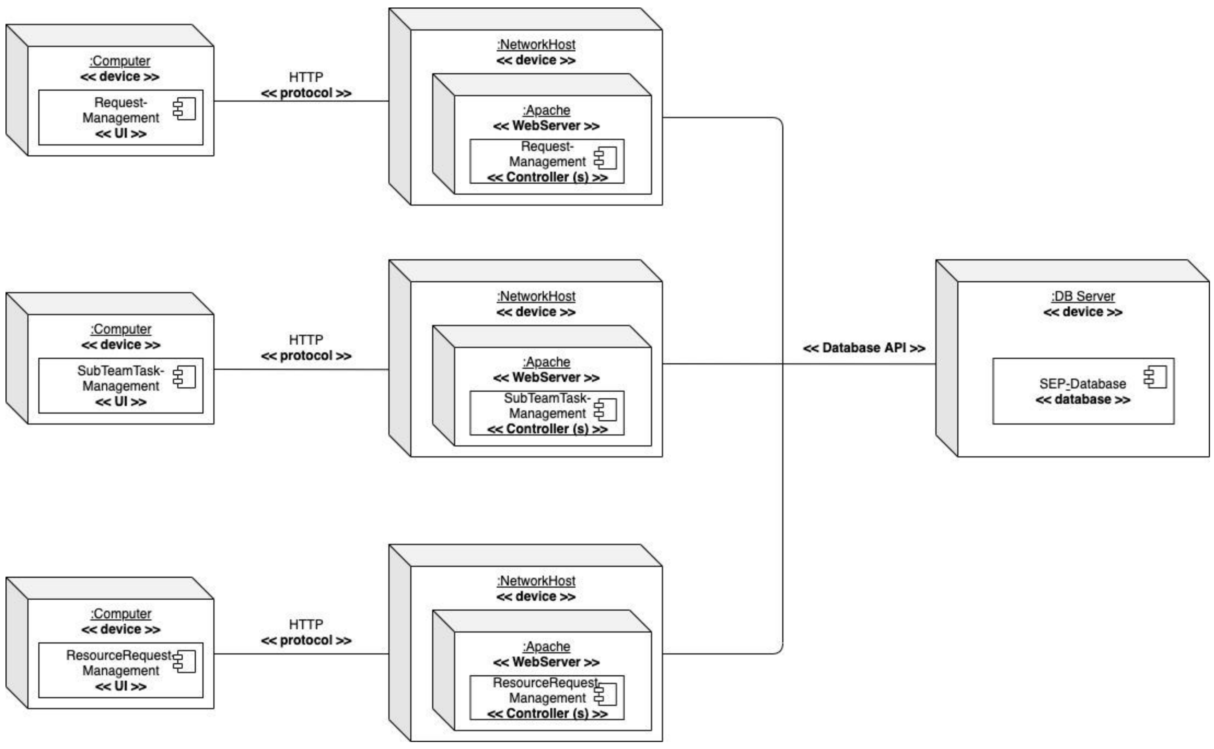
1. Subsystem decomposition structure

The class diagram below is an abstraction of the class diagram which was modeled in the previous assignment. The system is also divided into different subsystems in order to enhance the cohesion, meaning that classes with many relationships with each other are categorized into a certain subsystem.



2. Mapping subsystems to processors and components

The left nodes are computers containing each subsystem's graphical user interface. From here, the user might send various requests to the web browser (e.g. Google chrome, Mozilla) which in turn communicates the request to the web server. The web server contains each subsystem's application logic, which is marked as components being "controller(s)". Those components encapsulate the behavior of the system, namely executing functions in order for the system to operate accordingly. An example of this would be executing functions for communicating with the database in order to retrieve, update or insert data, which is depicted below.



Aksel Uhr
Abdullah Abdullah

3. Persistent storage solution for the problem

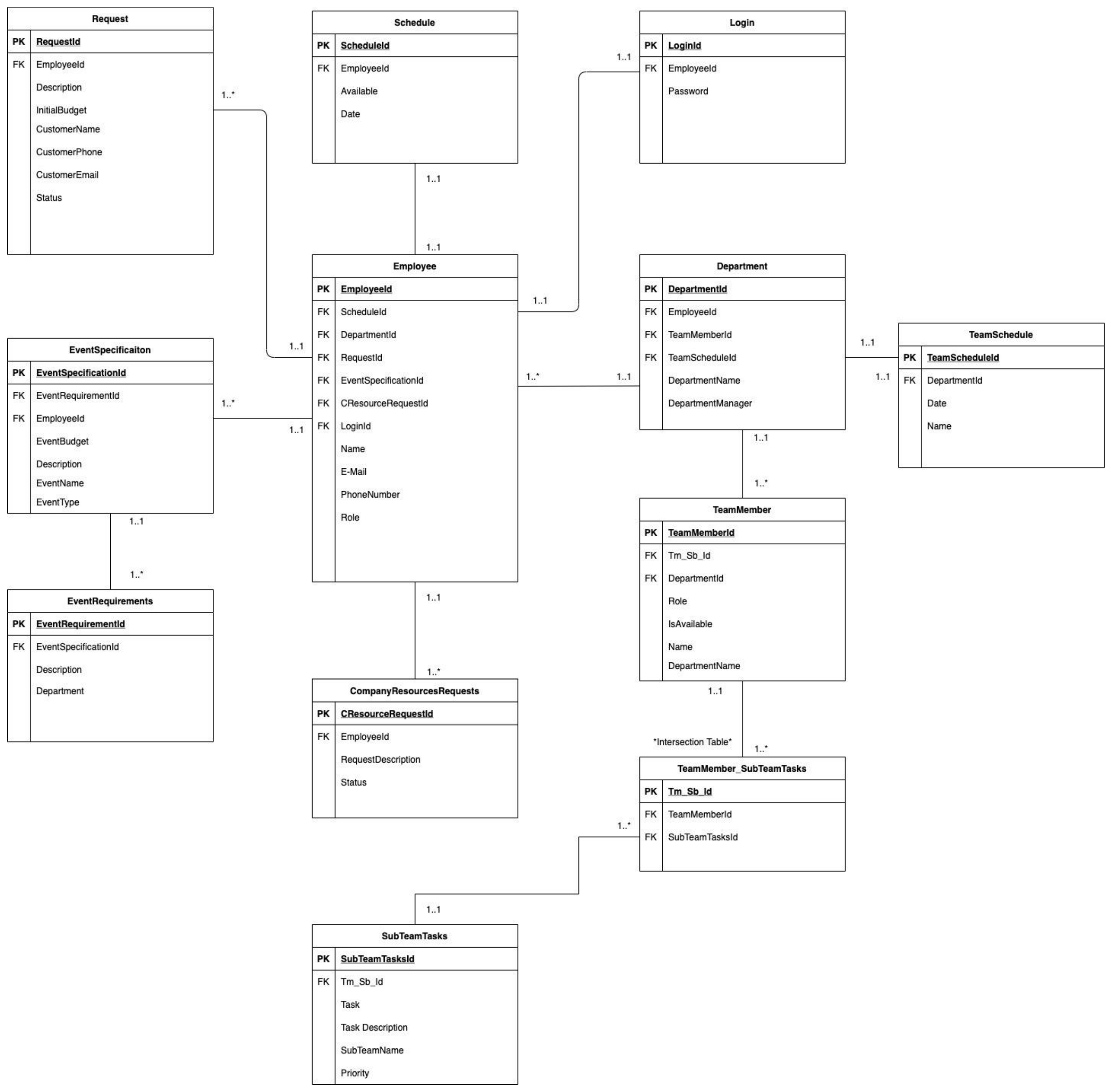
List of persistent objects

- *Employee entity*
  - (basic information regarding employees)
- *Login entity*
  - (login credentials of the employees, this is not in last document)
- *Authentication and Authorization entity*
  - (permission and validation upon employee login)
- *Schedule entity*
  - (info regarding employees schedules)
- *Department member entity*
  - (current projects, tasks and availability of the employees)
- *Request entity*
  - (info regarding customer's request)
- *Budget entity*
  - (info regarding customer's requests budget)
- *Event Specification entity*
  - (stores all data of different requirements of customers events)
- *Company Resource entity*
  - (info about need of hiring new employee or outsourcing the task)
- *Company Budget entity*
  - (stores data about company's budget)
- *Team Tasks Entity*
  - (Hire or outsource the project)

Storage management strategy

For the SEP system, storage management will be handled using relational databases. This allows for usage of SQL which in turn is a useful querying language for updating, deleting, modifying, inserting (etc.) data in a database. Since there are many functional requirements concerning creating, updating and deleting data (event requests, event specifications, employees etc.), a relational database and SQL is optimal. Querying the data is relatively easy and could thus be managed by technicians within the company. Relational databases scale well with large data sets, which means that as the company potentially grows, there won't be any need of changing the storage management in the future. Another argument for the chosen strategy follows that class diagrams have already been derived thus far. As class diagrams are somewhat similar to entity relationship diagrams (abstraction of a relationship database), one can use the class diagrams as a starting point in designing the relationship database. What needs to be done from here is adding unique ids to entities, deciding attributes, characterizing relations with primary and foreign keys, mapping cardinality and ultimately making sure that normalisation is achieved. See the ERD below. Please note that some persistent objects have been aggregated into one entity or entirely excluded (for example budget and authorization entity).

Entity relationship diagram (extra effort)



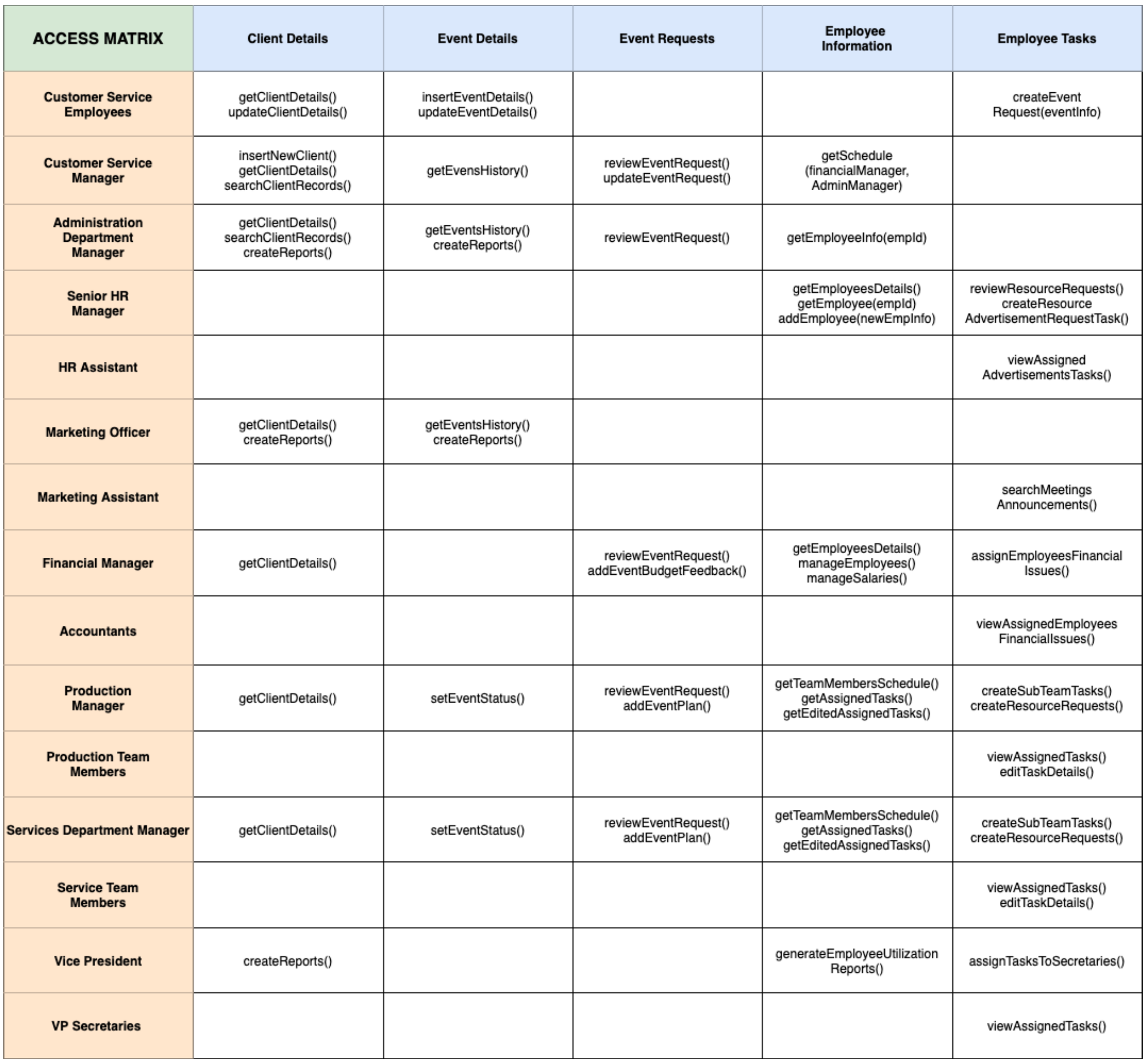
October 2021
Royal Institute of Technology

Aksel Uhr
Abdullah Abdullah

4. Access control, global control flow and boundary conditions

ACCESS CONTROL

1) **Access Matrix**



ACCESS MATRIX	Client Details	Event Details	Event Requests	Employee Information	Employee Tasks
Customer Service Employees	getClientDetails() updateClientDetails()	insertEventDetails() updateEventDetails()			createEvent Request(eventInfo)
Customer Service Manager	insertNewClient() getClientDetails() searchClientRecords()	getEvensHistory()	reviewEventRequest() updateEventRequest()	getSchedule (financialManager, AdminManager)	
Administration Department Manager	getClientDetails() searchClientRecords() createReports()	getEventsHistory() createReports()	reviewEventRequest()	getEmployeeInfo(empId)	
Senior HR Manager				getEmployeesDetails() getEmployee(empId) addEmployee(newEmpInfo)	reviewResourceRequests() createResource AdvertisementRequestTask()
HR Assistant					viewAssigned AdvertisementsTasks()
Marketing Officer	getClientDetails() createReports()	getEventsHistory() createReports()			
Marketing Assistant					searchMeetings Announcements()
Financial Manager	getClientDetails()		reviewEventRequest() addEventBudgetFeedback()	getEmployeesDetails() manageEmployees() manageSalaries()	assignEmployeesFinancial Issues()
Accountants					viewAssignedEmployees FinancialIssues()
Production Manager	getClientDetails()	setEventStatus()	reviewEventRequest() addEventPlan()	getTeamMembersSchedule() getAssignedTasks() getEditedAssignedTasks()	createSubTeamTasks() createResourceRequests()
Production Team Members					viewAssignedTasks() editTaskDetails()
Services Department Manager	getClientDetails()	setEventStatus()	reviewEventRequest() addEventPlan()	getTeamMembersSchedule() getAssignedTasks() getEditedAssignedTasks()	createSubTeamTasks() createResourceRequests()
Service Team Members					viewAssignedTasks() editTaskDetails()
Vice President	createReports()			generateEmployeeUtilization Reports()	assignTasksToSecretaries()
VP Secretaries					viewAssignedTasks()

2) **Brief description about security, authentication/authorization strategy, confidentiality of data, and network/infrastructure security**

- To have secure access to the system each user will be authenticated using email, hashed password and JWT (Json Web Tokens) tokens.
- To ensure safety of systems data each user in the system will be authorized with different roles which will allow only required access of data for that specific role.
- To have a secure system that is not vulnerable to attacks Refresh Tokens and Access Tokens will be used in the system.
- User critical/sensitive data and credentials will be stored in encrypted/hashed form to protect against security attacks.

- Regular Backup of data will be done to cope against server disasters.
- For Network Security we can have local network connection but a firewall will be needed to connect to the internet for using other applications such as emails, browsers. Another strategy can be using a VPN connection to access the system on the internet.

GLOBAL CONTROL FLOW

1) **Brief description of the selected flow for the system**

Decentralized Design
In decentralized design a single object is not in control, instead the control is distributed which means that there are more then one controlling objects in the system.
In previous assignments the proposed solution of sequence diagrams and use cases is a good fit for the approach of object oriented solution of the business and for that purpose the decentralized design is the best one to choose.

Event Driven Microservices (Extra Effort)
In event driven control resides within a dispatcher calling functions via callbacks. For the system we can have an event driven microservices based architecture that is very flexible. The advantage could be that we can extend the functionality later on by adding more services. Another advantage of the system will be that we can have independent services which could work on their own if there are failures for other services. One point to remember is that in Pub/Sub event driven structures the reliability and guarantee delivery of event source is very critical.

MultiThreading (Extra Effort)
Threads allow handling concurrent processes so instead waiting for data to arrive another action can be performed simultaneously.
Our system needs to handle multiple actions simultaneously like accessing employees information while handling requests for events data from an employee.

BOUNDARY CONDITIONS

Boundary Use Cases

Name: ManageNewSEPSystem
Participating actor(s): 1. SEP System Administrator
Entry conditions: 1. System Administrator logs in to the system.
Exit conditions: 1. Add new users to the system with their specific roles in the system.

2. The system is read to start the functionality.

Quality conditions:
1. The system should be available.
2. New users should be added with correct details and credentials.
3. Roles for users should be added in the system to have limited access in the system.

Event Flow:
1. The system administrator accesses the system.
2. System Administrator adds new users with provided details like credentials, role, department.
3. Initial configuration of the system is done by the system administrator.
4. Access is given to the new users added.

Name: ManageNewServer

Participating actor(s):
1. SEP System Administrator

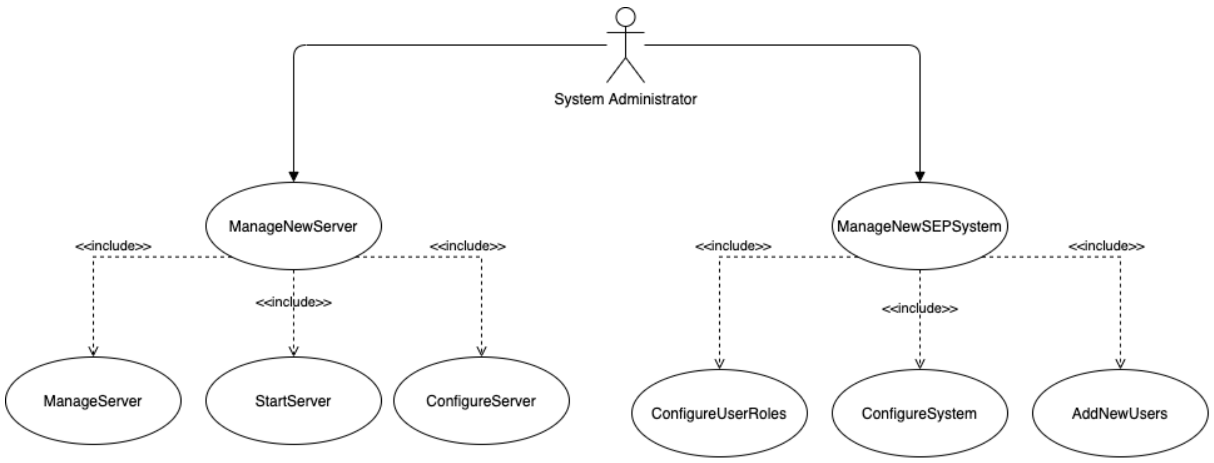
Entry conditions:
1. System Administrator logs in to the server.

Exit conditions:
1. Server is active and available to handle requests from the SEP client side.

Quality conditions:
1. The server machine should be available.
2. The server services should be correctly configured and operational.

Event Flow:
1. The system administrator accesses the server machine.
2. System Administrator adds the initial configuration of the server like the number of copies of each service needed on the server side, port configurations etc.
3. Adding failure conditions like auto initializing a copy of service in case of failure.
4. System Administrator starts the server and is now available to handle requests.

Use Case Diagram



OTHER Boundary Conditions

FAILURE
Hardware Failure in Users Computer and Server Computers.
Network Failures
Software Failure in SEP System due to bugs in different functionalities.

5. Applying design patterns to designing objects for the problem

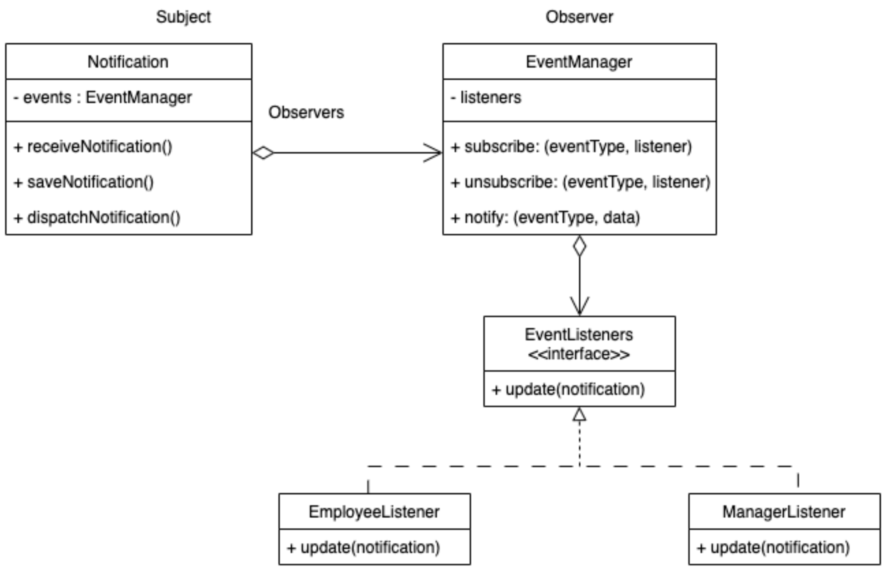
We have different types of design patterns. For the SEP System the design patterns we will be using are

BEHAVIOURAL OBSERVER PATTERN
Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically. Observer pattern falls under the behavioral pattern category.

In our SEP the system needs to notify other objects of changes in an object. From our previous assignment we had a notification controller for the purpose to notify other objects within the system when new tasks are assigned, new requests are created or any other important events occur that need to be communicated to all other objects.

For example when a task is assigned to a team member by the manager the team member and the manager both should be notified about the new update.

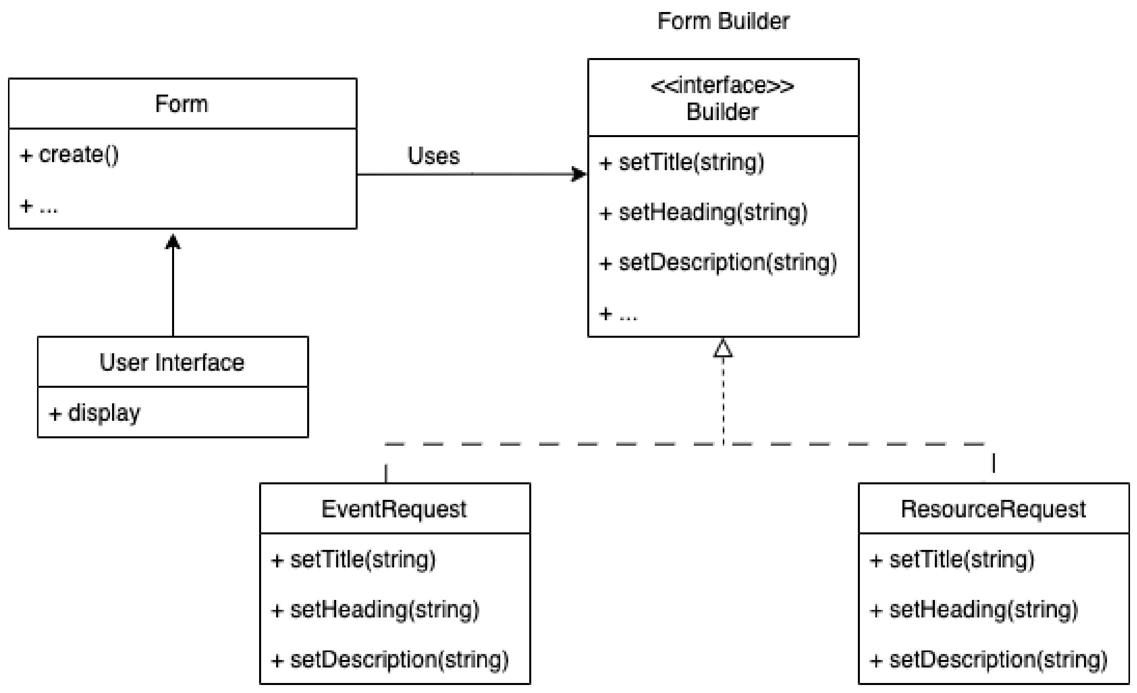
The Observer pattern suggests that you add a subscription mechanism to the publisher class so individual objects can subscribe to or unsubscribe from a stream of events coming from that publisher.



CREATIONAL BUILDER PATTERN
Builder pattern builds a complex object using simple objects and using a step by step approach. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In our SEP we have a common pattern for different forms of requests like createEventRequest, createResourceRequest, createBudgetRequest etc. So the builder pattern can be used to create these form objects.

The Builder pattern suggests that you extract the object construction code out of its own class and move it to separate objects called builders.



6. Writing contracts for noteworthy classes

- Event requests can only be filled by customer service

context Login::authorizeUser(u:Employee) **pre**:
displayRequestForm()

- Pre condition since in order to display the request form, the user must have the customer service role (operation).

- An event request must be filled with all customer event details and customer information before being sent further and inserted in the database

context Request::createRequest(r:Request) **post**:

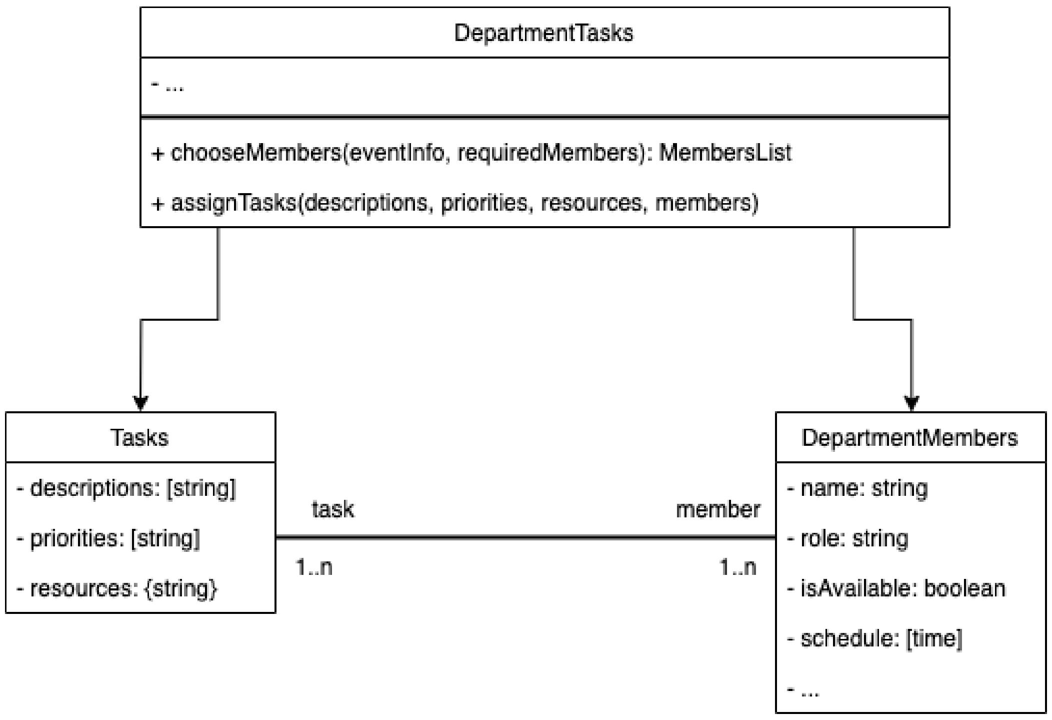
isRequestFilled(r)

- Post condition since isRequestFilled must be true after a new event request (the operation) is created.

● The event specification must have event requirements attached to it

context EventSpecification **inv**:
!self.getRequirements.isEmpty()

- Invariant as this must be true for all event specification instances.



● When an event is approved, then members from the production and the service team will be choosed to cover the particular event. The constraint here will be post condition that the required number of members chosen to cover the event should be available.

context DepartmentTasks::chooseMembers(eventInfo,requiredMembers): MembersList
post: chooseMembers -> exists(m | m.isAvailable) and (chooseMembers -> size >=) requiredMembers

- Post-condition since after choosing members the outcome should be that the required selected members will be available to cover the event requirements.

● To assign tasks to sub team members by the production and service managers the required condition or constraint will be to see the availability of required team members to cover the event requirements.

context DepartmentTasks::assignTasks(taskDescriptions,priorities,resources,members)
pre: members -> exists(m | m.isAvailable) and (members -> size) >= members.length

- Pre-condition since before assigning tasks to team members it should be checked the required number of members to cover the event requirements are available.