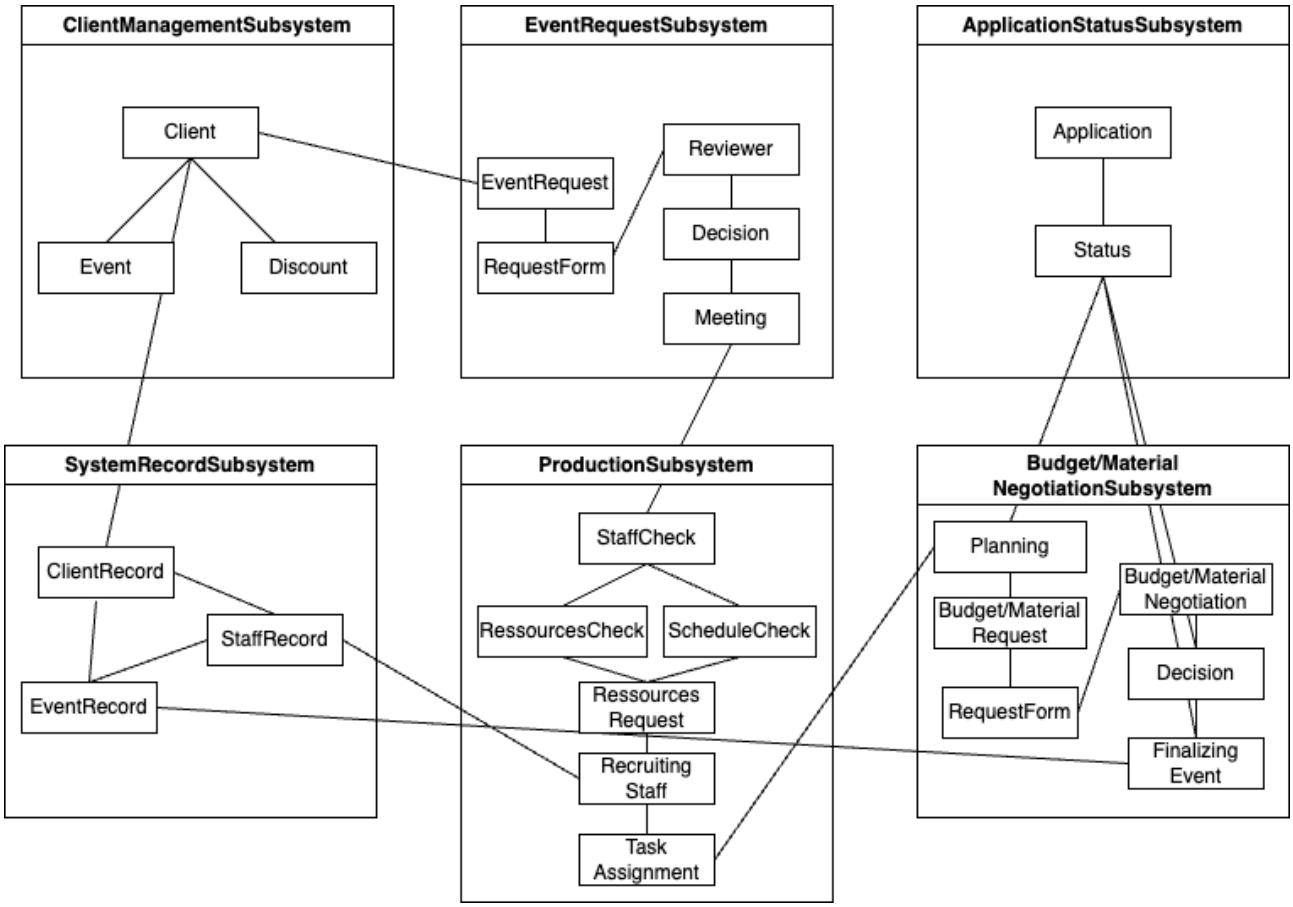
ID2207 HT23 Modern Methods in Software Engineering (50928)

Homework 4 - Group 8
Dominika Drela, Eugen Lucchiari Hartz

System design and object design document

Subsystem decomposition structure (using class diagrams)



ClientManagementSubsystem: The ClientManagementSubsystem is responsible for managing client information, including details of events, and handling discounts for frequent clients.

EventRequestSubsystem: The EventRequestSubsystem is responsible for managing the process of receiving client requests, entering data into forms, and forwarding them through the approval hierarchy. It is also includes organised meetings with clients after approval where preferences and planned budgets are discussed with relevant managers.

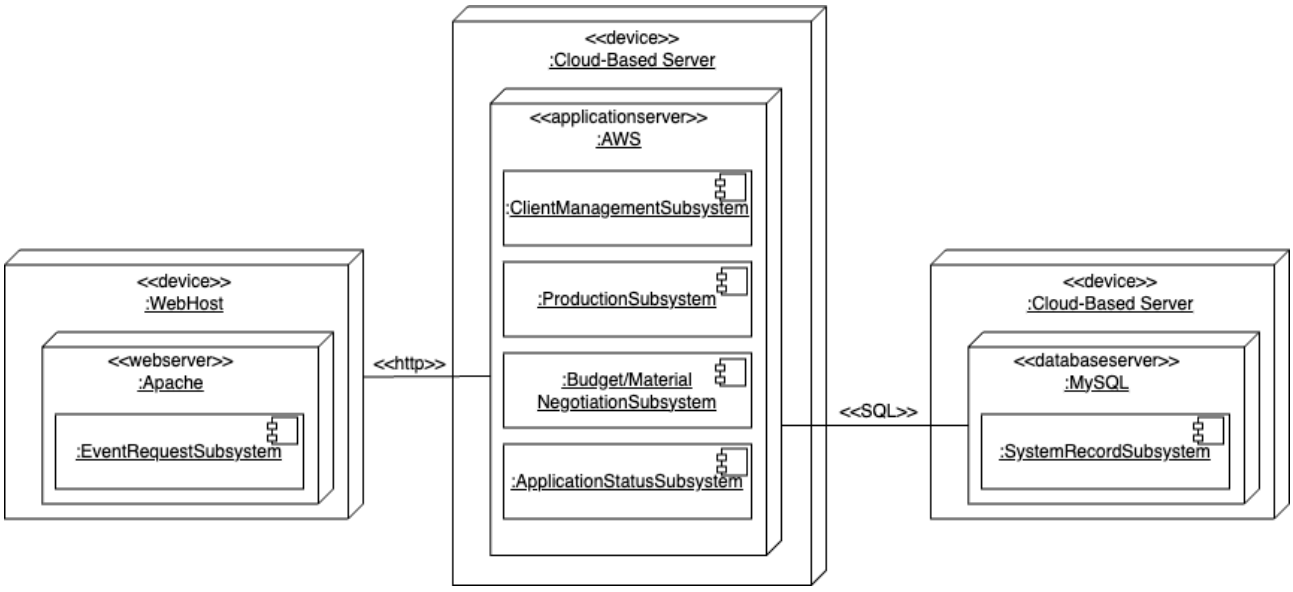
ApplicationStatusSubsystem: The ApplicationStatusSubsystem handles the lifecycle of event applications (open, in progress, closed).

SystemRecordSubsystem: The SystemRecordSubsystem maintains records of clients, events, and staff members, including information related to assignments and client history.

ProductionSubsystem: The ProductionSubsystem is responsible for managing resource allocation within the production department, including staff scheduling and handling requests for additional resources. It also includes the management of tasks.

Budget/Material NegotiationSubsystem: The Budget/Material NegotiationSubsystem is responsible for the negotiation of budgets with clients and ensures agreement between the client and the financial department. Once all issues are solved the event gets finalised.

Mapping subsystems to processors and components (hardware/software mapping) using UML deployment diagrams



Web Server: Responsible for handling initial client requests and forwarding them to the Application Server. In this case, a server like Apache is used. I communicates with the application server via http.

Application Server: Executes the business logic of the subsystems, so it handles tasks such as request processing, workflow management and interacts with the Database Server. The application server in this case is hosted in the cloud by using the cloud service AWS.

Database Server: Manages the storage and retrieval of data for all subsystems. In this scenario, MySQL is used as an example.

Persistent storage solution for the problem

List of persistent objects:
- ClientManagementSubsystem: Client records, including details of all events managed by the organisation.
- EventRequestSubsystem: Information related to incoming requests, potentially involving client details and request metadata, details of requests, reviews, and approvals, information related to organised meetings, including client preferences and planned budgets.
- ApplicationStatusSubsystem: Data related to the lifecycle of applications, including status, progress, and completion information for each event.
- SystemRecordSubsystem: Records of clients, events, staff members, and historical data related to managed events.
- ProductionSubsystem: Data about the production process, including staff assignments, tasks, and resource requirements.
- Budget/Material NegotiationSubsystem: Details of budget negotiations, including proposed budgets, financial manager feedback, and client agreements.

Brief description of the selected storage management strategy:
These objects should be stored in a Relational Database Management System (RDBMS). Choosing a Relational Database Management System (RDBMS) makes sense for our Event Management System because it's really good at organising large data sets, like information about clients and events. Furthermore, there’s a need for complex queries involving relationships between different attributes. An RDBMS is better suited for this need. An RDBMS accordingly provides a robust solution for efficient data retrieval.

Access control, global control flow, and boundary conditions

Access Matrix

Objects Actors	Client Records	Event Requests	Staff Records	Budjet Adjustments
Customer Service Officer	Read, Create	Create	None	None
Senior Customer Service	Read, Create	Read, Update	Create, Read	None
HR Team	Read, Update	Read, Update	Create, Read	Read, Update
Marketing Team	Read	Read	None	None
Administration Team	Read, Update	Read, Update	Create, Read	None
Financial Manager	Read	Read, Update	Read, Update	Read, Update
Production Manager	Read, Update	Read, Update	Read, Update	Read, Update
Service Manager	Read, Update	Read, Update	Read, Update	Read, Update
Vice President	Read	Read, Update	Read, Update	Read, Update

Brief description about security, authentication/authorisation strategy, confidentiality of data, and network/infrastructure security

Security:
The system will implement robust security measures to protect data and ensure the integrity of operations.
Security protocols will be in place to prevent unauthorized access, data breaches, and cyberattacks.
Regular security audits and vulnerability assessments will be conducted to identify and mitigate potential threats.

Authentication/Authorization Strategy:
User authentication will be based on secure methods such as username and password, two-factor authentication, or biometrics.
Role-based access control (RBAC) will be implemented to ensure that users only have access to functionalities relevant to their roles.
Access control lists (ACLs) will specify fine-grained permissions for data and operations.

Confidentiality of Data:
Data encryption will be used to ensure the confidentiality of sensitive information, such as client records, financial data, and employee details.
Access to confidential data will be restricted to authorized personnel only, based on their roles and permissions as indicated in the access matrix.

Network/Infrastructure Security:
Firewalls, intrusion detection systems, and intrusion prevention systems will be deployed to protect the network from external threats.
Data transmission over networks will be encrypted using secure protocols like HTTPS.
Regular security patches and updates will be applied to servers and infrastructure components to address known vulnerabilities.

Brief description of the selected control flow for the system

The global control flow of the system uses an event-driven control approach, which means that actions are triggered by specific events or user interactions. This approach is flexible and well-suited for tasks like designing user interfaces and making it easy to add new features or functionality as needed.

Boundary use cases

Use case name: UserRegistration
Entry condition: User decides to register for the system.
Flow of Events: 1. User accesses the registration interface. 2. Enters required information (e.g., username, password, email). 3. System validates entered data. 4. If validation passes, the system registers the user. 5. Generates a confirmation message.
Exit condition: User successfully registered and can now log in with the provided credentials.

Use case name: EventRequestHandling
Entry condition: A client initiates a new event planning request through customer service.

Flow of Events:
1. The system starts as the client initiates the request.
2. Basic system configurations and user authentication data are accessed.
3. Necessary services, like customer service handling, are registered.
4. The user interface presents an event details form.
5. The request is submitted to the Senior Customer Service Officer.
6. Senior Customer Service validates and may redirect to administration or financial managers.
7. Administration or financial manager reviews and provides feedback or approvals.
8. System updates the request status.
9. If approved, customer service contacts the client.
10. A meeting is scheduled to discuss preferences and planned budget.
11. Graceful termination of involved subsystems.
12. Notification to relevant subsystems if necessary.
13. Local updates, if any, communicated to the database.
14. Customer service prepares documentation and notifies stakeholders.

Exit condition:
The event request is successfully handled up to preparing for the business meeting with the client.

Use case name: ProductionHandling

Entry condition:
Production manager initiates staff check and task assignment.

Flow of Events:
1. System startup for task initiation.
2. Access to basic configurations and user data.
3. Registration of production management services.
4. User interface presents staff check and task assignment options.
5. Production manager reviews staff availability.
6. Identifies conflicts or shortages.
7. Requests additional resources from HR.
8. System notifies HR of resource request.
9. HR reviews and updates records.
10. Initiates recruitment or outsourcing if needed.
11. Production manager assigns tasks to sub-teams.
12. Sub-teams plan activities and note budget requirements.
13. Production manager reviews sub-team comments.
14. Requests budget negotiation with financial manager if needed.
15. Application processing upon client and financial department budget agreement.

Exit condition:
Successful handling of staff check and task assignment, including budget negotiation in the production subsystem.

Applying design patterns to designing object model for the problem

Observer Pattern:
The Observer Pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
In the event management system, the Observer Pattern can be applied to notify various subsystems or components (observers) when a significant event occurs, such as the approval of a client's event request. This allows for efficient and decoupled communication between different parts of the system.

Strategy Pattern:
The Strategy Pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.
In the budget negotiation subsystem, the Strategy Pattern can be applied to represent different negotiation strategies. Depending on the context or the type of event, the system can dynamically switch between negotiation algorithms without altering the client code. This allows for flexibility and easy extension of negotiation strategies.

Command Pattern:
The Command Pattern encapsulates a request as an object, thereby allowing for parameterisation of clients with different requests, queuing of requests, and logging of the parameters.
For the event planning requests, the Command Pattern can be applied to encapsulate the client's request as an object. This allows for queuing and executing requests in a structured manner, enabling undo/redo functionality and logging of requests for auditing purposes.

Writing contracts for 5 noteworthy classes

Client Class:
- Invariant: The client's name should not be empty.
- Precondition: Before updating the client's information, ensure that the new name is not empty.
- Postcondition: After updating the client's information, the new name should not be empty.

context Client
inv: self.name <> ''

pre: newName <> ''
post: self.name <> ''

EventRequest Class:

- Invariant**:** The event type should be one of the predefined types.
- Precondition**:** Before creating an event request, ensure that the event type is valid.
- Postcondition**:** After creating an event request, the event type should be valid.

context EventRequest
inv: self.eventType = 'Conference' or self.eventType = 'Wedding' or self.eventType = 'Workshop'

pre: newEventType = 'Conference' or newEventType = 'Wedding' or newEventType = 'Workshop'
post: self.eventType = newEventType

FinancialManager Class:

- Invariant**:** The discount applied by the financial manager should be within a reasonable range (0% to 30%).
- Precondition: Before updating the discount percentage, ensure that the new value is within the valid range.
- Postcondition: After updating the discount percentage, it should be within the valid range.

context FinancialManager
inv: self.discountPercentage >= 0 and self.discountPercentage <= 30

pre: newDiscount >= 0 and newDiscount <= 30
post: self.discountPercentage = newDiscount

ProductionManager Class:

- Invariant: Each sub-team should have at least one assigned task.
- Precondition: Before assigning tasks, ensure that there is at least one task to assign.
- Postcondition: After assigning tasks, each sub-team should have at least one assigned task.

context ProductionManager
inv: self.subTeams->forAll(subTeam | subTeam.tasks->notEmpty())

pre: newTasks->notEmpty()
post: self.subTeams->forAll(subTeam | subTeam.tasks->notEmpty())

ResourceAllocation Class:

- Invariant: Each resource allocation should have a positive allocation amount.
- Precondition: Before updating the allocation amount, ensure that the new amount is positive.
- Postcondition: After updating the allocation amount, it should be a positive value.

context ResourceAllocation
inv: self.allocationAmount > 0

pre: newAllocationAmount > 0
post: self.allocationAmount = newAllocationAmount