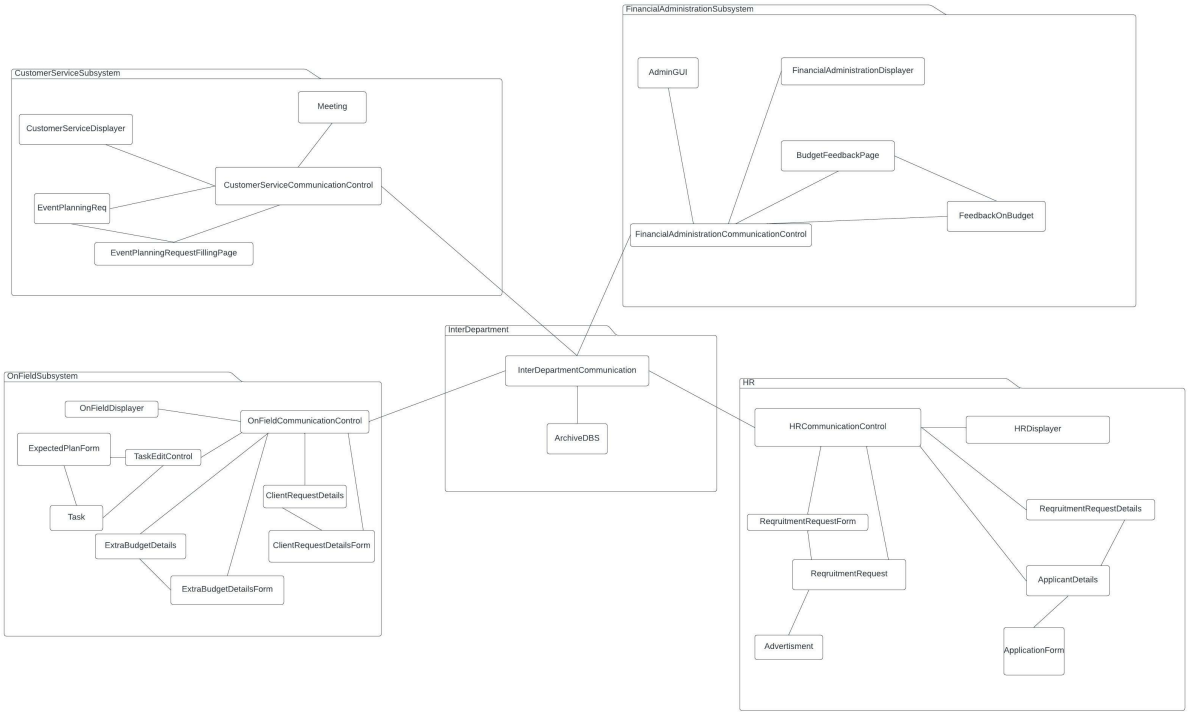
MMSE HW4

NIkoloz Miruashvili, Reethika Ambatipudi

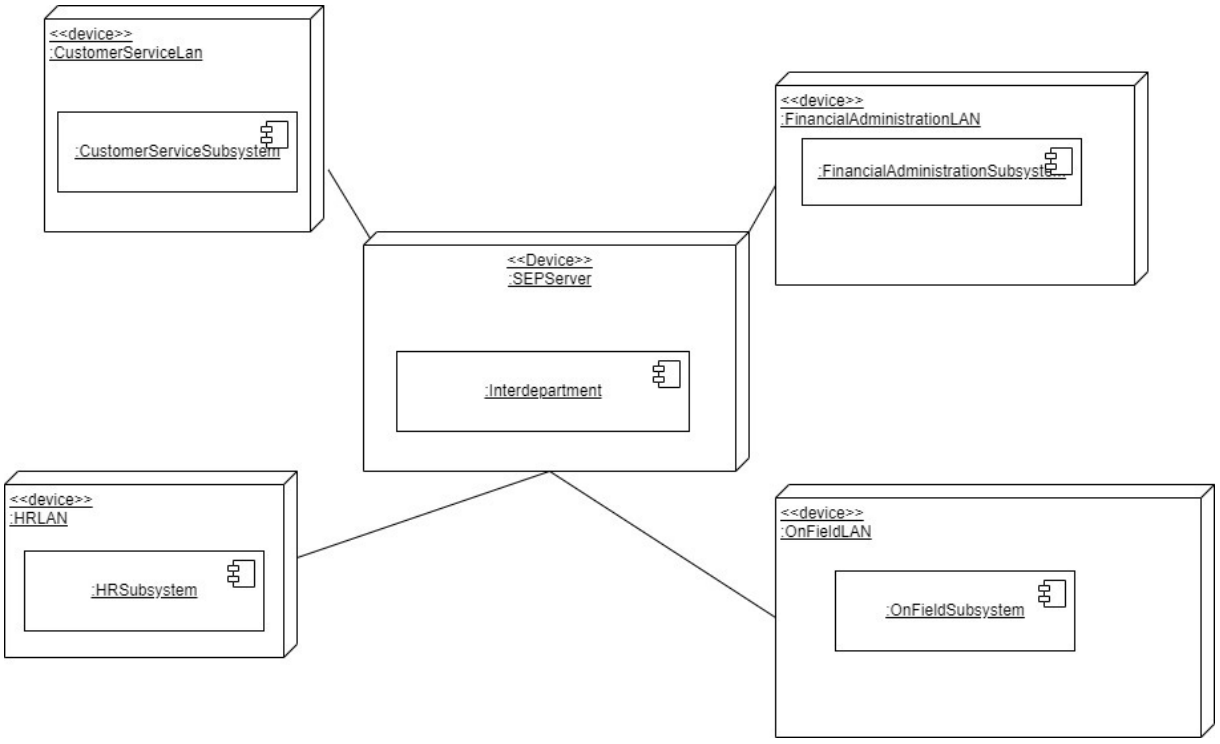
2nd October 2022

1 Subsystem decomposition structure



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

2.1 UML deployment diagram



2.2 The system is divided into 4 major subsystems, which will be deployed in Local Area Computer Networks, While there will be one connecting node deployed on a server controlling communication between the above mentioned subsystems, as well as access to the archives stored in the database.

3 Persistent storage solution for the problem

3.1 List of persistent objects -

- EventPlanningReq - It has all the information about the client's requirements like Event Type, Dates, Expected number of attendees, Preferences, Expected Budget.
- ClientRecord - Client details like Name, email ID, Phone number etc.
- EmployeeDetails - Employee ID, Name, Department, Contact Details, Emergency contact details, ID proofs, Visa validates etc.
- RecruitmentRequestDetails - Details like Contract type, Requesting Department, Required years of experience, Job title and Job description.
- ApplicantDetails - Information of the person who is applying for the job like Name, DOB, Visa validity, Years of experience, Expected salary etc.

3.2 Brief description of the selected storage management strategy -

The application is used concurrently by many people. Eventually we also should be ready to store a huge amount of data. So, Storing all the persistent objects in Relational data is better.

4 Access control, global control flow, and boundary conditions

4.1 Access Control -

1. Access matrix

Actor	EventPlanningReq	Task	ClientDetails	EmployeeDetails	RecruitmentReq	ApplicantDetails
CustomerServices Officer	createEventReq() sendToCSManager()					
CustomerServices Manager	updateEvent() recieveNewEventUpdate() sendToaAdminMan() sendToFinMan()		createClient()			
FinancialManager	updateEventReq() makeDiscount() recieveBudgetReqUpdate()		viewClients()	viewEmployees()		
AdminDept. Manager	updateEvent() generateEventReport()		generateClientReport()	generateEmployeeReport()		
OnField Managers	recieveNewEventUpdate() sendBudgetReqToFinMan()	initiateTask()		viewStaffShedule()	sendReq()	
HR Department				viewEmployee()	recieveHiringReq() publishAd()	viewApllicant() deleteApplicant()
Marketing Team	generateEventEarningReport()		viewClient() generateClientReport()			
Vice-President	reqReportFromFinMan()		generateClientNEventStat()	generateEmpUtiliSumry()		

Each and every user must be authorized to login to the system. The authorization token will be valid for 24hrs. The interface changes according to the user. So only authenticated users can access their UI.

4.2 Global control flow -

The control flow we selected is Threads since it could be both concurrent and procedural. Concurrency since many clients and staff members would be accessing at the same time and procedural because it needs a lot of approvals.

4.3 Boundary conditions -

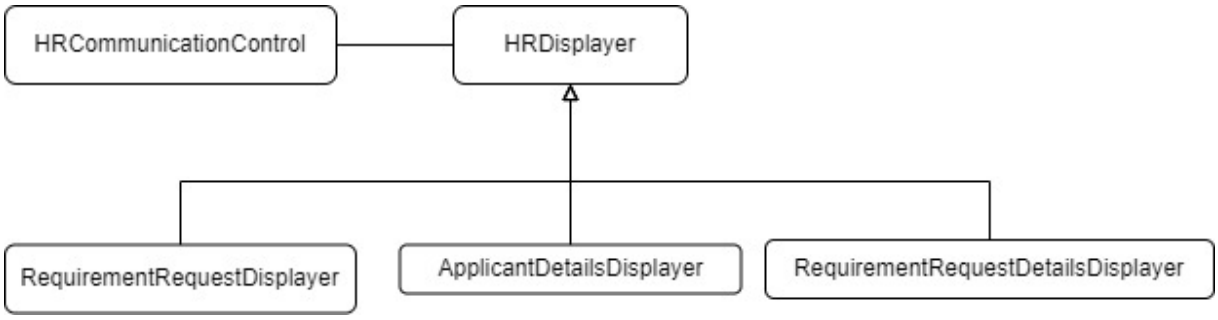
Boundary use cases

Use case name	HRSubSystem
Entry Condition	1. HR department officer successfully logs into the system.
Flow of Events	Upon Receiving Hiring Request, HR department officer executes publishAd command. 2. If the System previously was shutdown properly, it works well. Else, it might crash and notify the officer.
Exit Condition	3. Successfully posts Ad.

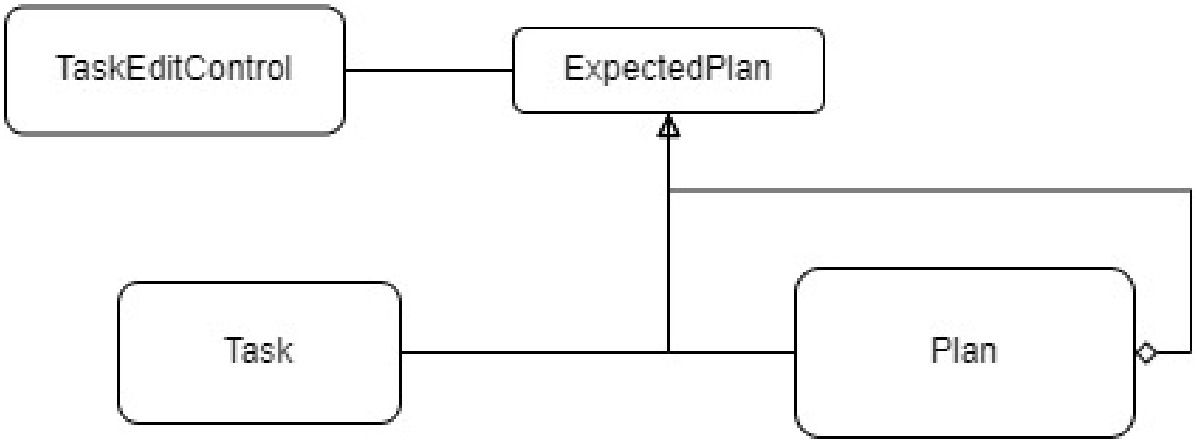
Use case name	StartCustomerServiceSubSystem
Entry Condition	1. Customer Services department officer successfully logs into the system.
Flow of Events	2. Upon client call, Customer services department officer executes initiateEventReq command. 3. If the System previously was shutdown properly, it works well. Else, it might crash and notify Customer Services officer.
Exit Condition	4. Successfully enters all the details of the client requirements.

5 Applying design patterns to designing object model for the problem -

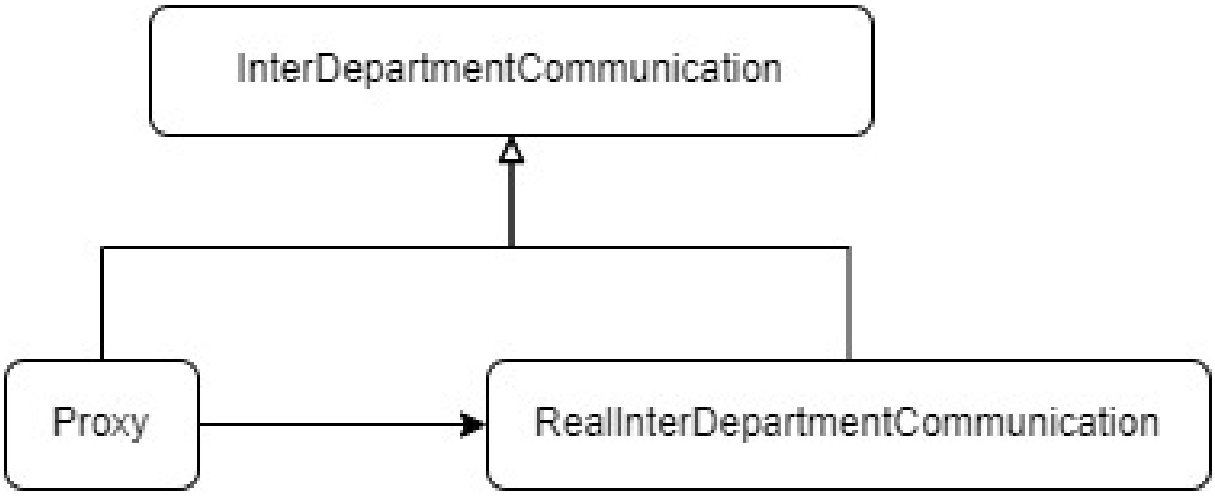
5.1 In this project we used a bridge pattern in order to develop a unified displaying interface for multiple types of objects.



5.2 We also used composite pattern in order to allow the plans for OnField teams to have sub-plans on top of individual tasks.



5.3 We also used a virtual proxy for the InterDeprtment-Communication section so that we can allow communication between departments without any database access implementations in case the main server is down and we use the reserve server which cannot store our archive database.



6 Writing contracts for noteworthy classes -

6.1

In event planning request, From date should be less than To date.
context EventPlanningReq inv:
from.before(to)

6.2

Status of the event should be set to close after successfully completing the event.
context EventPlanningReq::setStatus(close):string
post to.before(Calender.Day())

6.3

The meeting should be scheduled no earlier than in 2 days:
context Meeting::ScheduleMeeting(d):
d - today() ¿ 2*Calendar.Day()

6.4

When an expected plan is finalized, it should not be empty:
context TaskEditControl::FinalizePlan():
Not self-¿empty()

6.5

When sending a document through InterDepartmentCommunication, the sending department should not be the same as receiving department:

context InterDepartmentCommunication::SendDocument(d, sender, receiver):
sender != receiver