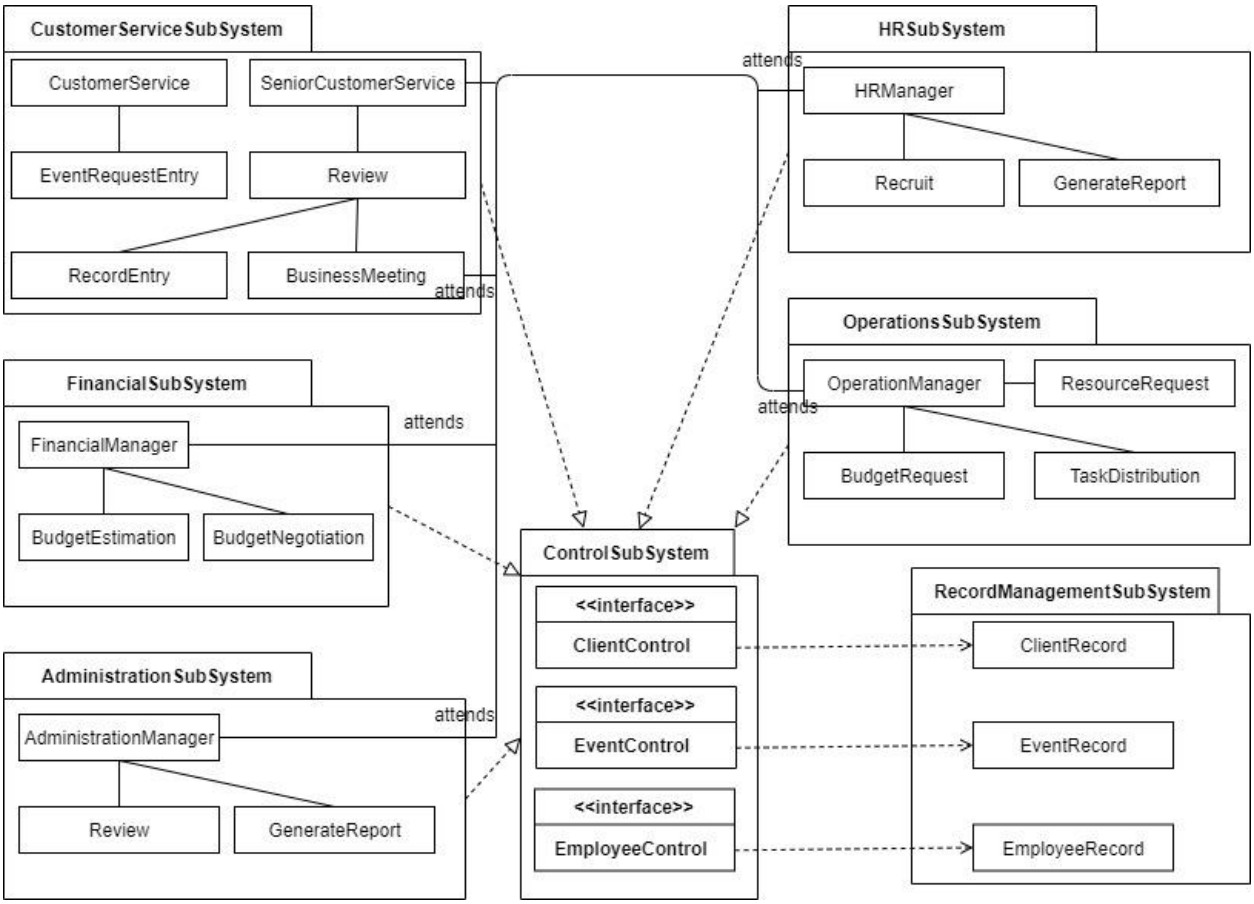
03-Oct-2021

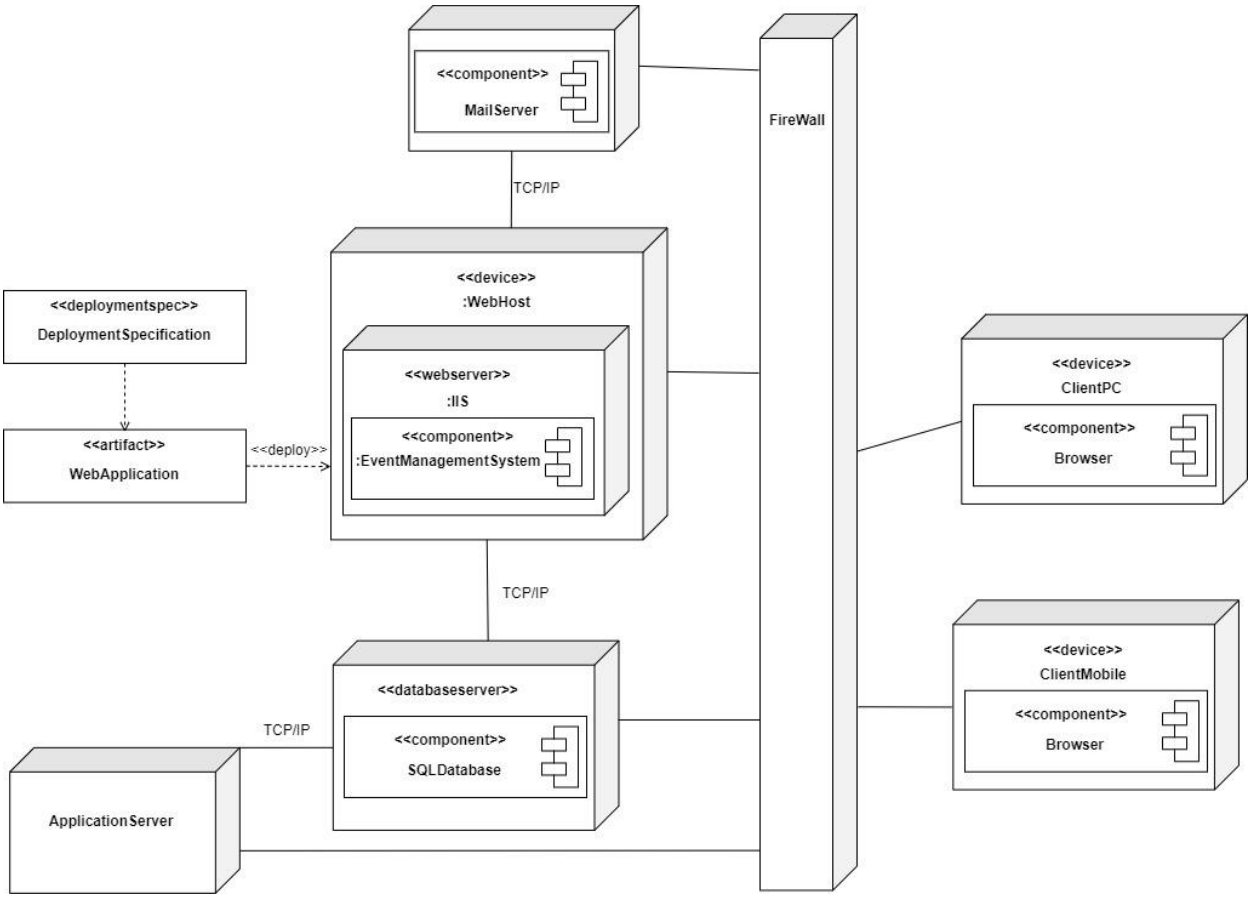
ID2207 - MODERN METHODS IN SOFTWARE ENGINEERING (HOMEWORK 4)

Submitted By: Group 24 (Rafat Khan, Ali Khalid)

1 Subsystem decomposition structure (using class diagrams)



2 UML deployment diagram



ClientPC
The user's personal PC which is pre-installed with a browser.
ClientMobile
Personal devices, like mobile phone and iPad, is pre-installed with a browser.
Firewall
All operations on the server are filtered through the firewall.
WebServer
The Web Server (IIS) in Windows Server provides a secure, easy-to-manage, modular and extensible platform for reliably hosting websites, services, and applications.
WebDeploymentSpecification
Web deployment specification lists most required hardware and software technologies.
DatabaseServer
MSSQL is well structured, fast, and supports all mainstream operating systems.
ApplicationServer
It is a type of server designed to install, operate and host associated services and applications necessary for the application. IIS acts as the application server for the application as it process both static and dynamic requests.

MailServer
The mail server handles and delivers e-mail over internal network and over the Internet.

3 Persistent Storage Solution

3.1 Persistent Objects

Client record
Event request
Event record
Employee record
Employee schedule
Event plan

3.2 Storage Management Strategy

The database subsystem stores all of the client records, event requests, event records, employee records, employee schedules, and event plans. It is important for all of this data to be accessed simultaneously and to be safe when a potential system crash occurs. The database is a relational database, since the database will potentially handle large data sets and it will execute query and stored procedures over the attributes.

4 Access control, global control flow, and boundary conditions

4.1 Access Control

Access Matrix

Objects Actors	EventRequests	ClientRecords	EventRecords	EmployeeRecords	EventPlan
OperationManager	receiveRequest() updateRequest()				initiateTask() searchPlan() viewPlan() receivePlan() updatePlan()
OperationSubTeam					viewTask() createPlan() forwardPlan()
AdministrationManager	viewEventRequest()	searchRecord() viewRecord()	searchRecord() viewRecord()	searchRecord() viewRecord() generateReport()	

	searchEventRequest() receiveEventRequest() updateEventRequest()	generateReport()	generateReport()		
FinancialManager	receiveRequest() updateRequest() estimateBudget() forwardRequest()	searchRecord() viewRecord() makeDiscount()		searchRecord() viewRecord()	
HRTeam				searchRecord() viewRecord()	
SrCustomerService	viewEventRequest() searchEventRequest() receiveRequest() updateRequest() forwardRequest()	createRecord() searchRecord() viewRecord()			
CustomerService	initiateRequest() forwardRequest()				

In order for staff to access the different objects, they must first log in to the system. This is done by the staff inputting their login credentials in the login portal. After this is done their credentials are verified and they are logged in. They can only access the objects according to the access matrix. For example, the users belong to the operation sub teams can only access the EventPlan object. They cannot access any other objects.

4.2 Global Control Flow

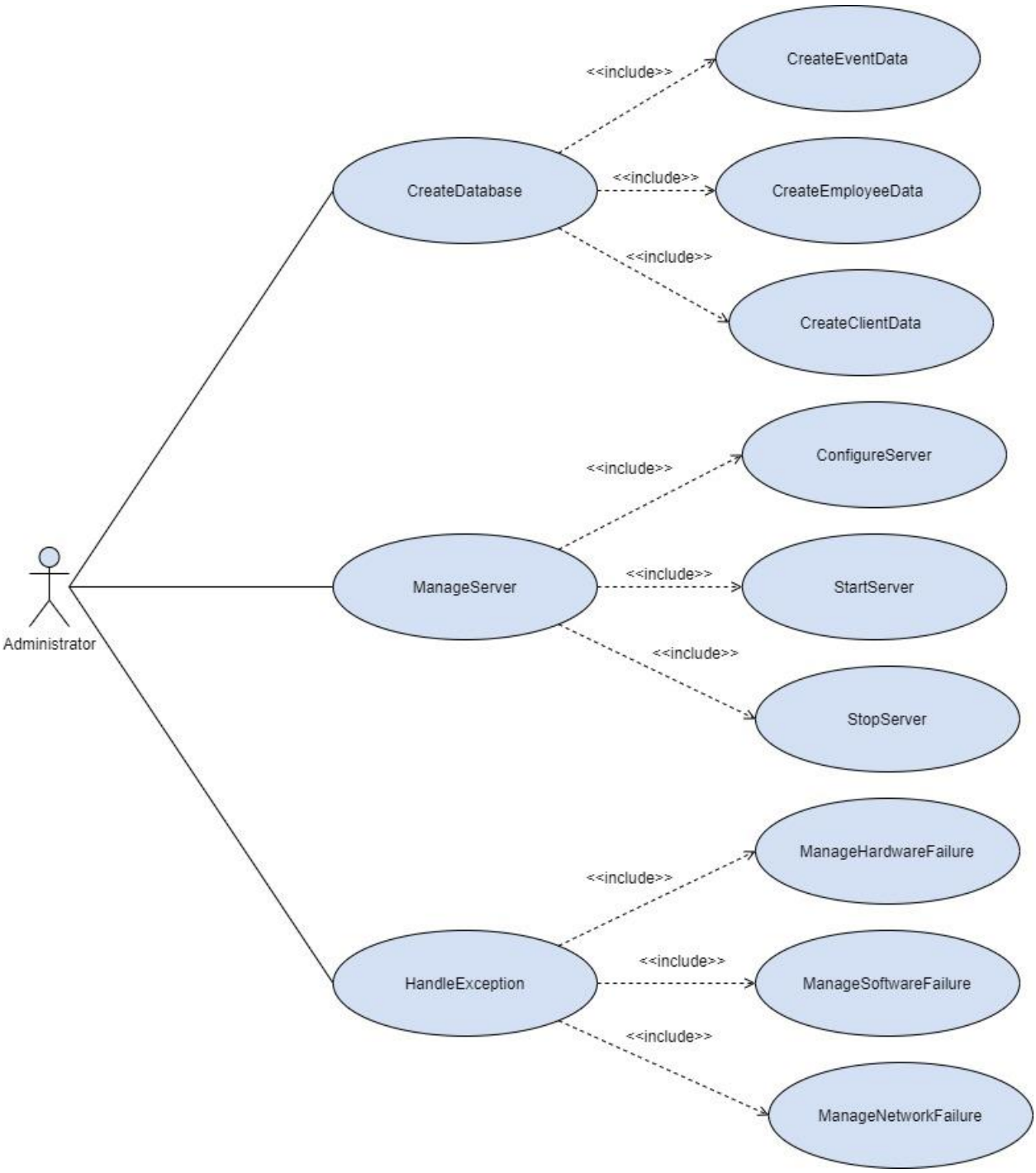
Selected Control Flow
Event-driven control

Reason
We have selected event-driven control for the system. We felt that procedure-driven control would not be ideal, since we will be implementing the system in c# using a MVC framework, which is an object-oriented language. Procedure-driven control has difficulties with object-oriented languages. Threads seemed like a good choice, but because of problems arising when debugging and testing threads we decided to use event-driven control instead. When debugging tools become better suited for threads then it would most probably be a better choice for this system.

4.3 Boundary Conditions

Name: CreateDatabase
Actors: System Administrator
Entry Conditions: 1. Administrator logins to the system. 2. System validates the credentials. 3. Administrator decided to create the database.
Exit Condition: When there is no new database that needs to be created.
Quality Conditions: The system should be available and functioning without unexpected interruptions.
Event Flow: 1. The Administrator creates a new database for the system. 2. The Administrator add different records of events, clients and employees to the database.

Name: ManageServer
Actors: System Administrator
Entry Conditions: 1. Administrator logins to the system. 2. System validates the credentials. 3. Administrator decided to manage the server.
Exit Condition: When the server does not require any operation.
Quality Conditions: The system should be available and functioning without unexpected interruptions.
Event Flow: 1. The Administrator decided to manage the server. 2. If the server had been shut down, the Administrator start it up. 3. If the server needs to be configured, the Administrator configure the server. 4. If the server is experiencing issues and needs to be shut, the Administrator can shut down or restart the server.

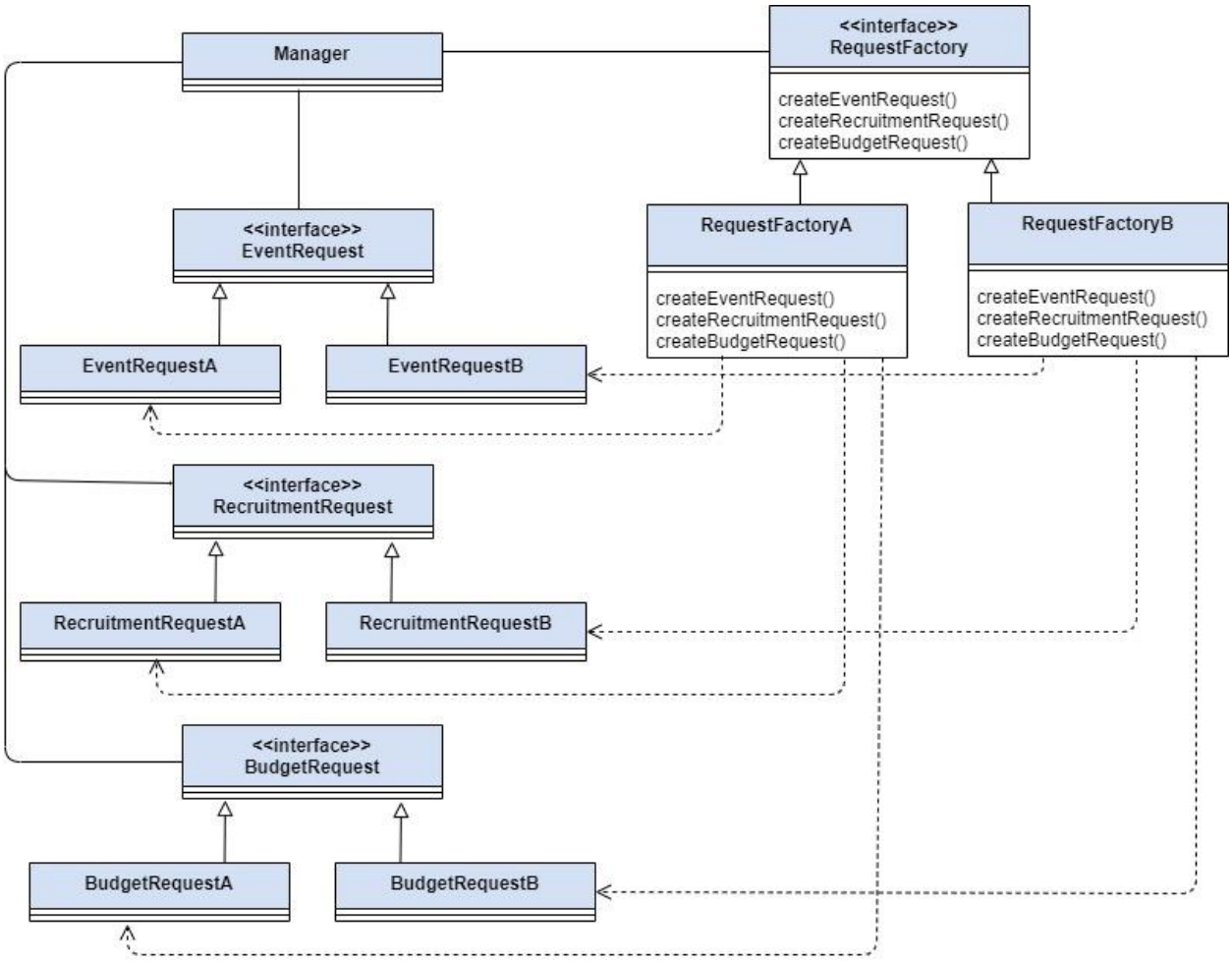

Administrator
CreateDatabase
CreateEventData
CreateEmployeeData
CreateClientData
ManageServer
ConfigureServer
StartServer
StopServer
HandleException
ManageHardwareFailure
ManageSoftwareFailure
ManageNetworkFailure
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>

5 Applying design patterns to designing object model for the problem

Selected Pattern
Abstract Factory Pattern

Definition
This design pattern can be an interface for creating families of related or dependent objects without specifying their concrete classes. We can say it is just an object maker which can create more than one type of object. The object it produces is known to the client only by that object's interface, not by the object's actual concrete implementation.

Reason for Using This Pattern
We are using this pattern because we have a requirement to create a set of related objects, or dependent objects which must be used together as families of objects. In our application we will use different form for sending requests. When materials are insufficient or there are new request, the responsible department needs to submit special request form to the relevant department.
The following UML class diagram represent the request management process:



The classes and objects participating in the above UML class diagram are as follow.

AbstractFactory (RequestFactory)
This is an interface for operations which is used to create abstract product.

ConcreteFactory (RequestFactoryA and RequestFactoryB)
This is a class which implements the AbstractFactory interface operations to create concrete products.

AbstractProduct (EventRequest, RecruitmentRequest and BudgetRequest)
This declares an interface for a type of product object

Product (Requests)
This defines a product object to be created by the corresponding concrete factory also implements the AbstractProduct interface

Client (Manager)
This is a class which uses AbstractFactory and AbstractProduct interfaces to create a family of related objects.

6 Contracts described in OCL

Customer Service Department

context CustomServiceOfficer :: redirectToSenior (EventRequest) **pre**:
EventRequest -> isFilled()
context SrCustomServiceOfficer :: organizeMeet () **pre**:
self -> contactClient()
context SrCustomServiceOfficer :: redirectToFinancial () **post**:
FinancialManager -> recvRequest()
context SrCustomServiceOfficer:: redirectToAdmin () **post**:
AdminManager -> recvRequest()
context ClientManage :: viewClient() **pre**:
ClientRecords -> isExist()
context BusinessMeeting :: startMeet() **pre**:
self -> allAttend()
context BusinessMeeting :: endMeet() **pre**:
ClientRequestDetails -> isFilled()
context EventRequest **inv**:
self.from < self.to
expectedBudget > 0.0
context ClientRequestDetails **inv**:
self.from < self.to
planBudget > 0.0

HR Department

context OperationManager :: requestRecruitment(e:Event) **pre**:
RecruitmentRequest -> isFilled()
context OperationManager :: requestRecruitment(e:Event) **post**:
HRManger -> recvRequest()
context EmployeeManage :: delEmployee(em:Empolyee) **pre**:
EmployeeRecords -> isExist()
context EmployeeManage :: addEmployee(em:Empolyee) **pre**:
! EmployeeRecords -> isExist()
context ADManage :: publishAD() **pre**:
self -> createAD()

Financial Department

context FinancialManager ::makeDiscount(c:Client, discount:Double) **pre**:
c.totalOrder > 1
context FinancialManager ::setSalary(em:Employee, sal:Double) **pre**:
EmployeeMange -> isExist(em:Employee)
sal > 0.0
context ProductionManager ::requestBudget(e:Event) **pre**:
FinancialRequest -> isFilled()
context FinancialRequest **inv**:
self.amount > 0.0

Operation Department

context OperationManager ::checkSchedule(fromdate:date,todate:date) **pre**:
NewEvent -> isExist()
context OperationManager ::requestForResource(e:Event) **pre**:
Resource -> notExist() ||
Resource -> isBusy()
context OperationManager::assignResource(em:Employee, e:event) **pre**:
Resource -> isExist()
context OperationSubTeams ::makePlan(e:Event) **pre**:
TaskDistributionRequest -> isFilled()
context OperationSubTeams ::askForBudget(e:Event) **pre**:
EstimatedBudget < RequiredBudget
AdditionalBudget > 0
context OperationManager::BudgetRequest(b:budget, e:event) **pre**:
AdditionalBudget > 0
context OperationManager::BudgetRequest(b:budget, e:event) **post**:
FinancialManger -> recvRequest()
context PlanApproval **inv**:
self -> approvePlan() || self -> rejectPlan()