同济大学学生宿舍管理系统

详细设计文档

小组成员：涂远鹏-1652262

刘铸煌-1652313

黎盛烜-1652310

指导老师：王继成

目录

1 引言

1.1 编制目的

本报告详细完成对校内运动场地管理系统的详细设计，达到指导后续软件构造的目的，同时实现和测试人员及用户的沟通。

本报告面向开发人员、测试人员及最终用户而编写，是了解系统的导航。

1.2 词汇表

词汇名称	词汇含义	备注
DM	宿舍管理系统	Dormitory manager
Bomb	云端服务器	在线云端服务器

1.3 参考资料

1）郭霖. 第一行代码[M]. 人民邮电出版社:郭霖, 2014. 69-307

2）IEEE[1471-2000]标准

3）骆斌.软件工程与计算（卷二）——软件开发的技术基础[M].机械工业出版社:骆斌, 2016.82-118，403-410

4）校内运动场地管理系统软件详细设计设计文档

2 产品概述

同济大学宿舍的外来人员登记、强开宿舍电闸申请、个人信息查询等服务都只能在一楼的宿管处进行查询，而住在高层的学生经常需要从高层到一楼进行信息登记等操作，不仅浪费时间也给学生造成较大不便，另外有时遇上宿管休息的时候，而学生进行报修申请时，必定会叫醒宿管，从而打扰宿管休息，造成学生、宿管双方的不悦。另一方面人为记录的管理方式具有易出错，较繁琐，不易更改等诸多缺点。

另外由于近年来学生数量的增加，导致了每个学年开始以及结束之际，入宿退宿的学生人数增加，每个学年入宿退宿查询工作的手工工作不仅操作缓慢耗费大量时间，排队现象严重，浪费大量时间并且造成诸多不便，其次就是出错的概率较大，登记的信息优势较为繁琐，很容易出现错误的情况，并且宿舍管理员在查询退宿入宿填写信息时也难以将如此大量的信息中的错误信息全部纠正。其次就是学校希望通过在线管理的服务削减人员方面的开支，降低人力成本，统一化宿舍管理，为全校学生提供高效无误的服务。

同济大学宿舍管理系统就是为了满足同济大学宿舍管理发展要求而开发的。它包含一个数据 集中服务器和若干个移动客户端，数据服务器将所有用户的数据集中存储便于管理与维护， 用户则通过客户端完成报修申请、电费查询等便捷服务，客户端与数据集中服务器采用 TCP/IP 协议的实时通信的方式完成数据的交换与维护。

宿舍管理系统是为同济大学开发的校内宿舍管理运营系统，开发的目标是帮助学校

进行日常的宿舍管理以及信息发布。通过该系统，用户可以查看学生的基本信息、宿舍信息等各方面的资料，能够方便的了解学生和宿舍的总体情况。该管理系统分别为学生（基本信息查询、报修申请、电费查询、入宿退宿状态查询）、宿舍管理员（学生信息查询、学生住宿管理、离返校情况统计、卫生评比、报修管理、外来人员登记、楼层公告管理）、学校人事管理部门（学生信息管理、宿管人员管理）三类对象提供对应的在线服务。

通过在线宿舍管理系统的应用，期望提高学校宿舍管理效率，降低人力管理成本，并减少人工登记所可能产生的错误，不仅为学校管理层面提供便利的在线服务，同时也为学生用户提供电费缴纳查询等便捷服务，实现学校管理运营学生宿舍利润最大化，并做到一些公示信息通知到每一个学生等在线便捷服务。

3 体系结构设计概述

参考宿舍管理系统体系结构设计文档中对体系结构设计的概述。

4 结构视角

4.1 业务逻辑层的分解

业务逻辑层的开发包图参见软件体系结构设计文档。

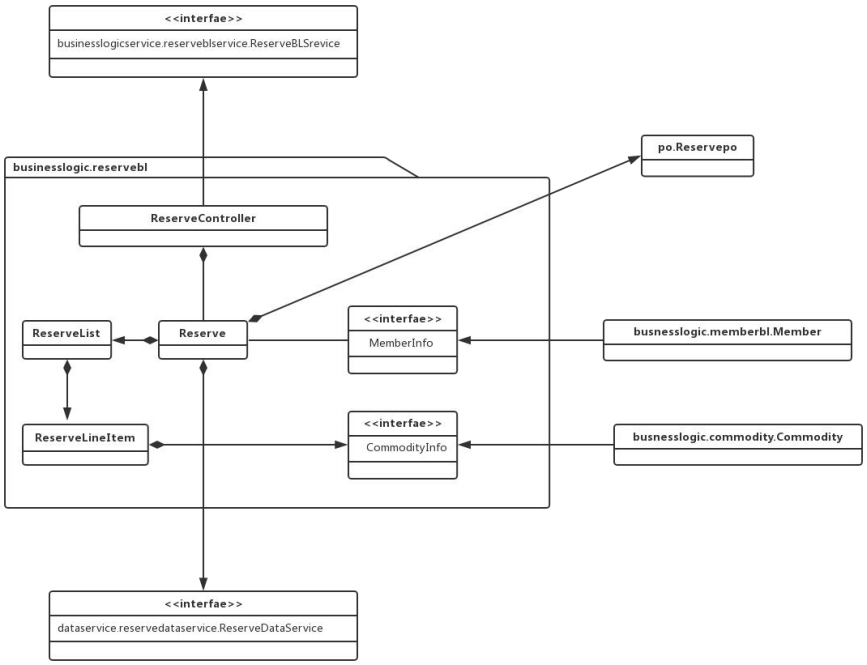
4.1.1 模块概述

Repairbl 模块承担的需求参见需求规格说明文档功能需求说明和相关非功能性需求说明。

Repairbl 模块的职责及接口参见软件体系结构设计文档。

4.1.2 整体结构

根据体系结构的设计，我们将系统分为用户界面层，业务逻辑层，数据层。每一层之间为了增加灵活性，我们会添加接口。比如用户界面层和业务逻辑层之间，我们添加了 businesslogicservice.reserveblservice.RepairBLSrevice 接口。业务逻辑层和数据层之间我们添加了 dataservice.reservedataservice. RepairDataService 接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了 RepairController，这样 RepairController 会将对报修的业务逻辑梳理委托给 Repair 对象。RepairPO 是作为预定记录的持久化对象被添加到设计模型中去的。而 RepairList 和 RepairLineItem 的添加是 CommodityInfo 的容器类。RepairLineItem 保有预定场地的数据，以及相应的计算费用的职责。而 RepairList 封装了关于 RepairLineItem 的数据集合的数据结构的秘密和计算总价的职责。CommodityInfo 和 MemberInfo 都是根据依赖倒置的原则，为了消除循环依赖而产生的接口。下图为 Repairbl 各个类的设计



Repairbl 模块的各个类的职责如下表所示。

模块	职责
LogicController	负责实现对应于登录界面所需要的服务
RepairController	负责实现报修界面需要的服务
User	系统用户的领域模型对象，拥有用户数据的姓名和密码，可以解决登录问题
Repairs	报修的领域模型对象，拥有一次报修所持有的用户，场地，信息，报修记录等信息，可以帮助完成报修界面所需要的服务

4.1.3 模块内部的接口规范

下面两个表分别为 RepairController 和 Repair 的接口规范。

RepairController 的接口规范

提供的服务（供接口）		
RepairController.addMember	语法	Public ResultMessage addMember(long id)
	前置条件	已创建一个 Reserve 对象，并且输入符合输入规则
	后置条件	调用 Reserve 领域对象的 addMember 方法
RepairController.addCommodity	语法	PublicResultMessage addCommedity(long id,long quantity)
	前置条件	已创建一个 Repair 对象，并且输入符合输入规则
	后置条件	调用 Repair 领域对象的 addCommedity 方法

RepairController.getTotal	语法	Public ResultMessage getTotal(long id,long quantity)
	前置条件	已创建一个 Repair 对象，已添加预定用户和场地信息，并且输入符合输入规则
	后置条件	调用 Reserve 领域对象的 getTotal 方法
RepairController.endReserve	语法	Public ResultMessage endReserve (long id,long quantity)
	前置条件	已创建一个 Repair 对象
	后置条件	调用 Reserve 领域对象的 endReserve 方法
RepairController.extraReserve	语法	Public ResultMessage extraReserve(long id,long quantity)
	前置条件	已创建一个 Repair 对象
	后置条件	调用 Reserve 领域对象的 extraReserve 方法
需要的服务（需接口）		
服务名		服务
Repair. addMember(long id)		加入一个预定用户对象
Repair. addCommedity(long id,long quantity)		加入一个预定场地对象
Repair. extraReserve (long id,long quantity)		加入额外商品信息
Repair. getTotal(long id,long quantity)		计算预定总费用
Repair. endReserve (long id,long quantity)		结束一次预定流程

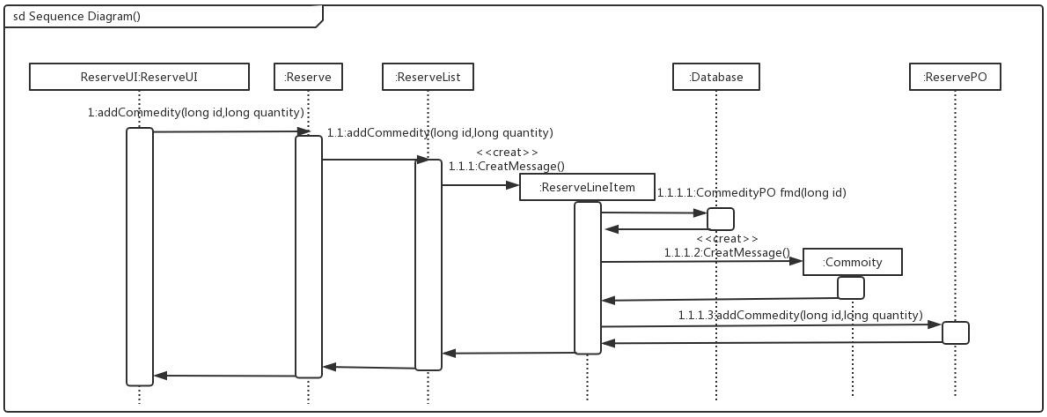
Repair 的接口规范

提供的服务（供接口）		
Repair.addMember	语法	Public ResultMessage addMember(long id)
	前置条件	启动一个报修回合
	后置条件	在一个报修回合中，增加报修的用户信息
Repair.addCommodity	语法	Public ResultMessage addCommedity(long id,long quantity)
	前置条件	启动一个报修回合
	后置条件	在一个报修回合中，增加报修的场地信息
Repair.Repaired	语法	Public ResultMessage Repaired(long id,long quantity)
	前置条件	已添加用户信息和报修场地信息
	后置条件	返回此次报修是否成功
Repair.endReserve	语法	Public ResultMessage endRepair (long id,long quantity)
	前置条件	已报修
	后置条件	结束此次报修回合，持久化更新涉及的领域对象的数据
Repair.extraRepair	语法	Public ResultMessage extraRepair(long id,long quantity)

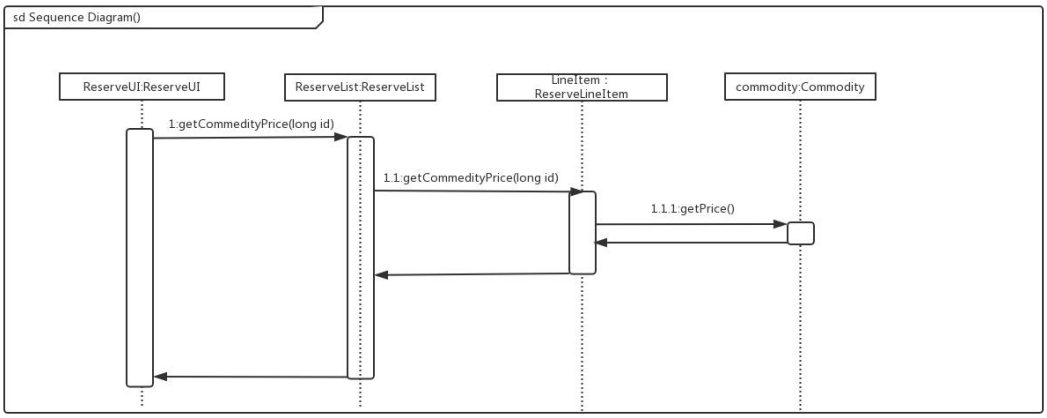
	前置条件	启动一个报修回合
	后置条件	增加额外报修信息，持久化更新涉及的领域对象的数据
需要的服务（需接口）		
服务名		服务
RepairDataservice.find(long id)		根据 id 进行查找单一持久化对象
RepairDataservice.insert(RepairPo po)		插入单一持久化对象
RepairDataservice.delete(RepairPo po)		删除单一持久化对象
RepairDataservice.update(RepairPo po)		更新单一持久化对象
RepairDataservice.init(RepairPo po)		初始化单一持久化对象
RepairDataservice.finish(RepairPo po)		结束持久化数据库的使用

4.1.4 数据层的动态模型

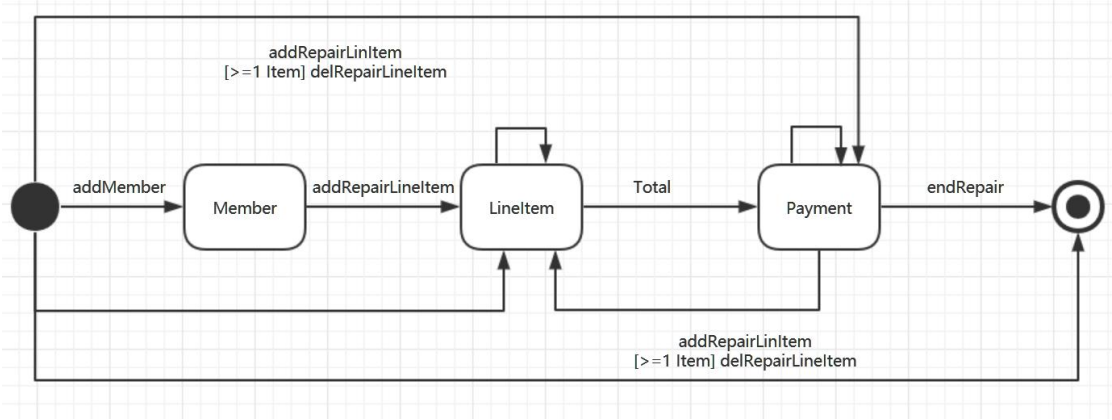
下图表示在宿舍管理系统中，单用户输入报修的场地信息后，预定业务逻辑处理的相关对象之间的协作。



下图为 Repair 领域对象想要获得报修结果时候的顺序图。



下图所示的状态图描述了 Reserve 领域对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。随着 addMember 方法被 UI 调用，Reserve 进入 Member 状态；之后通过添加场地进入 LineItem 状态。UI 也可以不输入用户账号，直接添加场地进入 LineItem 状态。



4.1.5 数据层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑有各自的控制器委托个不同的领域对象。

4.2 数据层的分解

数据层的开发包图参见软件体系结构设计文档。

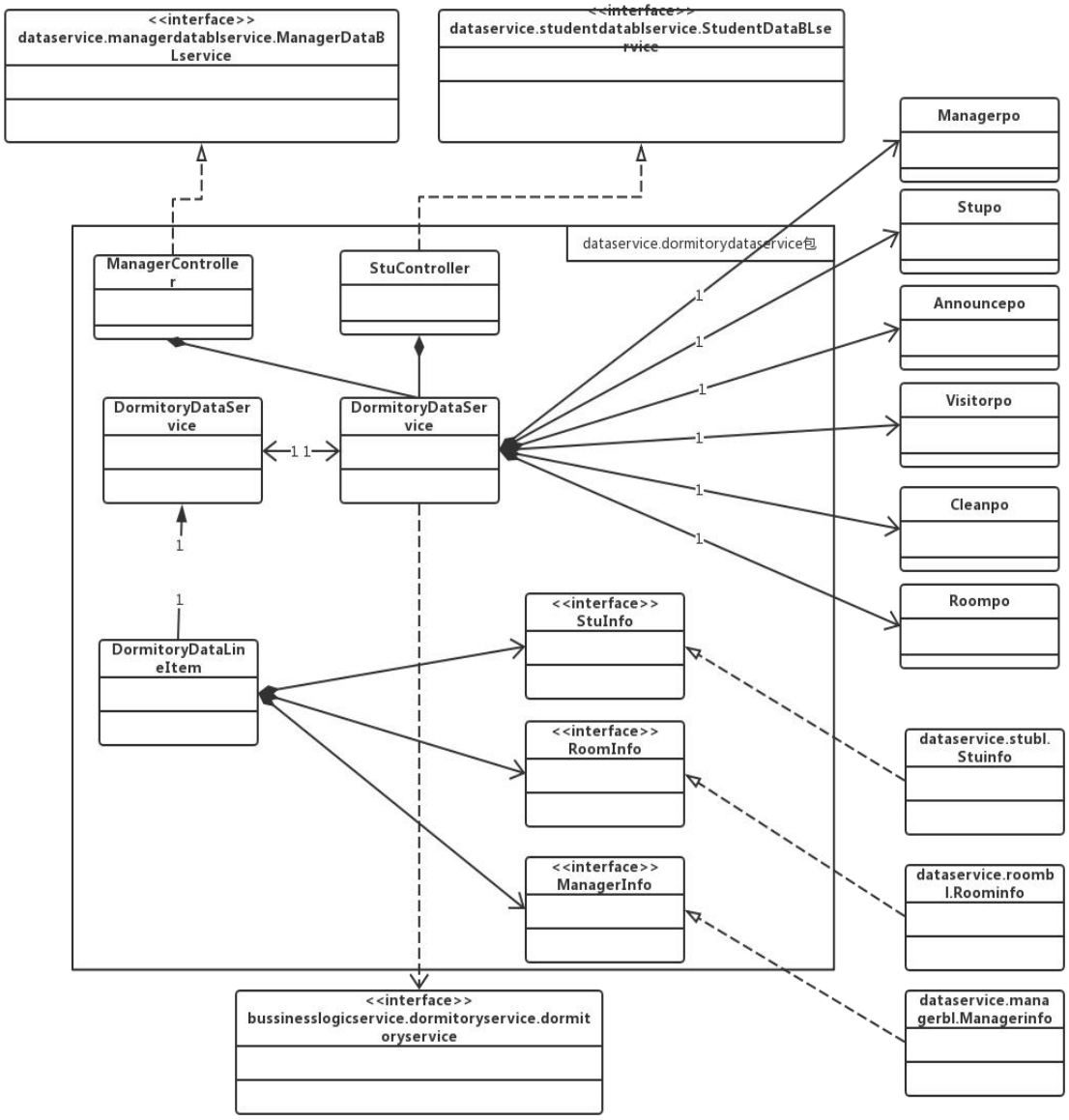
4.2.1 模块概述

DormitoryDataServicebl 模块承担的需求参见需求规格说明文档功能需求说明和相关非功能性需求说明.

DormitoryDataServicebl 模块的职责及接口参见软件体系结构设计文档。

4.2.2 整体结构

根据体系结构的设计，系统分为用户界面层，业务逻辑层，数据层。每一层之间为了增加灵活性，我们会添加接口.业务逻辑层和数据层之间我们添加了 dataservice.dormitorydataservice.DormitoryDataService 接口。为了隔离数据层职责和业务逻辑层职责，我们增加了 StuController，这样 StuController 会将对学生用户管理信息进行委托给 DormitoryDataService 对象。增加了 ManagerController，这样 ManagerController 会将对宿舍管理员用户管理信息进行委托给 DormitoryDataService 对象 DormitoryDataLineItem 保有预定具体相关所有数据包含宿舍报修申请等各类内容。而 DormitoryDataList 封装了关于 DormitoryDataLineItem 的数据集合的数据结构 的秘密。Stumemberinfo、Managermemberinfo、Topmanagermemberinfo、Repairinfo、Announceinfo 都是根据依赖倒置的原则，为了消除循环依赖而产生的接口。下图为 DormitoryDataServicebl 各个类的设计:



DormitoryDataServicebl 模块各个类的职责如下表所示:

模块	职责
StuController	负责实现对应于学生管理界面所需服务
ManagerController	负责实现对应与宿管人员管理界面所需服务
RepairController	负责宿舍报修及报修处理界面所需服务
TopManager	系统人事管理人员用户的领域模型对象，拥有人事管理人员用户数据的姓名和密码，可以解决人事管理人员用户登录问题
Student	系统学生用户的领域模型对象，拥有学生用户数据的姓名和密码，可以解决学生用户登录问题
Manager	系统宿管用户的领域模型对象，拥宿管用户数据的姓名和密码，可以解决宿管用户登录问题
StuDetail	系统学生用户的领域模型对象，拥有学生用户的所有具体信息，可以

	解决学生用户信息查询问题
ManagerDetail	系统宿管用户的领域模型对象，拥有宿管用户的所有具体信息，可以解决宿管用户信息查询问题
Repair	宿舍报修的领域模型对象，拥有一次报修所持有的学生用户信息，报修宿舍、报修时间等信息，可以帮助完成宿舍报修界面所需要的服务

4.2.3 模块内部的接口规范

下面三个表分别为 StuController、ManagerController 和 TopManager 的接口规范。

StuController 的接口规范

提供的服务（供接口）		
StuController.addStudent	语法	Public ResultMessage addStudent（long id）
	前置条件	信息输入符合输入规则
	后置条件	调用 Student 领域对象的 addStudent 方法
StuController.deleteStudent	语法	Public ResultMessage deleteStudent（long id）
	前置条件	该 Student 对象已被创建
	后置条件	调用 Student 领域对象的 deleteStudent 方法
StuController.changeStudent	语法	Public ResultMessage changeStudent（long id）
	前置条件	该 Student 对象已被创建
	后置条件	调用 Student 领域对象的 changeStudent 方法
StuController.findStudent	语法	Public ResultMessage findStudent（long id）
	前置条件	该 Student 对象已被创建
	后置条件	调用 Student 领域对象的 findStudent 方法
StuController.updateStudent	语法	Public ResultMessage UpdateStudent（long id）
	前置条件	该 Student 对象已被创建
	后置条件	调用 Student 领域对象的 updateStudent 方法

ManagerController 的接口规范

提供的服务（供接口）		
ManagerController.addManager	语法	Public ResultMessage addManager（long id）
	前置条件	信息输入符合输入规则
	后置条件	调用 Manager 领域对象的 addManager 方法
ManagerController.deleteManager	语法	Public ResultMessage deleteManager（long id）
	前置条件	该 Manager 对象已被创建
	后置条件	调用 Manager 领域对象的 deleteManager 方法
ManagerController.changeManager	语法	Public ResultMessage changeManager （long id）
	前置条件	该 Manager 对象已被创建
	后置条件	调用Manager领域对象的changeManager方法
ManagerController.findManager	语法	Public ResultMessage findManager（long id）
	前置条件	该 Manager 对象已被创建
	后置条件	调用 Manager 领域对象的 findManager 方法
ManagerController.updateManager	语法	Public ResultMessage updateManager(long id)
	前置条件	该 Manager 对象已被创建

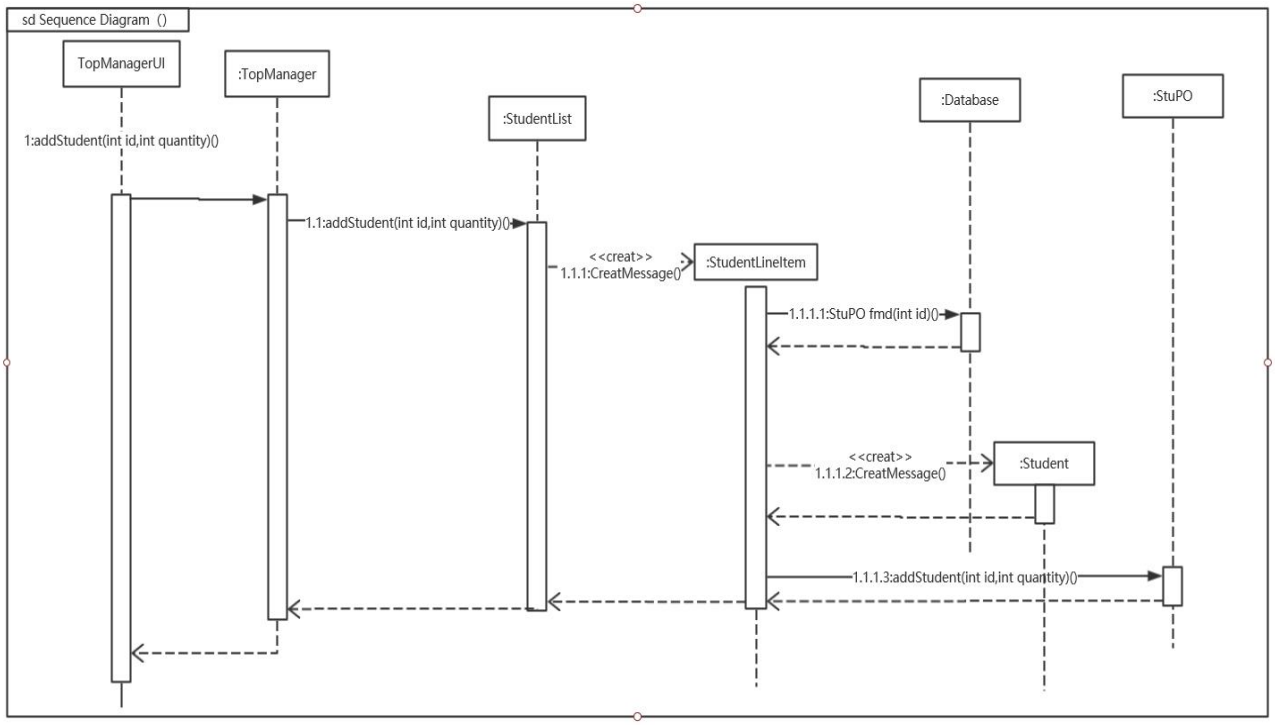
	后置条件	调用 Manager 领域对象的 updateManager 方法

TopManager 的接口规范

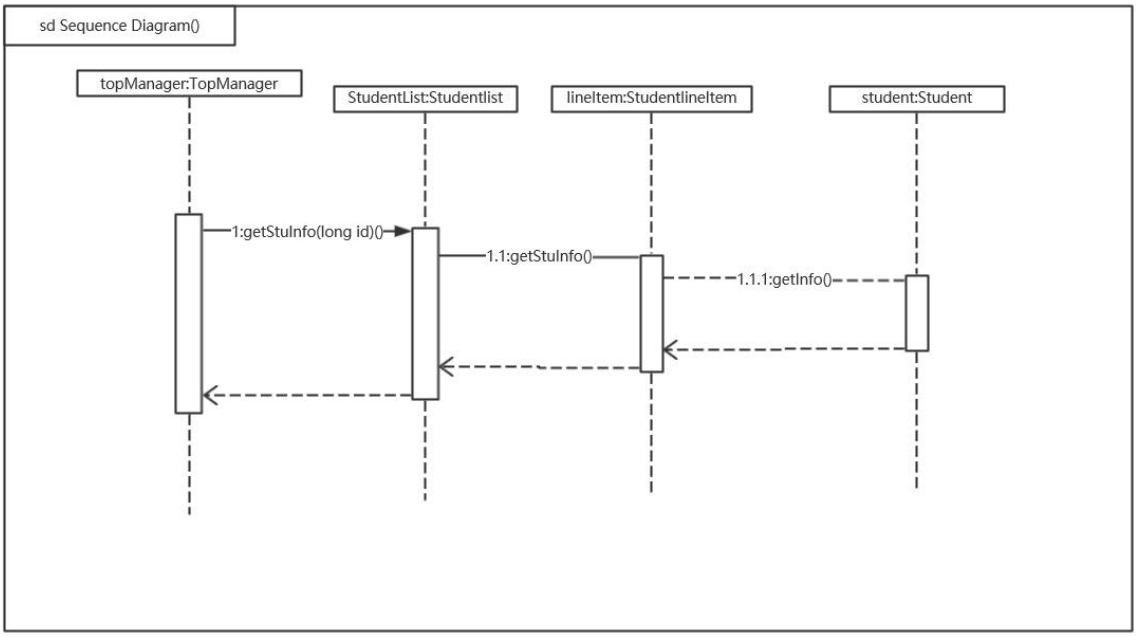
提供的服务（供接口）		
TopManager.addManager	语法	Public ResultMessage addManager（long id）
	前置条件	启动一个宿管管理回合
	后置条件	增加一个宿管用户对象
TopManager.deleteManager	语法	Public ResultMessage deleteManager（long id）
	前置条件	启动一个宿管用户管理回合且该宿管用户 id 已存在
	后置条件	删除该 id 宿管用户对象信息
TopManager.changeManager	语法	Public ResultMessage changeManager （long id）
	前置条件	启动一个宿管用户管理回合且该宿管用户 id 已存在
	后置条件	修改宿管用户信息
TopManager.findManager	语法	Public ResultMessage findManager（long id）
	前置条件	该宿管用户 id 已存在
	后置条件	查找该 id 宿管用户对象信息返回该宿管用户 id
TopManager.updateManager	语法	Public ResultMessage updateManager（long id）
	前置条件	启动一个宿管用户管理回合且修改内容合规
	后置条件	更新该 id 宿管用户对象信息
TopManager.addStudent	语法	Public ResultMessage addStudent（long id）
	前置条件	启动一个学生管理回合
	后置条件	增加一个学生用户对象
TopManager.deleteStudent	语法	Public ResultMessage deleteStudent（long id）
	前置条件	启动一个学生用户管理回合且该学生用户 id 已存在
	后置条件	删除该 id 学生用户对象信息
TopManager.changeStudent	语法	Public ResultMessage changeStudent（long id）
	前置条件	启动一个学生用户管理回合且该学生用户 id 已存在
	后置条件	修改学生用户信息
TopManager.findStudent	语法	Public ResultMessage findStudent（long id）
	前置条件	该学生用户 id 已存在
	后置条件	查找该 id 学生用户对象信息返回该学生用户 id
TopManager.updateStudent	语法	Public ResultMessage updateStudent（long id）
	前置条件	启动一个学生用户管理回合且修改内容合规
	后置条件	更新该 id 学生用户对象信息

4.2.4 数据层的动态模型

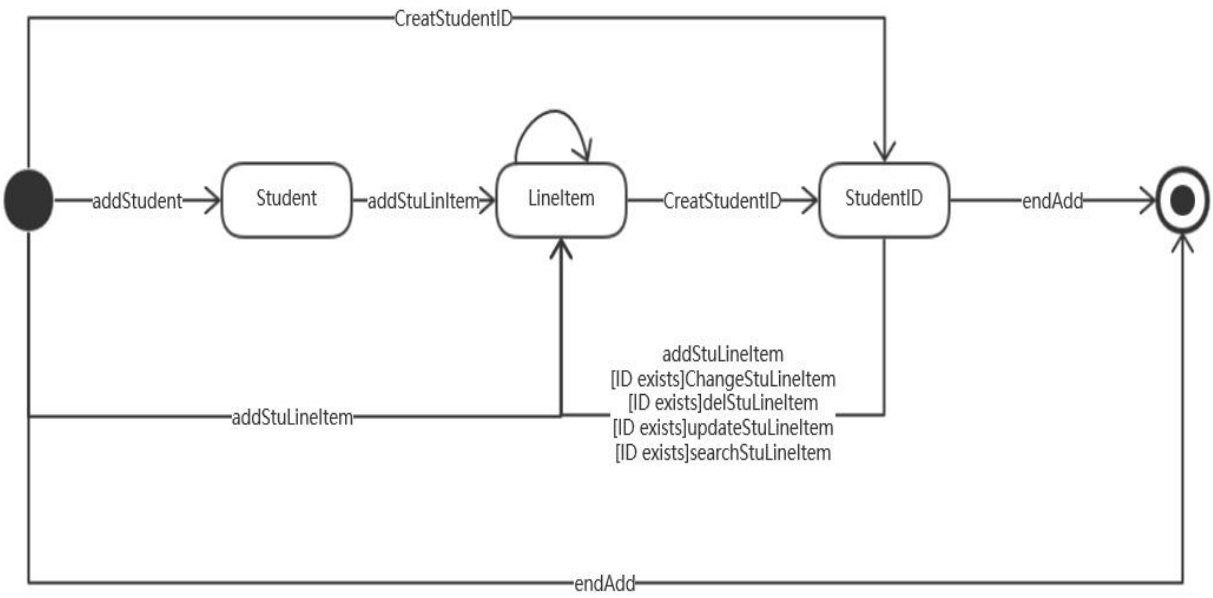
下图为同济大学宿舍管理系统中，人事管理人员输入学生管理操作后，学生用户管理所产生的个对象间的协作关系：



下图为人事管理人员 TopManager 领域对象想要进行查找用户信息时对应的顺序图关系：



下图所示的状态图描述了 TopManager 领域对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。随着 addStudent 方法被 UI 调用，TopManager 进入 Student 状态；之后通过输入人事管理人员账户密码进入 StudentinfoLineItem 状态：

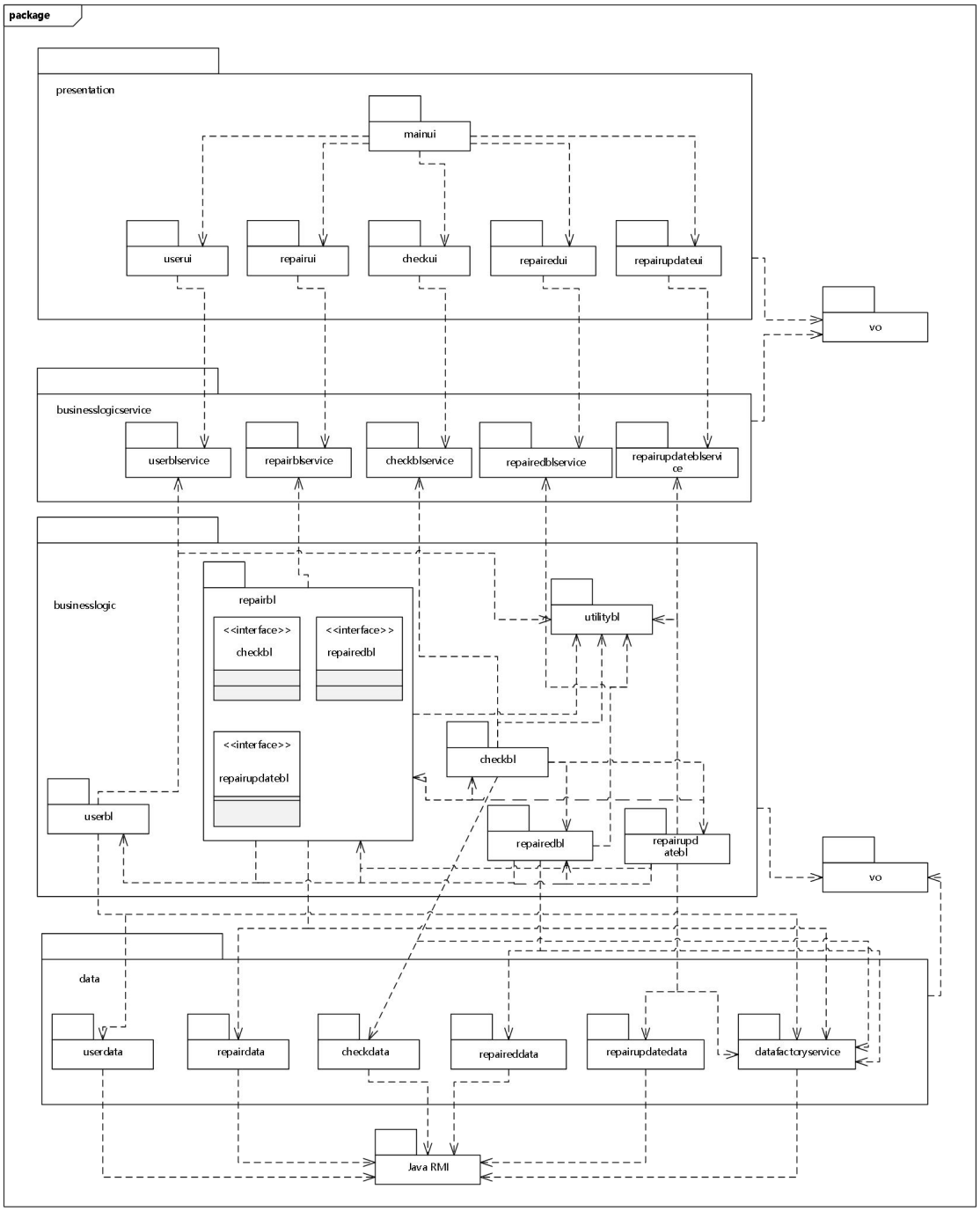


4.2.5 数据层的设计原理

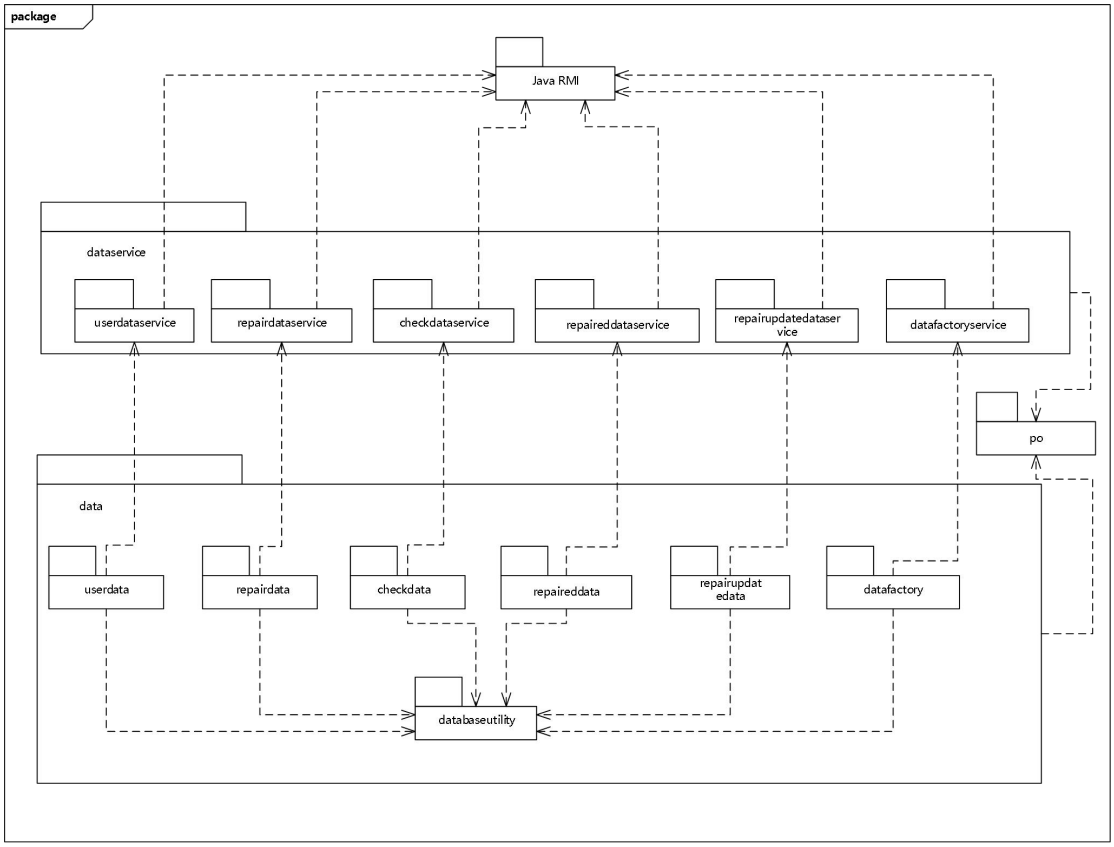
利用委托式控制风格，每个界面需要访问的业务逻辑有各自的控制器委托个不同的领域对象.

5 依赖视角

下面两图是客户端和服务器端各自的包之间的依赖关系。


package
presentation
mainui
userui
repairui
checkui
repairedui
repairupdateui
vo
businesslogicservice
userblservice
repairblservice
checkblservice
repairedblservice
repairupdateblservice
businesslogic
repairbl
<<interface>>
checkbl
<<interface>>
repairedbl
<<interface>>
repairupdatebl
utilitybl
checkbl
userbl
repairedbl
repairupdatebl
vo
data
userdata
repairdata
checkdata
repaireddata
repairupdatedata
datafactoryservice
Java RMI

客户端包图


package
Java RMI
dataservice
userdataservice
repairdataservice
checkdataservice
repaireddataservice
repairupdatedataser
vice
datafactoryservice
po
data
userdata
repairdata
checkdata
repaireddata
repairupdat
edata
datafactory
databaseutility

服务器端包图