Gitmining软件详细设计描述文档

1. 引言

1.1编织目的

本报告详细完成对连锁商店管理系统的详细设计，大道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. 产品概述

参考用例文档和需求规格说明文档中对产品的概括描述。

1. 结构视角

3.1业务逻辑层的分解

3.1.1Repository模块

（1）模块概述

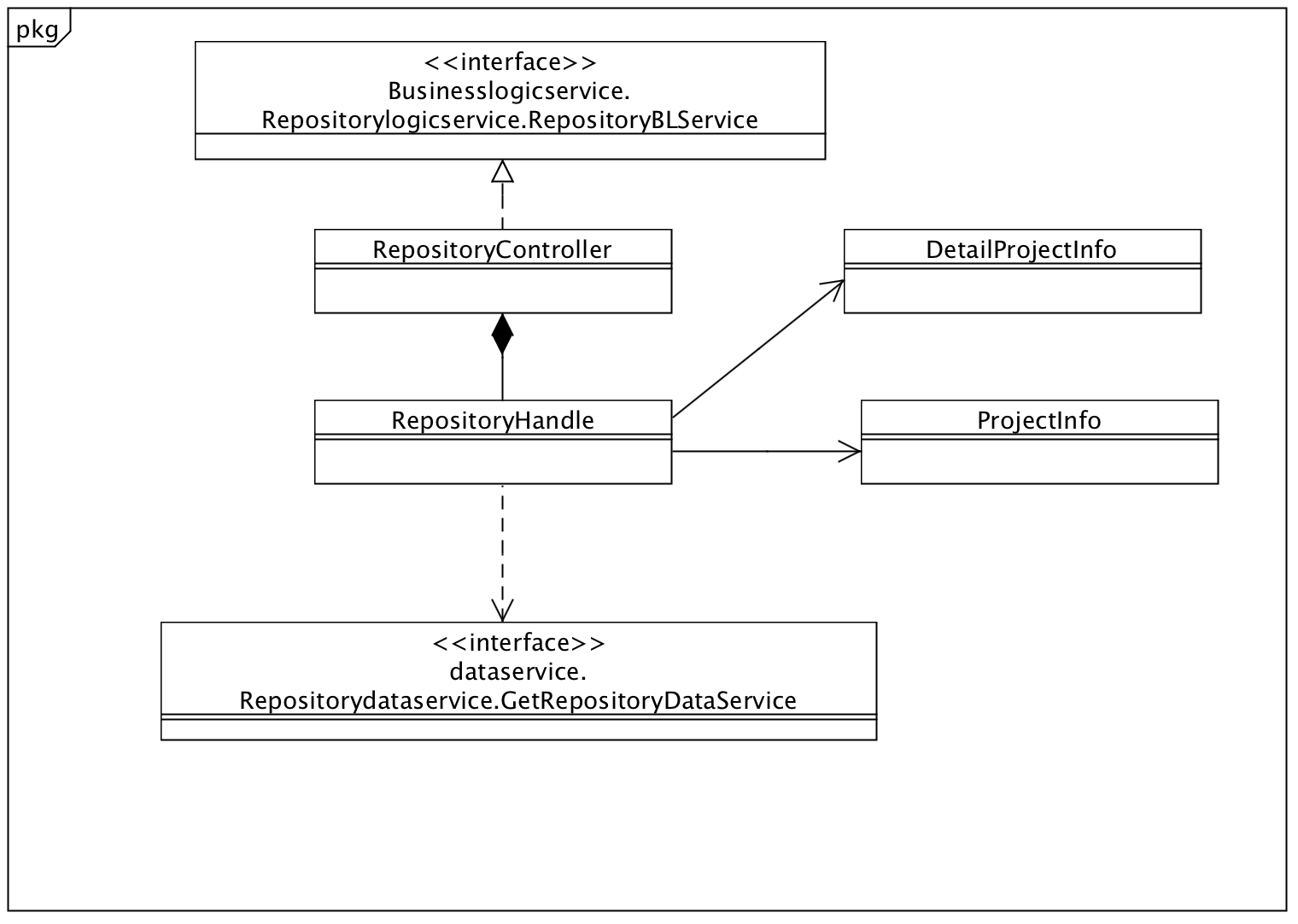
Repository模块承担向展示层提供相应的显示数据

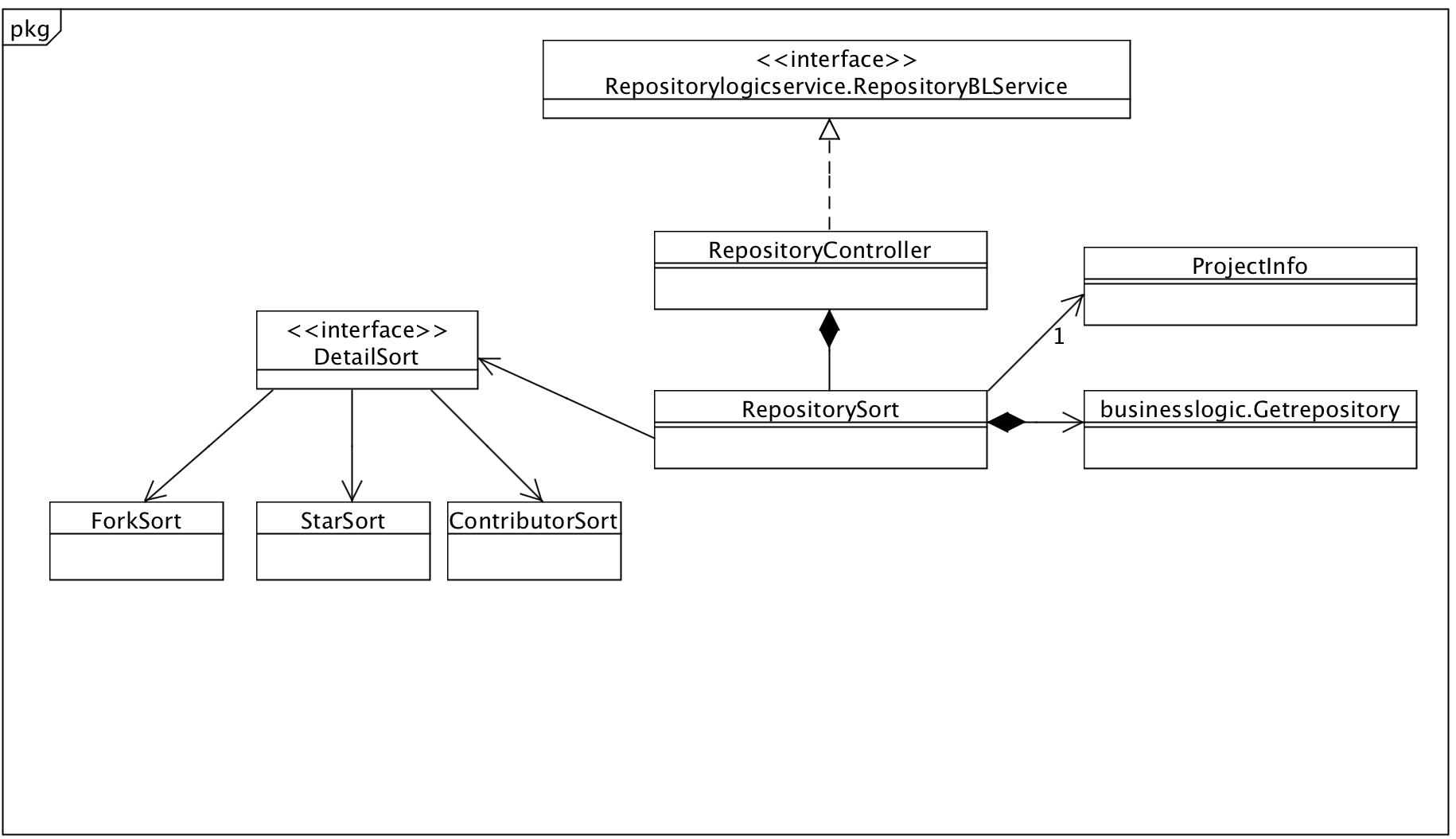
Repository模块的职责及接口参见代码注释

（2）整体结构

根据体系结构的设计，我们将系统分为展示层、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。比如展示层与业务逻辑层之间，我们添加Businesslogicservice.Repositorylogicservice. GetRepositoryBLService接口。业务逻辑层与数据层之间我们添加dataservice.

Repositorydataservice.GetRepositoryDataService接口。





Repository模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| RepositoryController | 隔离展示层与逻辑层，提供Repository相关的操作 |
| RepositoryHandle | 负责从数据层取得粗略的项目信息列表和某一详细的项目信息;和通过关键字搜索相关数据 |
| RepositorySort | 负责对项目列表进行排序 |

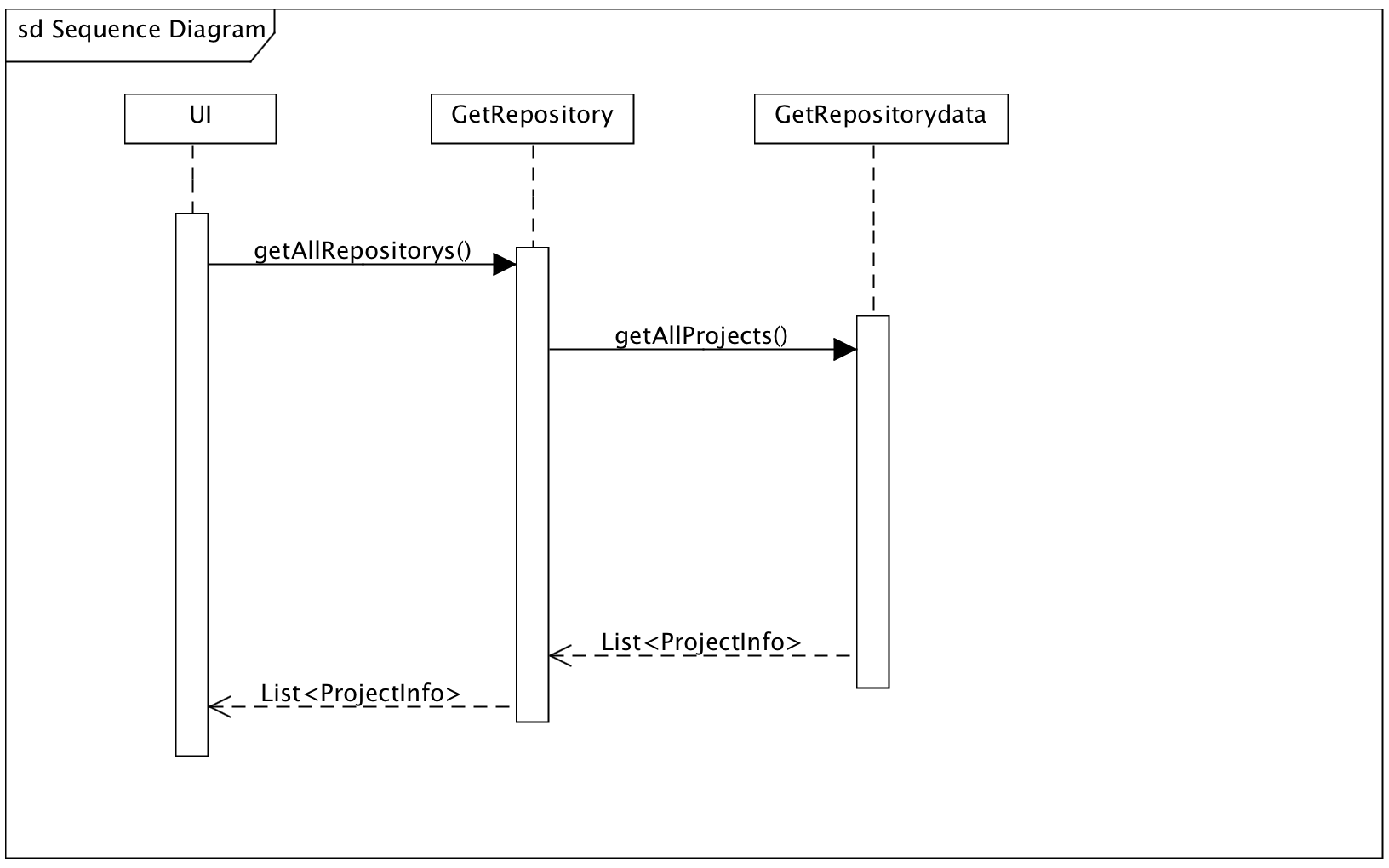
（3）模块内部类的接口规范

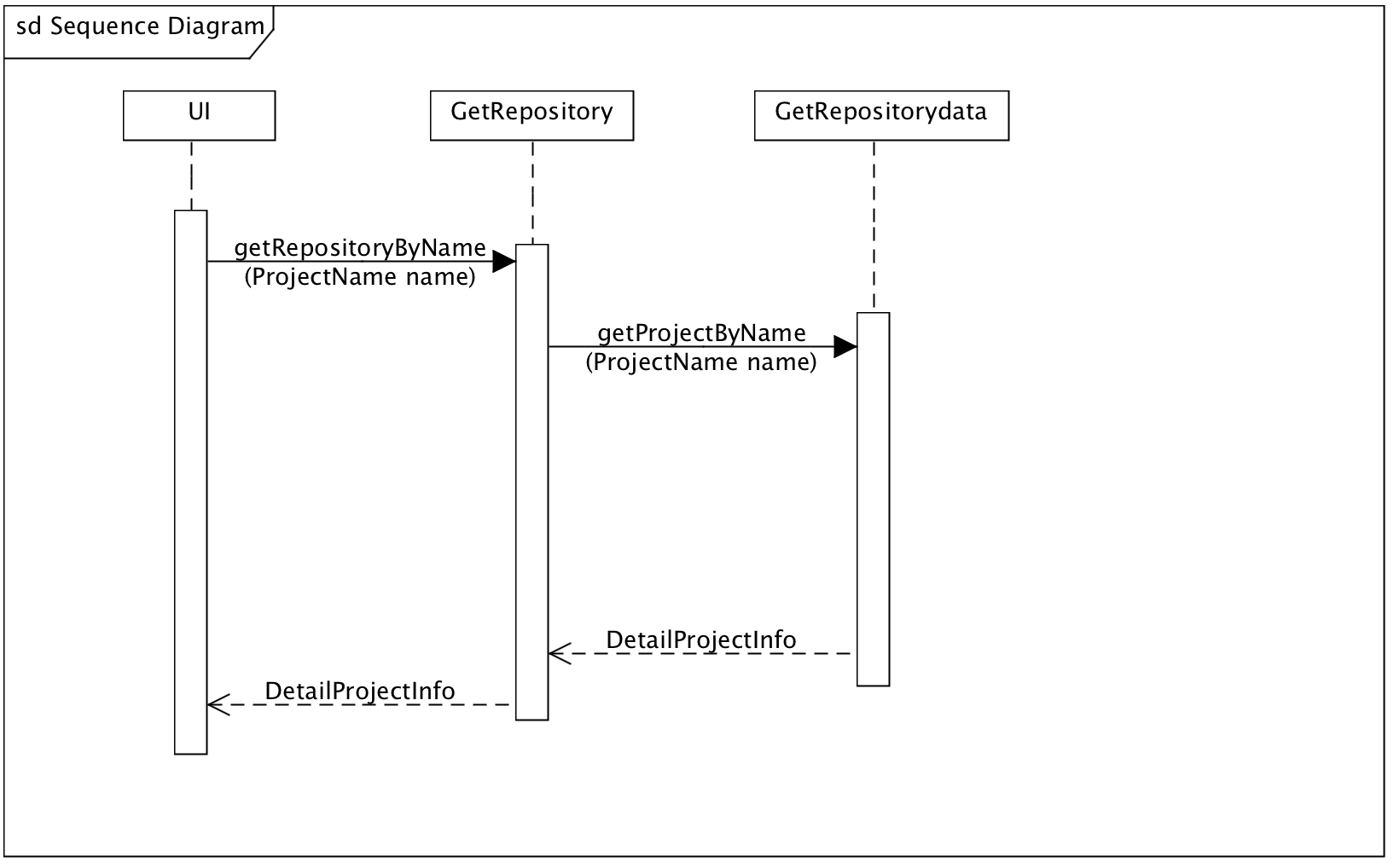
Repository模块内的相关接口

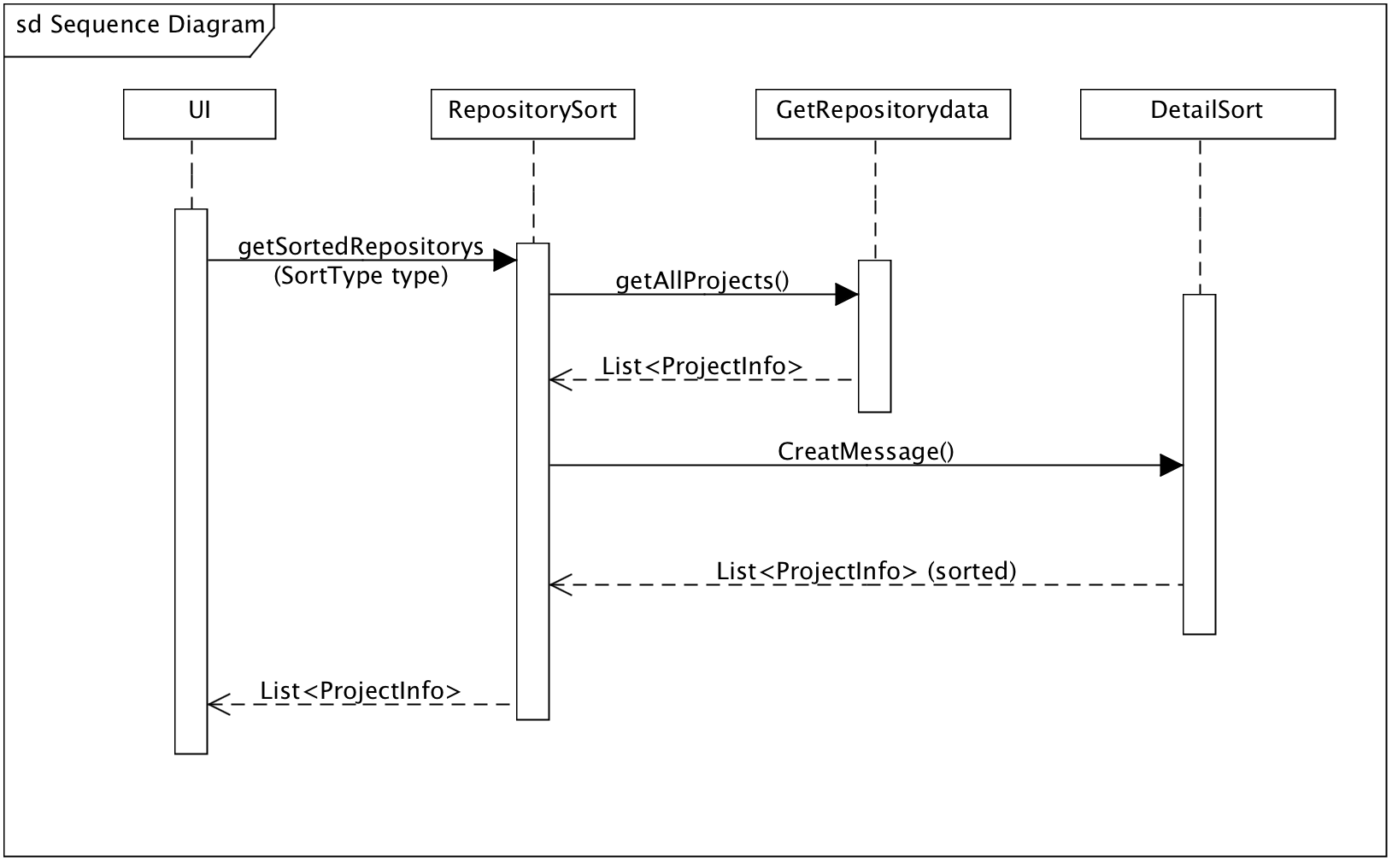
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 提供的服务（供接口） | | | | |
| RepositoryHandle.getAllRepositorys | 语法 | | Public List<ProjectInfo> getAllRepositorys() | |
| 前置条件 | | 创建Get Repository对象 | |
| 后置条件 | | 返回所有相关的项目信息列表 | |
| RepositoryHandle.getRepositoryByName | 语法 | | Public DetailProjectInfo getRepositoryByName(ProjectName name) | |
| 前置条件 | | 创建了对象，并传递了一个项目名 | |
| 后置条件 | | 如果name = null或者name对应的项目不存在，返回null。否则，返回该项目名对应的详细项目信息 | |
| RepositoryHandle.searchRepositorys | 语法 | | Public List<ProjectInfo> searchRepositorys(String key) | |
| 前置条件 | | 创建了对象，并传递了一个关键字 | |
| 后置条件 | | 如果无法找到与key(可能为null)匹配的项目信息，返回null。否则，返回所有与该key匹配的项目信息 | |
| RepositorySort.getSortedRepositorys | 语法 | | Public List<ProjectInfo> getSortedRepositorys(SortType type) | |
| 前置条件 | | 创建对象，传递排序方法 | |
| 后置条件 | | 如果type = null，则返回null。否则，返回以type为排序依据的已经排好序的所有项目信息 | |
| 需要的服务（需接口） | | | | |
| RepositoryData.getAllProjects | | 语法 | | Public List<ProjectInfo> getAllProjects() |
| 前置条件 | | 创建对象 |
| 后置条件 | | 返回所有相关的项目信息表（粗略的） |
| RepositoryData.getProjectByName | | 语法 | | Public DetailProjectInfo getProjectByName(ProjectName name) |
| 前置条件 | | 创建对象，传递项目名 |
| 后置条件 | | 如果name = null或者name对应的项目不存在，返回null。否则，返回该项目名对应的详细项目信息 |

（4）业务逻辑层的动态模型

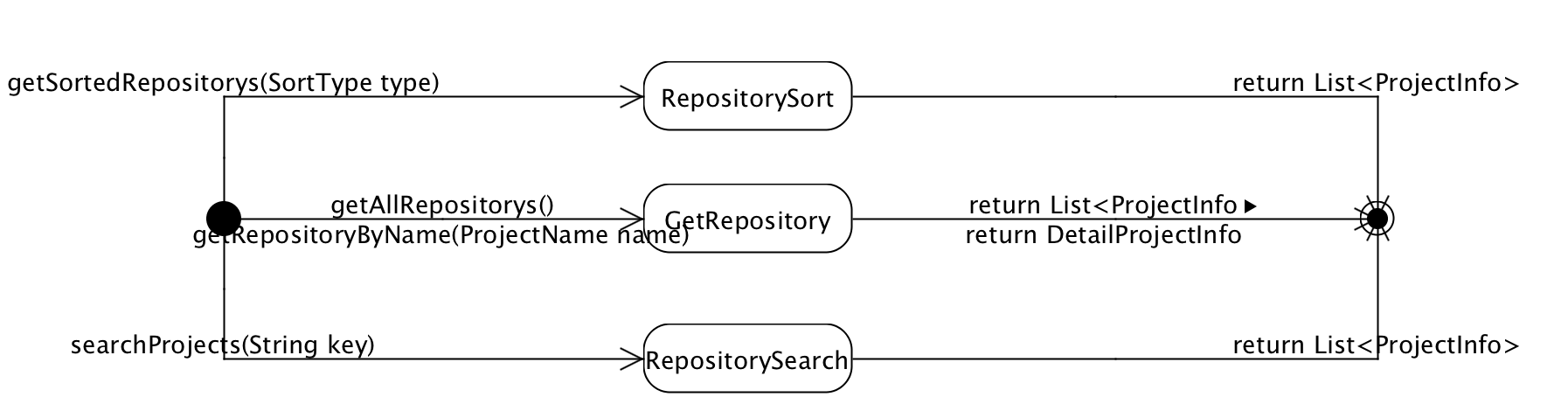
下图表明Repository的相关对象之间的协作







下图描述了Repositroy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。



* + 1. User模块

（1）模块概述

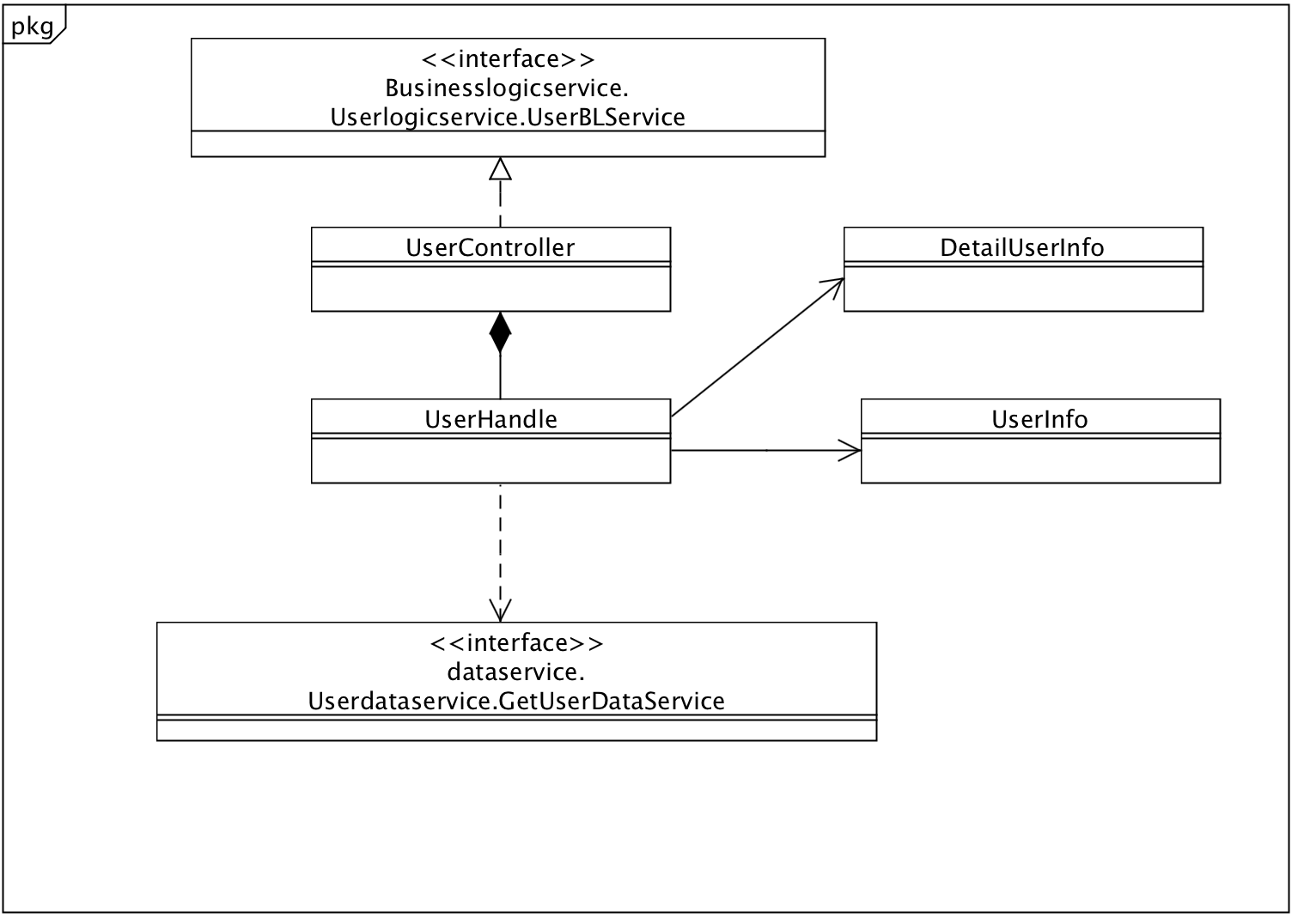
User模块承担向展示层提供用户的相应的显示数据

User模块的职责及接口参见代码注释

（2）整体结构

根据体系结构的设计，我们将系统分为展示层、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。比如展示层与业务逻辑层之间，我们添加Businesslogicservice.Userlogicservice. GetUsersBLService接口。业务逻辑层与数据层之间我们添加dataservice.

Repositorydataservice.GetUserDataService接口。



User模块各个类的职责

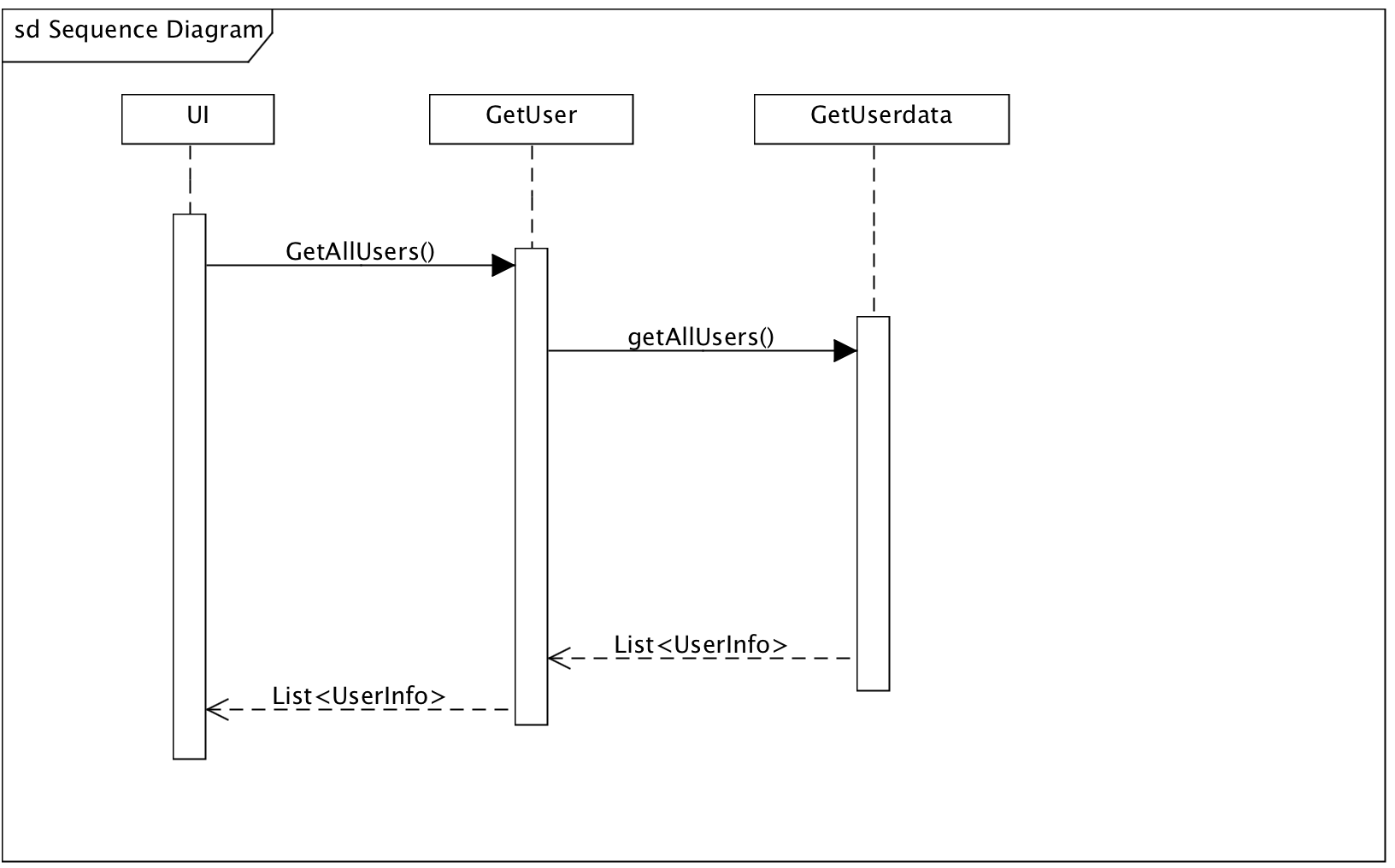
|  |  |
| --- | --- |
| 模块 | 职责 |
| UserController | 隔离展示层与逻辑层，提供User相关的操作 |
| UserHandle | 负责从数据层取得粗略的用户信息列表和某一用户的项目信息和负责通过关键字在全部的用户列表里搜索相关的用户信息 |

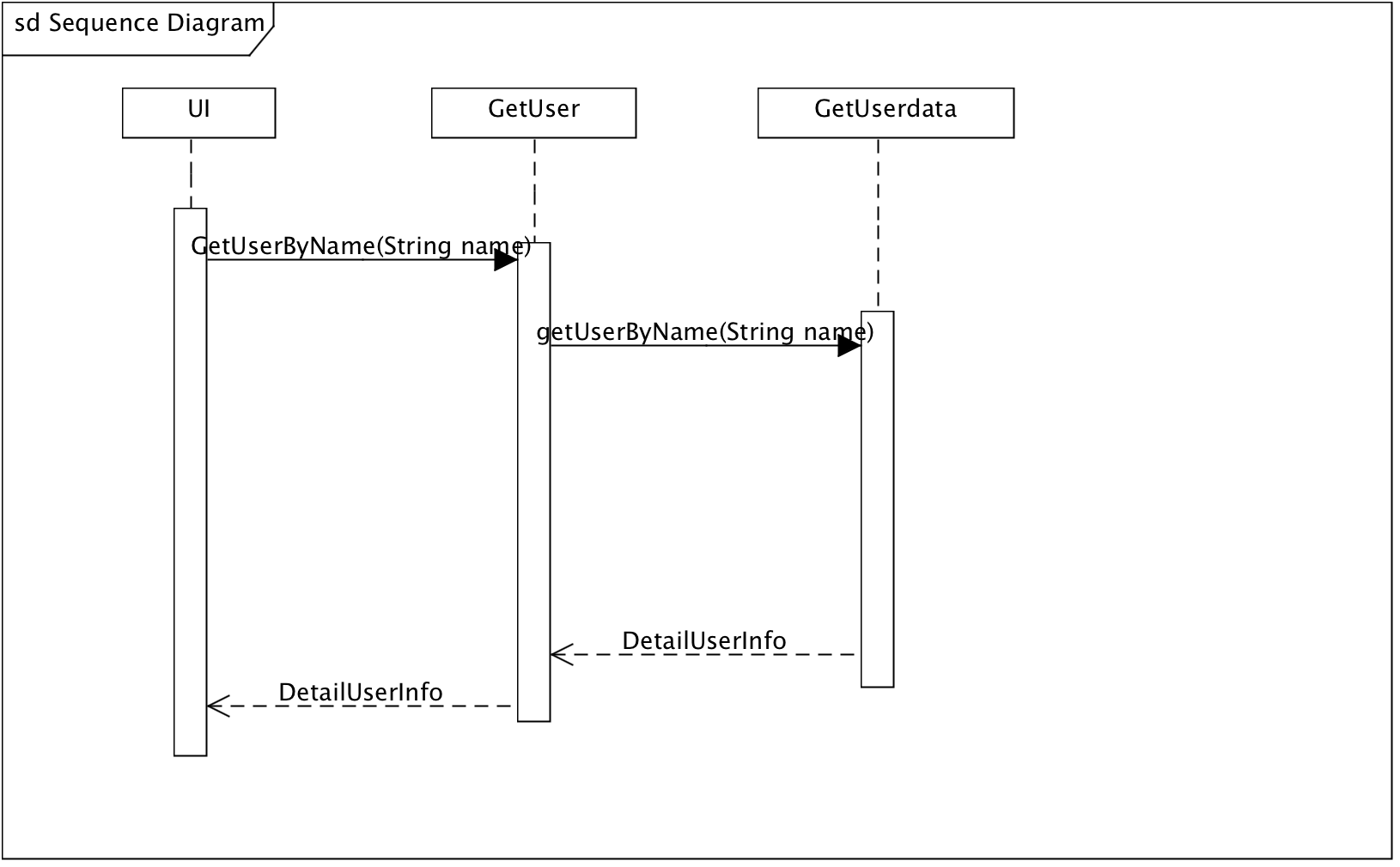
（3）模块内部类的接口规范

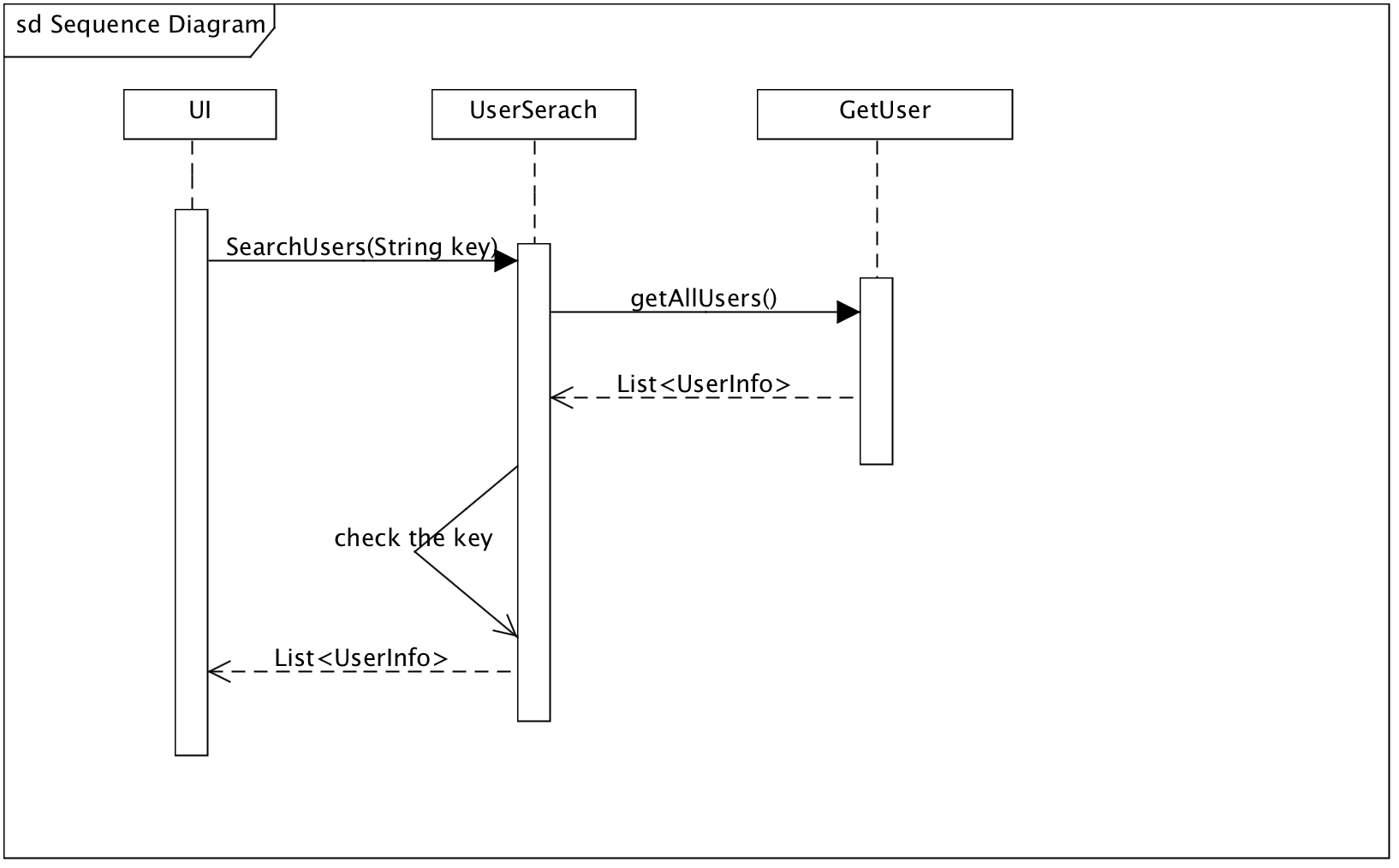
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Userhandle.GetAllUsers | 语法 | Public List<UserInfo> getAllUsers() |
| 前置条件 | 创建对象 |
| 后置条件 | 返回所有相关的用户信息 |
| UserHandle.getUserByName | 语法 | Public DetailUserInfo getUserByName(String name) |
| 前置条件 | 创建对象，传递用户名 |
| 后置条件 | 如果name = null或者name对应的用户不存在，返回null。否则，返回该项目名对应的用户详细信息 |
| UserHandle.searchUsers | 语法 | Public List<UserInfo> searchUsers(String key) |
| 前置条件 | 创建对象，传递关键字 |
| 后置条件 | 如果无法找到与key(可能为null)匹配的用户信息，返回null。否则，返回所有与key匹配的用户信息 |
| 需要的服务（需接口） | | |
| UserData.getAllUsers | 语法 | Public List<UserInfo> getAllUsers() |
| 前置条件 | 创建对象 |
| 后置条件 | 返回所有相关的用户信息表（粗略的） |
| UserData.getUserByName | 语法 | Public DetailUserInfo getUserByName(String name) |
| 前置条件 | 创建对象 |
| 后置条件 | 如果name = null或者name对应的用户不存在，返回null。否则，返回该项目名对应的详细用户信息 |

（4）业务逻辑层的动态模型

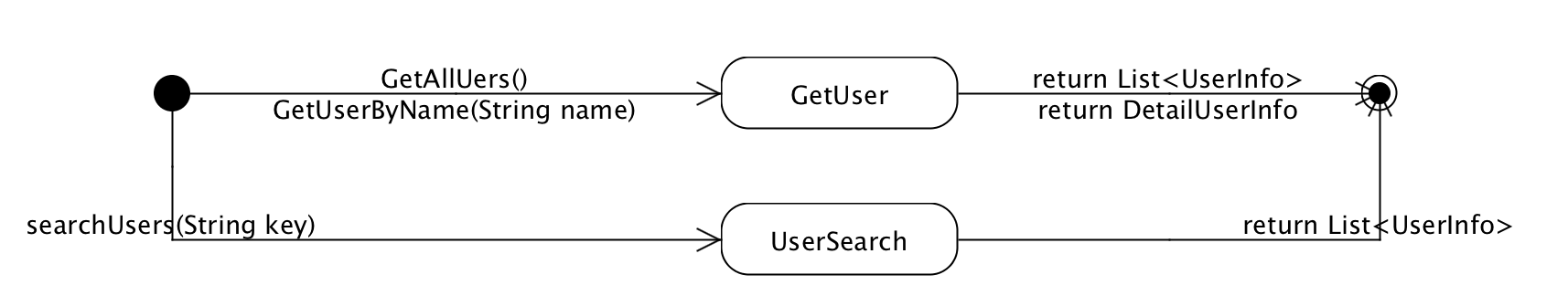
下图表明User的相关对象之间的协作







下图描述了User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。



（5）业务逻辑层的设计原理

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

1. 依赖视角

下图为各个包之间的依赖关系

