ZTE中兴

当 DDD 遇上 DCI

张晓龙

中兴通讯资深软件架构师

Geekbang. 极客邦科技 InfoQ

极客邦科技
会议推荐2019

ArchSummit 深圳
全球架构师峰会
大会：7月12–13日
培训：7月14–15日

ArchSummit 北京
全球架构师峰会
大会：12月6–7日
培训：12月8–9日

5月 6月 7月 10月 11月 12月

QCon 北京
全球软件开发大会
大会：5月6–8日
培训：5月9–10日

QCon 广州
全球软件开发大会
培训：5月25–26日
大会：5月27–28日

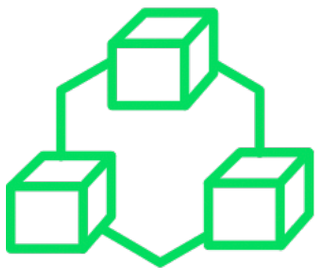
GTLC GLOBAL TECH LEADERSHIP CONFERENCE 上海
技术领导力峰会
时间：6月14–15日

GMTC 北京
全球大前端技术大会
大会：6月20–21日
培训：6月22–23日

QCon 上海
全球软件开发大会
大会：10月17–19日
培训：10月20–21日

GMTC 深圳
全球大前端技术大会
大会：11月8–9日
培训：11月10–11日

AiCon 北京
全球人工智能与机器学习大会
大会：11月21–22日
培训：11月23–24日



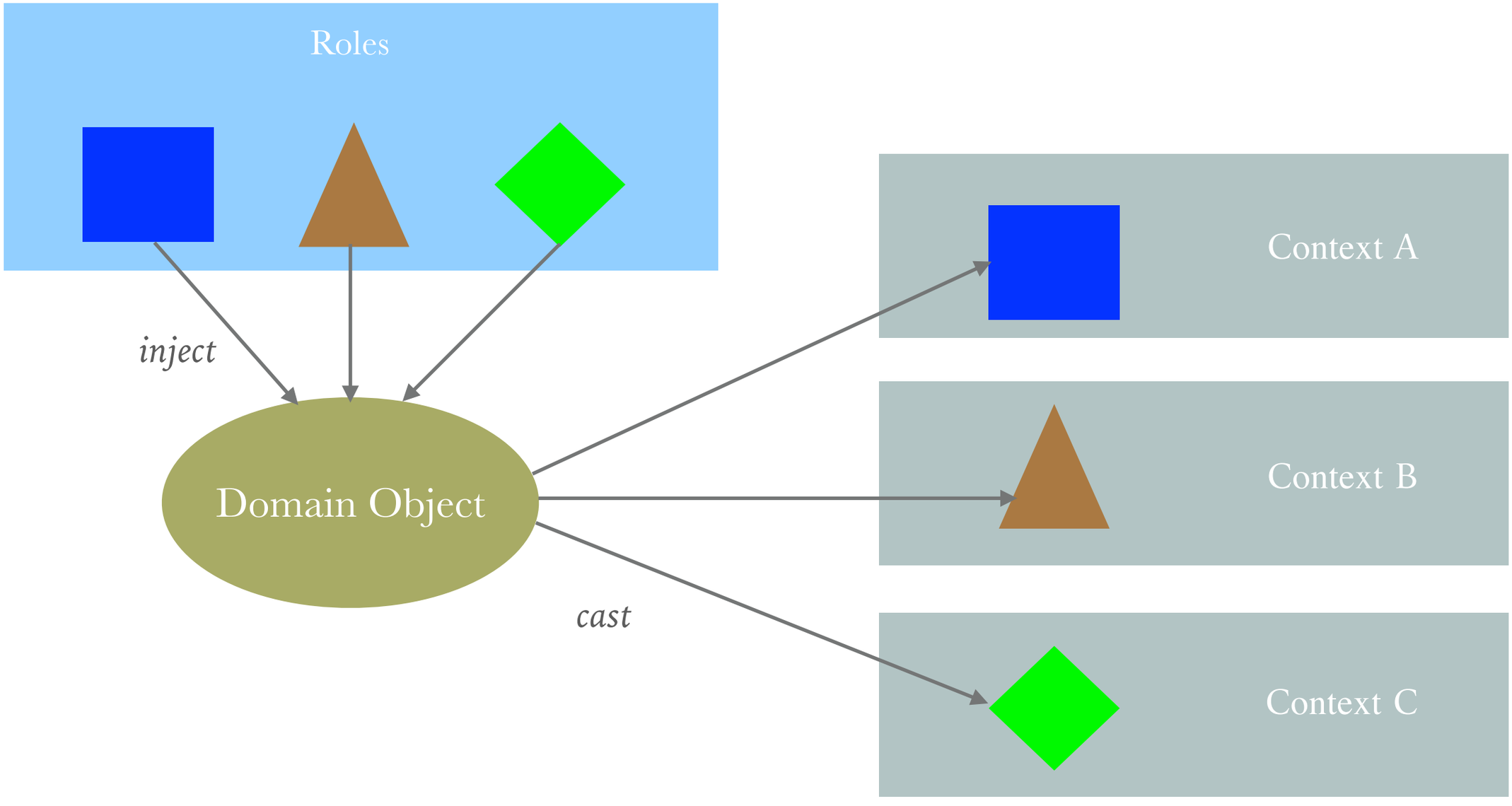
CONTENTS

DCI 架构模式

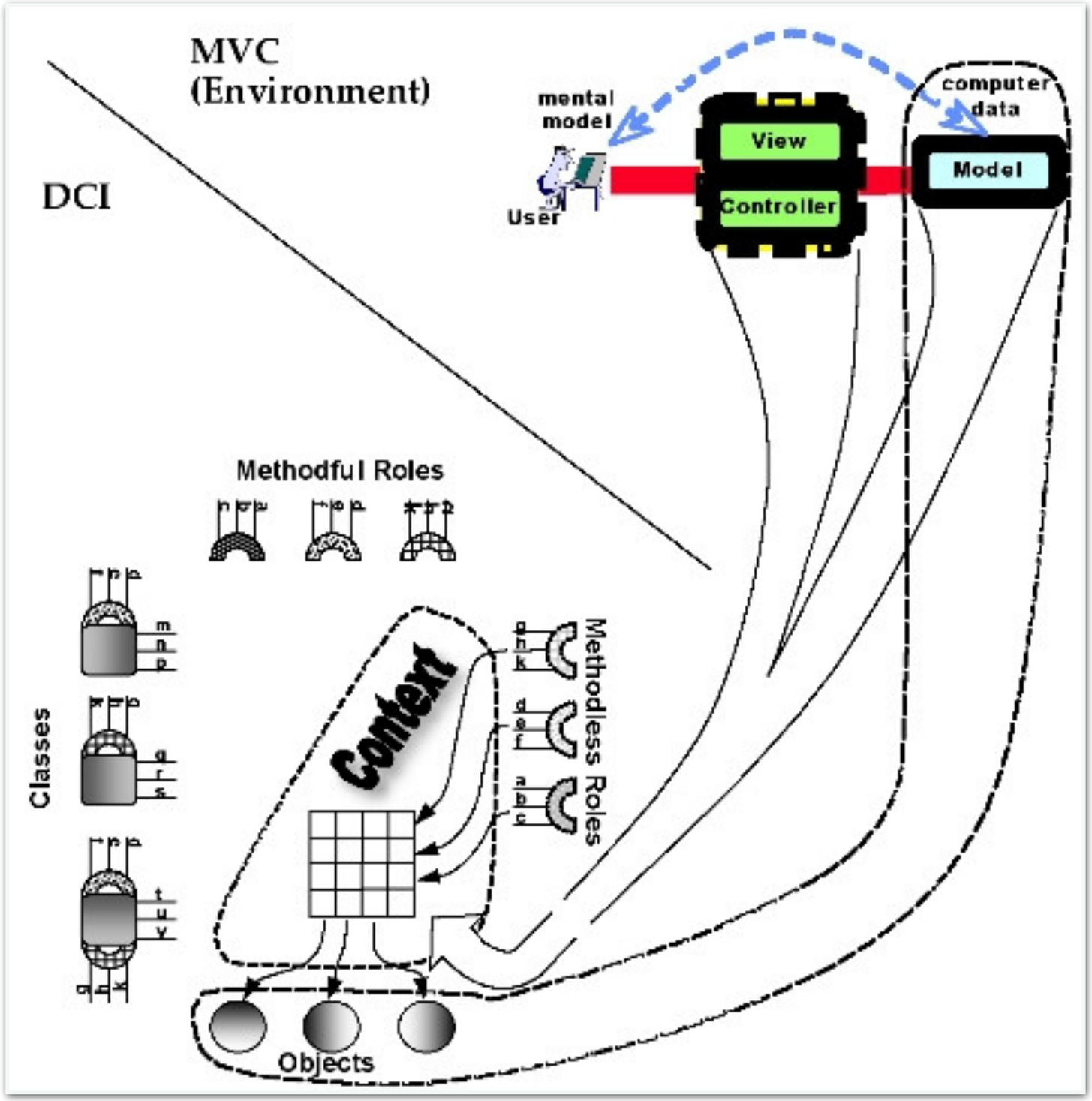
DCI （Data, Context, Interactive）

- Data: 描述领域对象，关注系统是什么
- Context: 理解业务的主线，知道什么对象应该扮演什么角色
- Interactive: 用角色 Role 来表达，体现系统做什么

DCI 视图


Roles
inject
Domain Object
cast
Context A
Context B
Context C

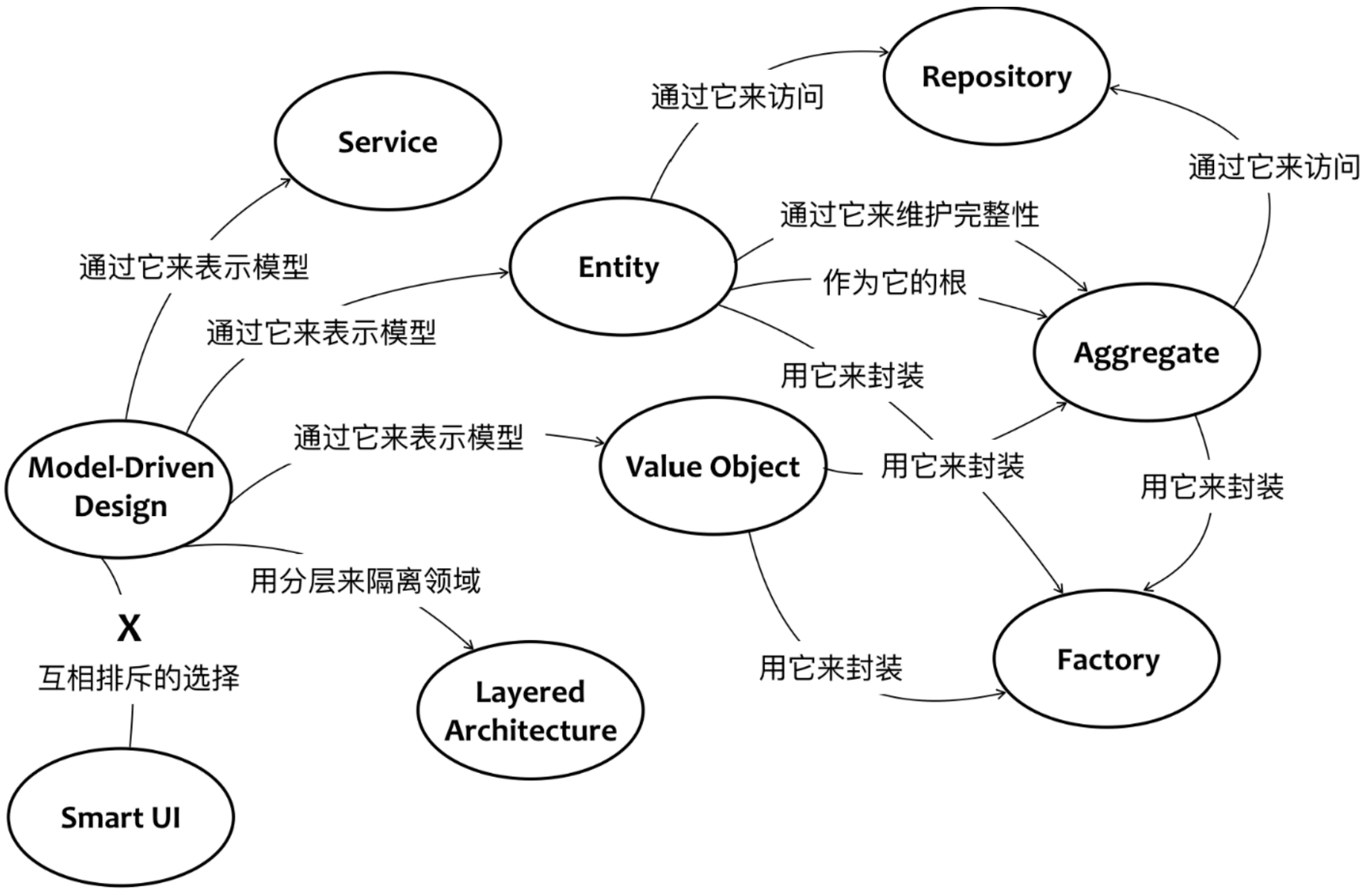
DCI 架构模式



Data, Context and Interaction : A New Architectural Approach by James O. Coplien and Trygve Reenskau

DCI 助力 DDD 战术设计

DDD 战术设计元素及其关系



DCI 与 DDD 的融合

- DCI:Data 概念对应 DDD:Aggregate
- DCI:Context 概念对应 DDD:Domain Service
- DCI:Role 的概念在 DDD 中没有
  - 对 Role 的建模解决了贫血模型和充血模型之争
  - 将 Role 的建模看作 DDD 战术建模的补充

Role 的依赖

- Role 中属性为系统行为相关数据
- Aggregate 中属性（实体和值对象）为领域模型相关数据
- Role 是行为类或接口，Aggregate 中的实体和值对象是数据类，Role 没有血肉，工作时需要依赖数据类

显式化关系

- 一个 Aggregate 可以支持哪些 Role
- 一个 Role 可以由哪些 Aggregate 扮演
- 一个场景下哪些 Aggregate 要扮演哪些 Role
- 当 Aggregate 内部实体行为比较多时可以嵌套使用 DCI 来拆分和组合

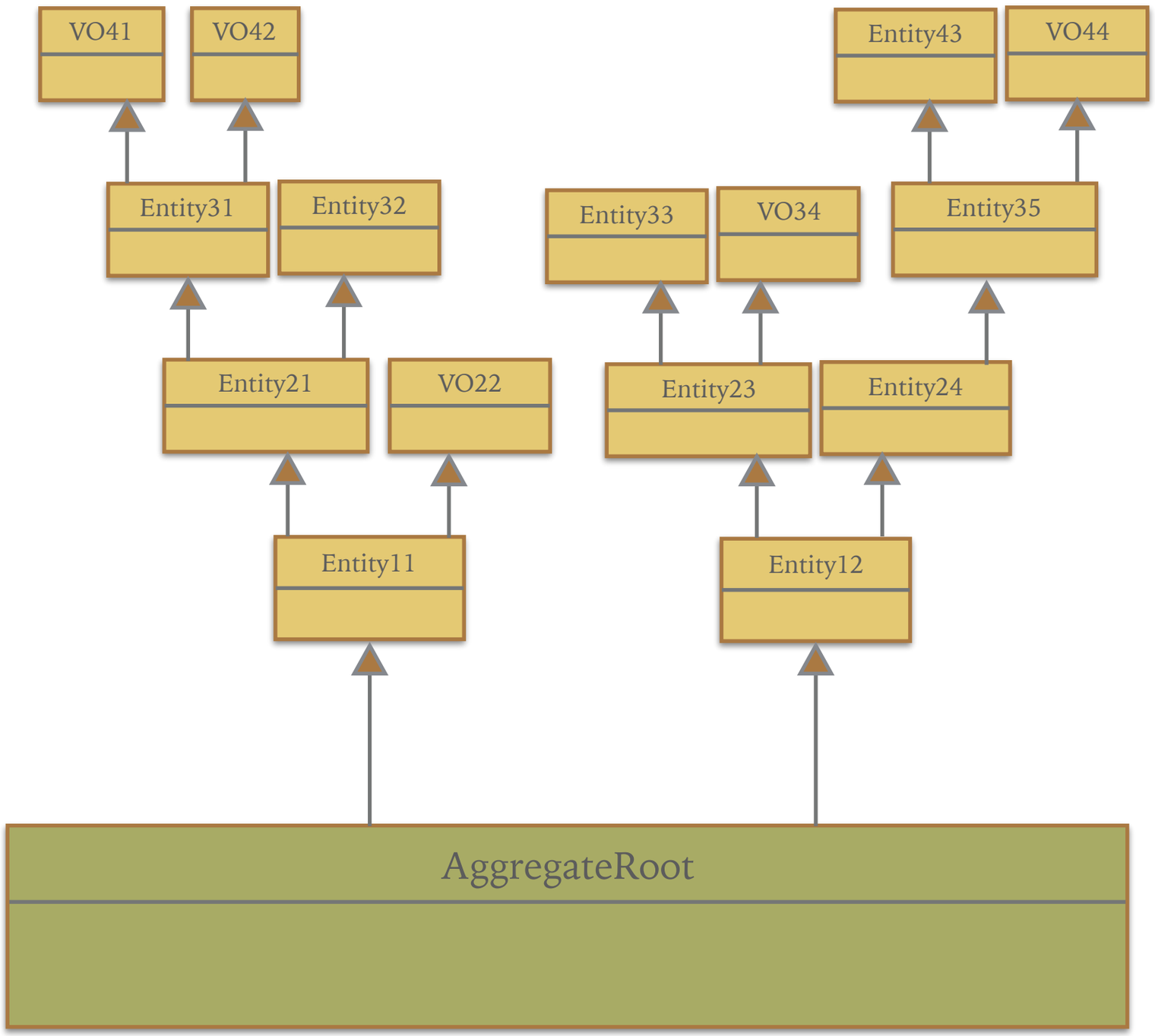
如何拆分 Role

- 使用场景
- 稳定性

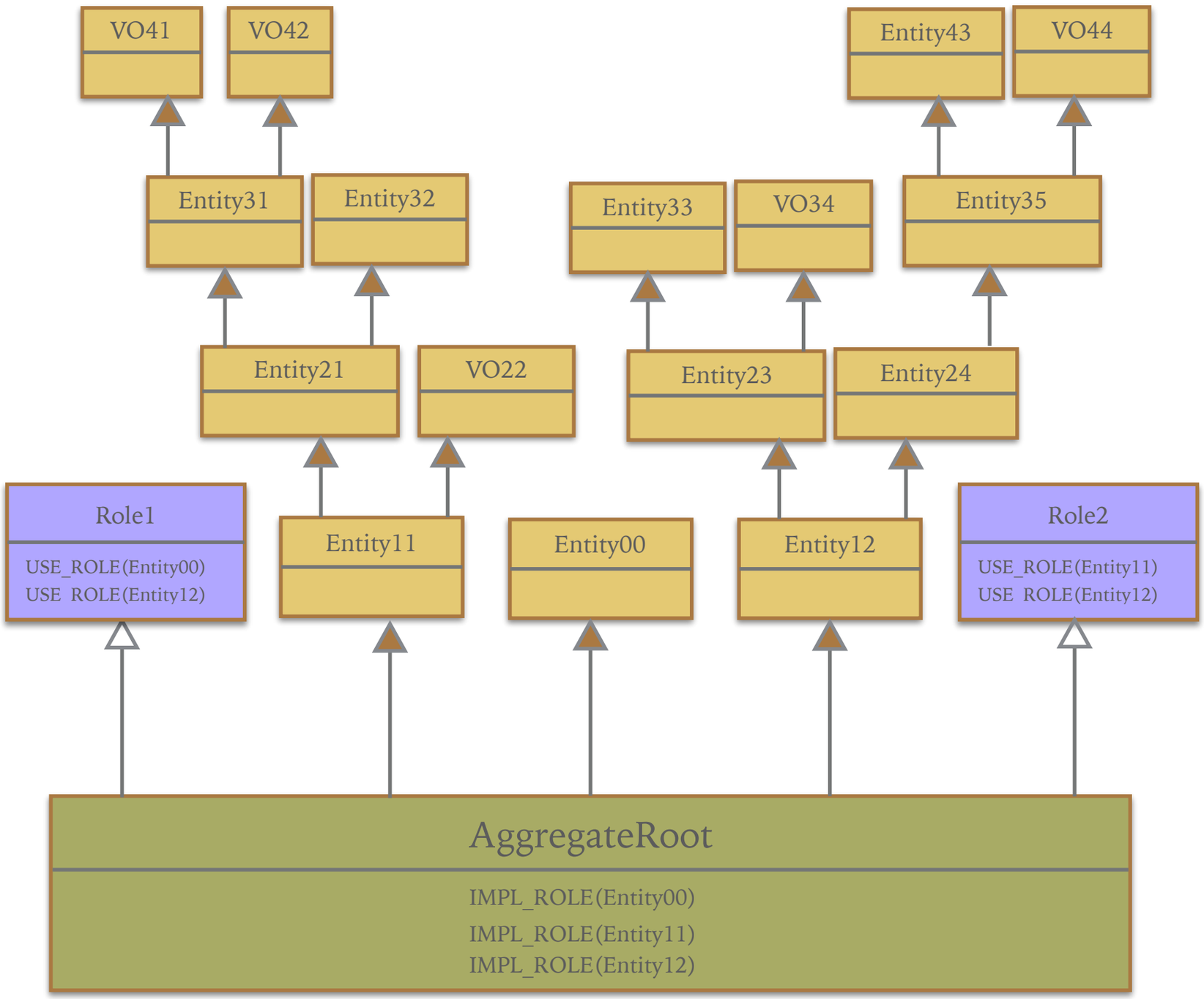
如何组合 Role

- 依赖注入是常用的组合手段
- 对于一些高级语言，有更好的组合手段：
  - C++ 中最有效的组合手段是多重继承
  - Scala 中最有效的组合手段是 Traits
  - Ruby 中最有效的组合手段是 Mixin

领域模型：无 Role


VO41
VO42
Entity43
VO44
Entity31
Entity32
Entity33
VO34
Entity35
Entity21
VO22
Entity23
Entity24
Entity11
Entity12
AggregateRoot

领域模型：有 Role


VO41
VO42
Entity43
VO44
Entity31
Entity32
Entity33
VO34
Entity35
Entity21
VO22
Entity23
Entity24
Role1
USE_ROLE(Entity00)
USE ROLE(Entity12)
Entity11
Entity00
Entity12
Role2
USE_ROLE(Entity11)
USE ROLE(Entity12)
AggregateRoot
IMPL_ROLE(Entity00)
IMPL_ROLE(Entity11)
IMPL_ROLE(Entity12)

DCI 助力 DDD 代码落地

类与对象

学院派：

大类大对象

数量很少

容易理解

工程派：

小类小对象

数量很多

容易修改

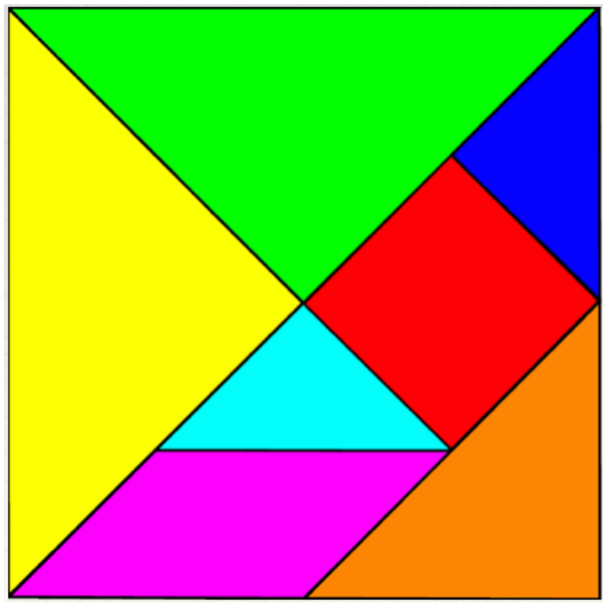
扩展使用 DCI

对于一个对象来说，不管是否有领域概念与之对应，只要行为很多，都可以应用 DCI 的思想进行拆分和组合：

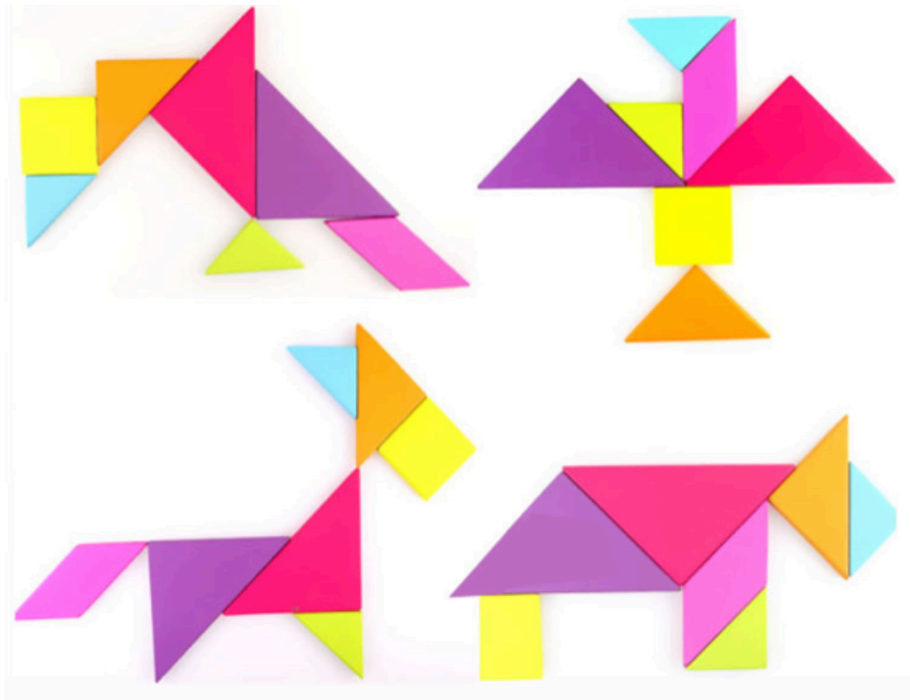
- 对象对应 DCI:Data
- 对象的 Client 对应 DCI:Context
- 对象在每个 Client 下的行为对应 DCI:Role
- 不管是数据类还是行为类，我们都按照 Role 的方式来交织
- 对象的工作仅仅是组合类，并注入依赖

正交设计四原则

七巧板



素材库



变化无穷的图案

设计对象的步骤

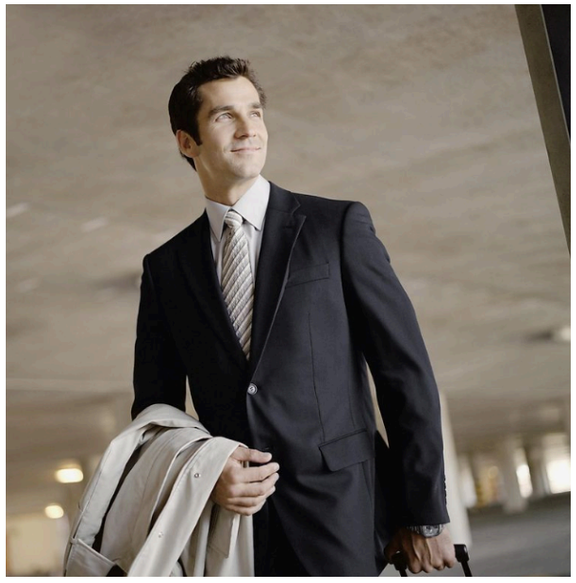
- 查看素材库，确认是否有所需的角色实现
- 如果存在，则通过组合手段来复用
- 如果不存在，则编写针对该角色的特定实现，除了自己使用外，也成了素材库的一部分

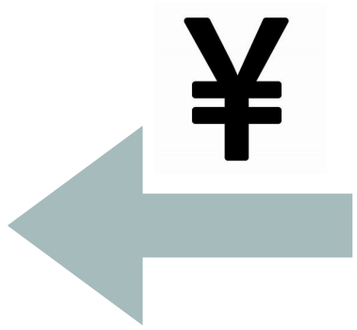
重新审视类与对象的关系

- 类作为一种模块化手段，遵循高内聚，低耦合，让软件易于应对变化
- 对象作为一种领域对象的的直接映射，解决了过多的类带来的可理解性问题，让领域可以指导设计，设计真正反映领域；领域对象需要真正意义上的生命周期管理

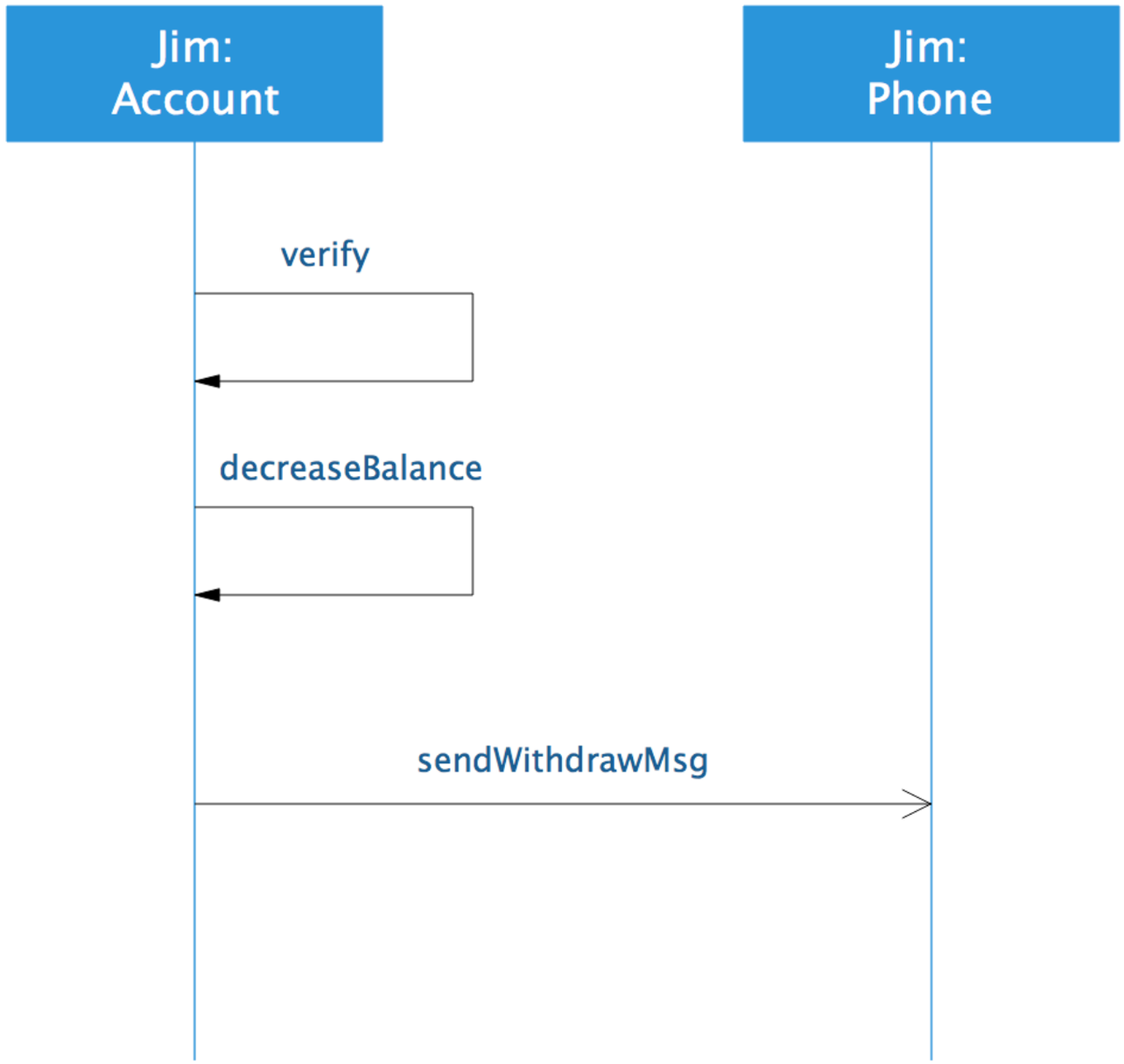
DDD/DCI 实践案例

需求1：取钱

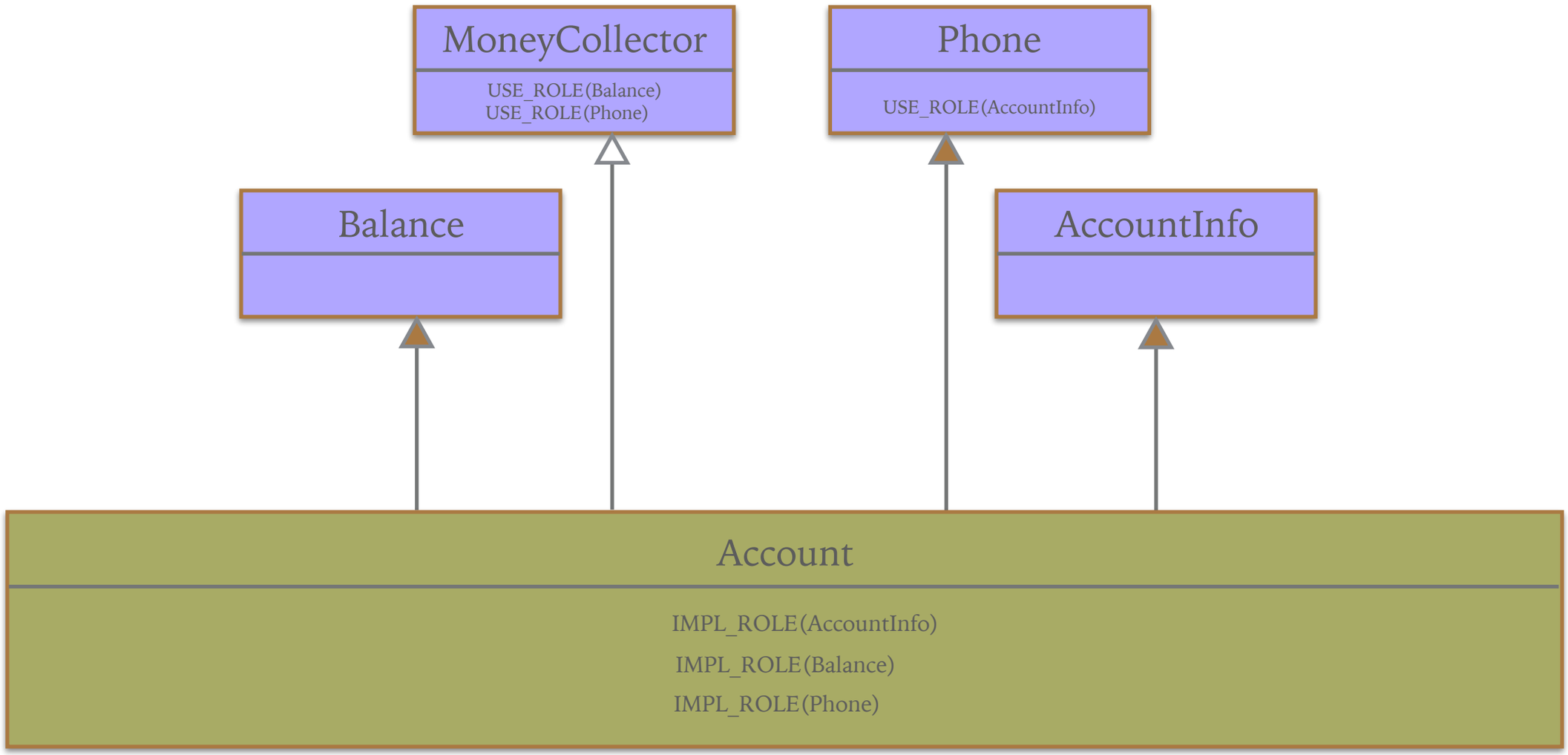




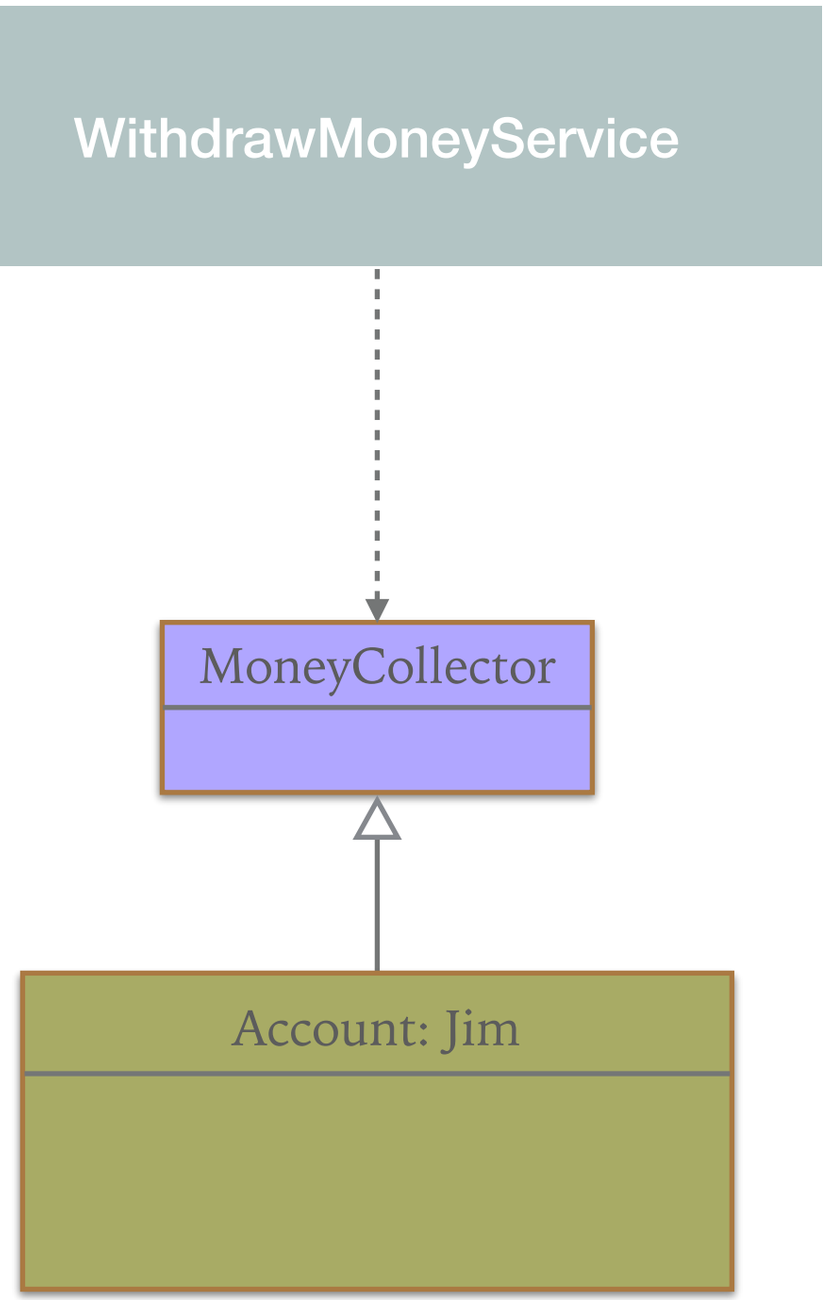
需求1：流程


Jim:
Account
Jim:
Phone
verify
decreaseBalance
sendWithdrawMsg

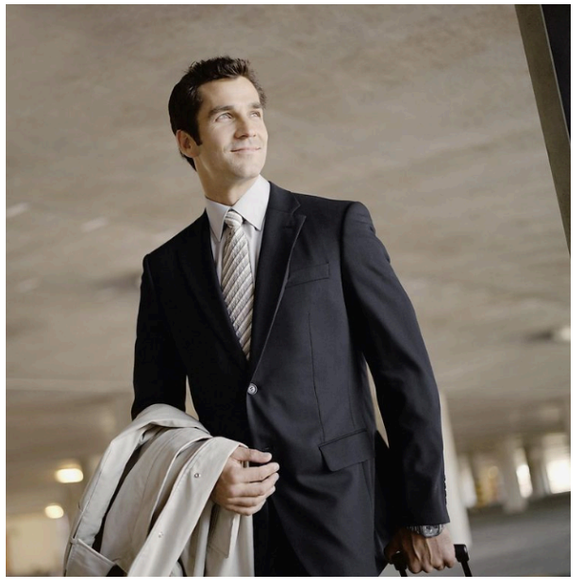
需求1：领域模型 Account

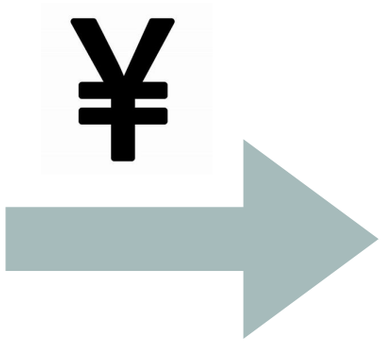


运行时视图

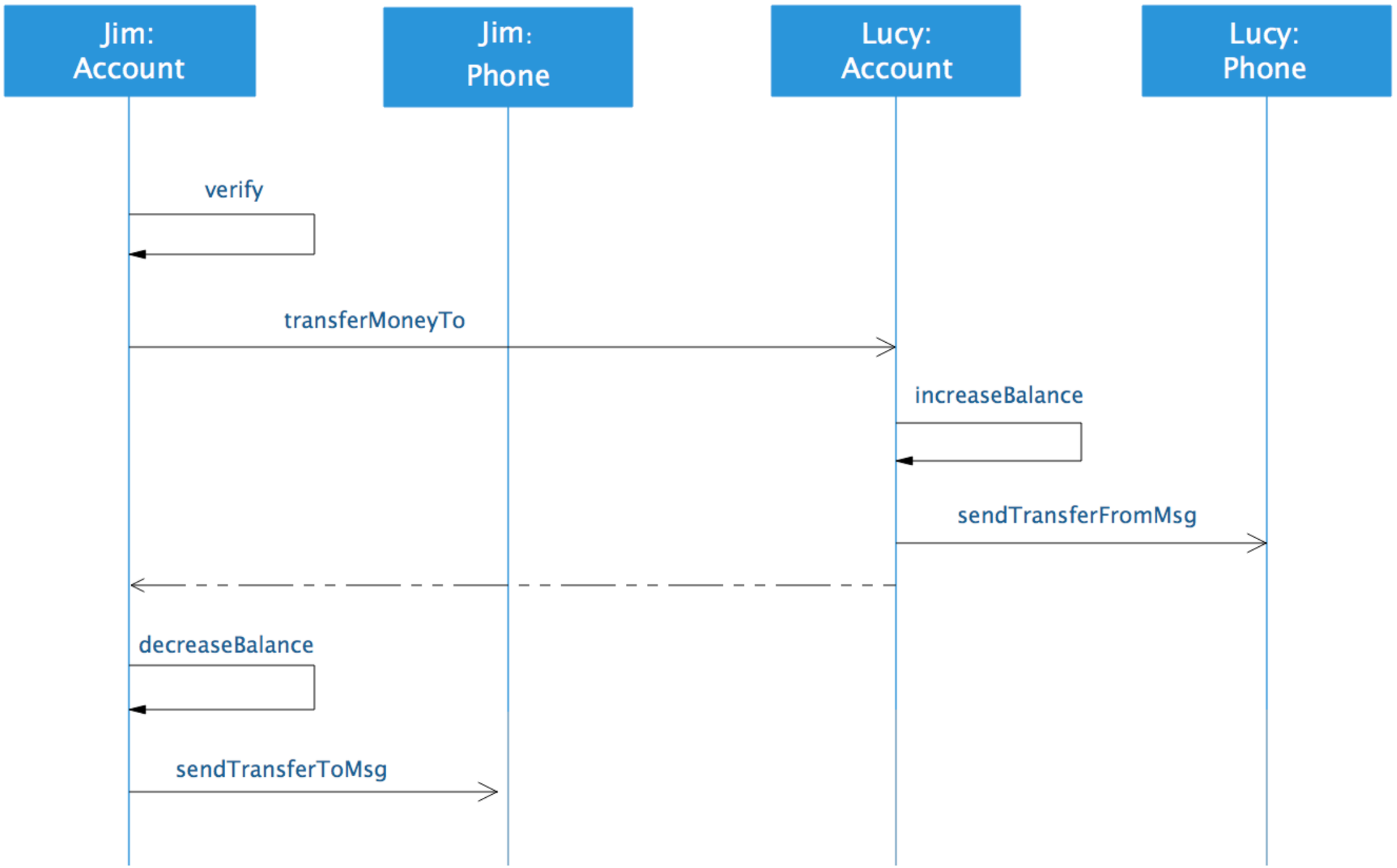


需求2：转账

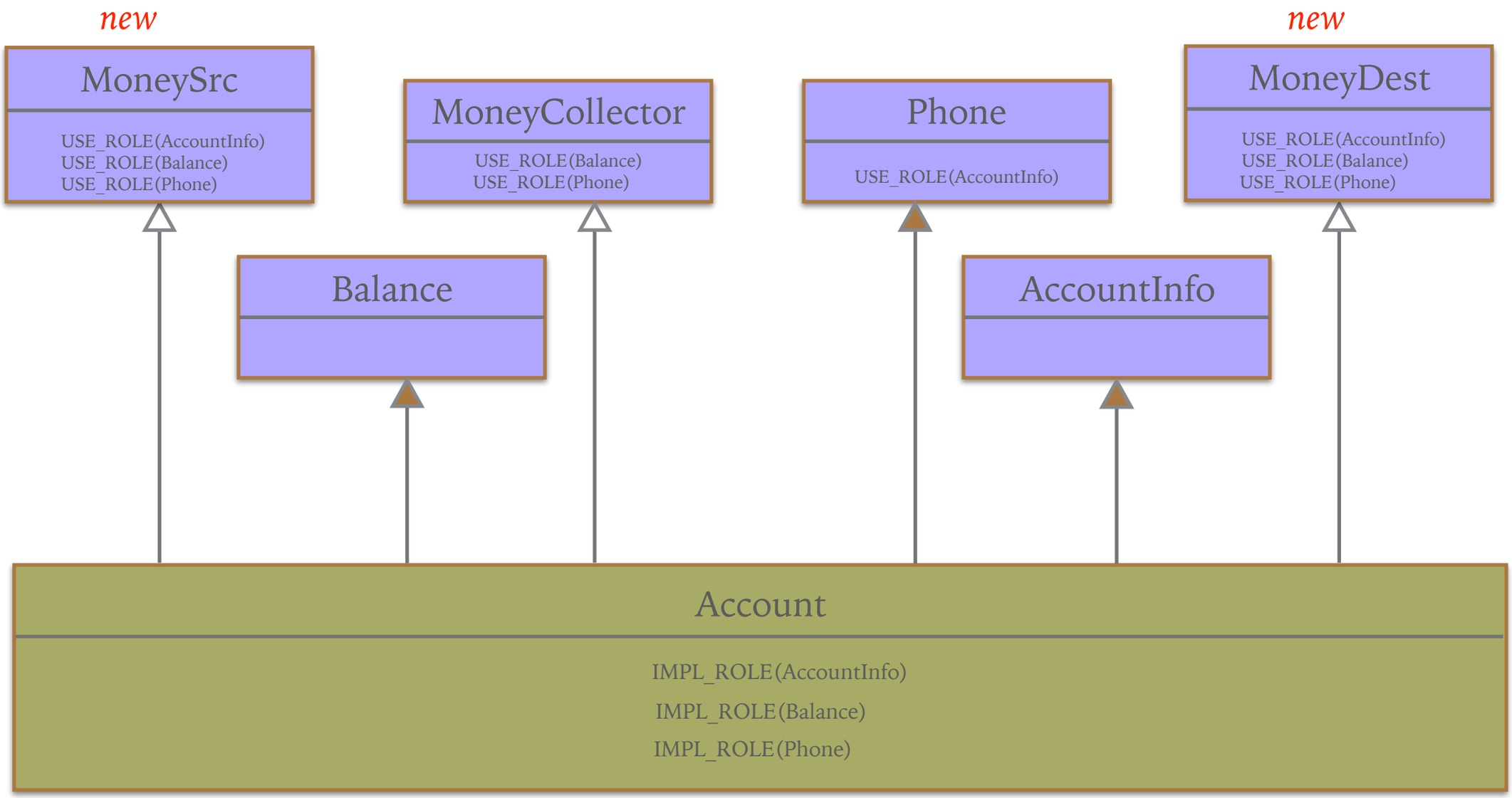




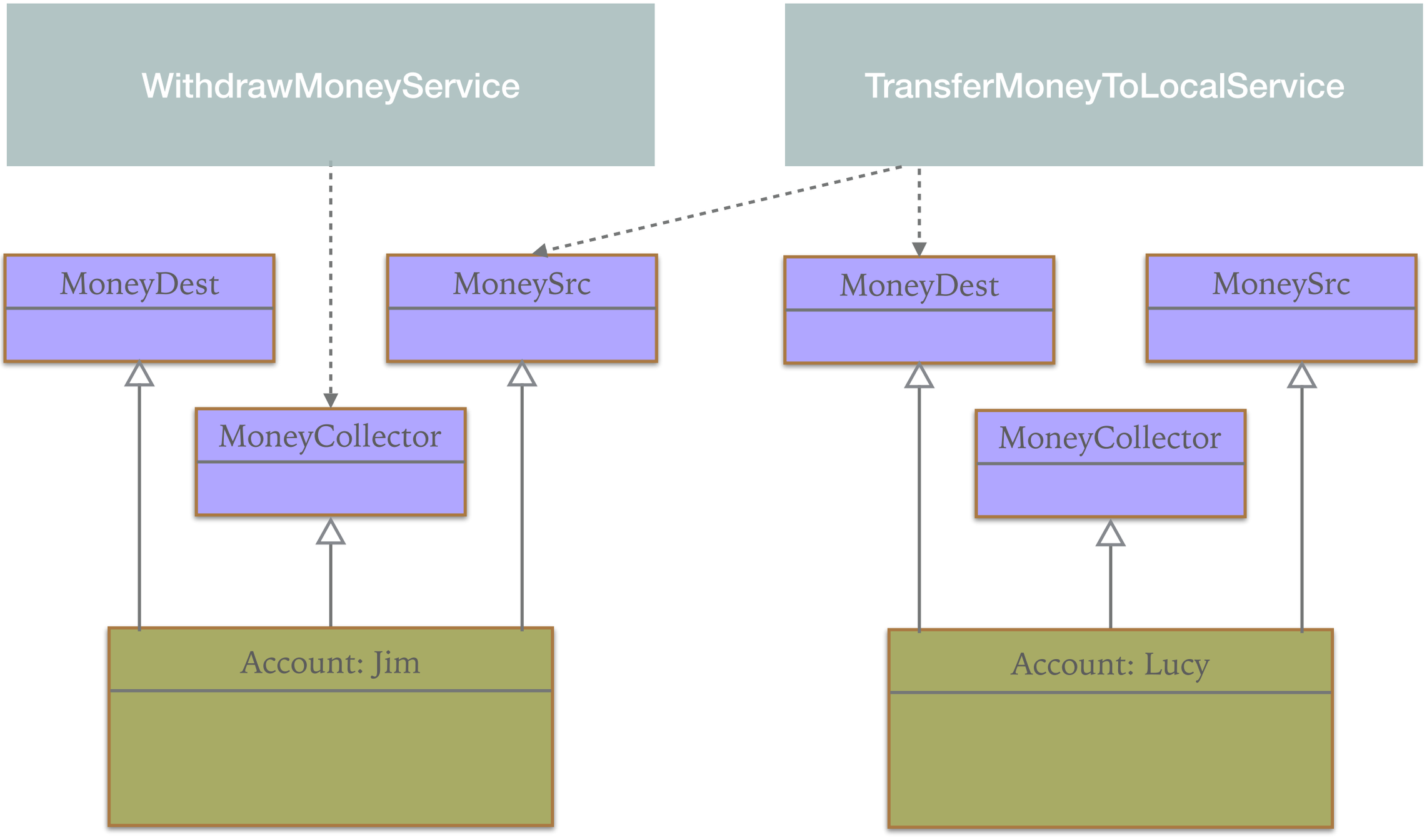
需求2：流程


Jim:
Account
Jim:
Phone
Lucy:
Account
Lucy:
Phone
verify
transferMoneyTo
increaseBalance
sendTransferFromMsg
decreaseBalance
sendTransferToMsg

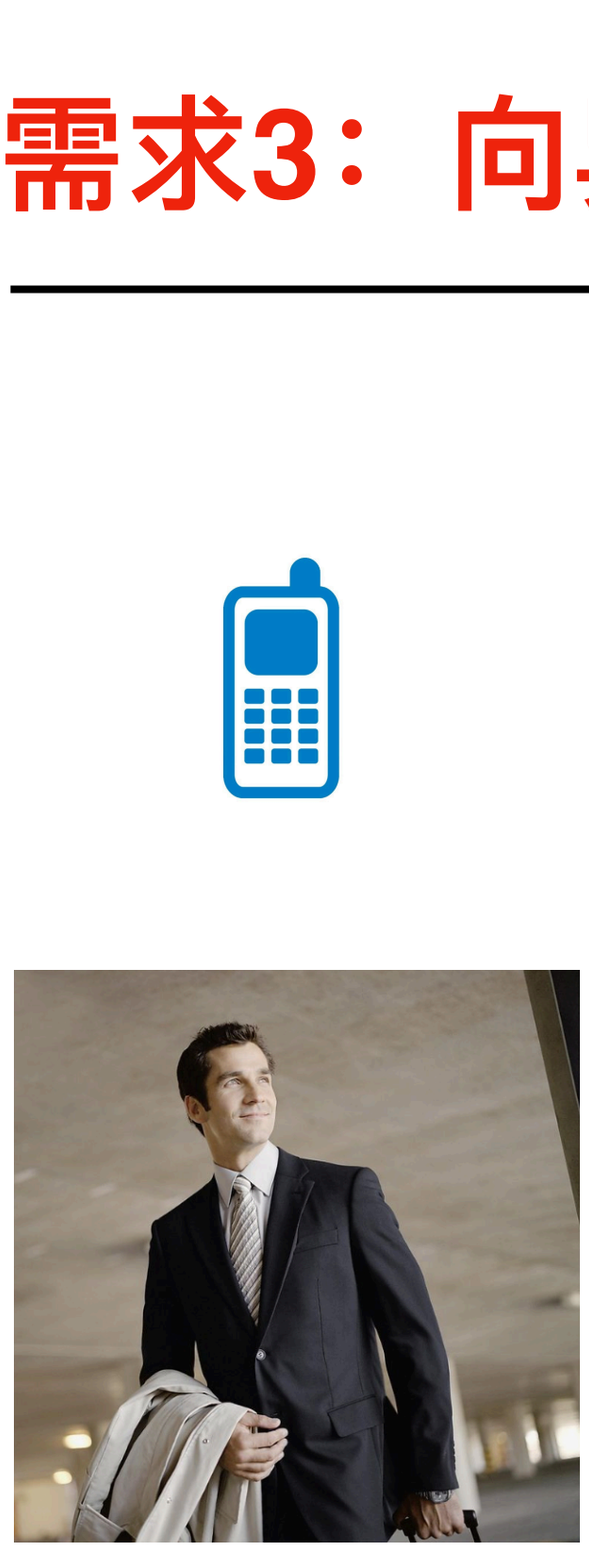
需求2：领域模型 Account

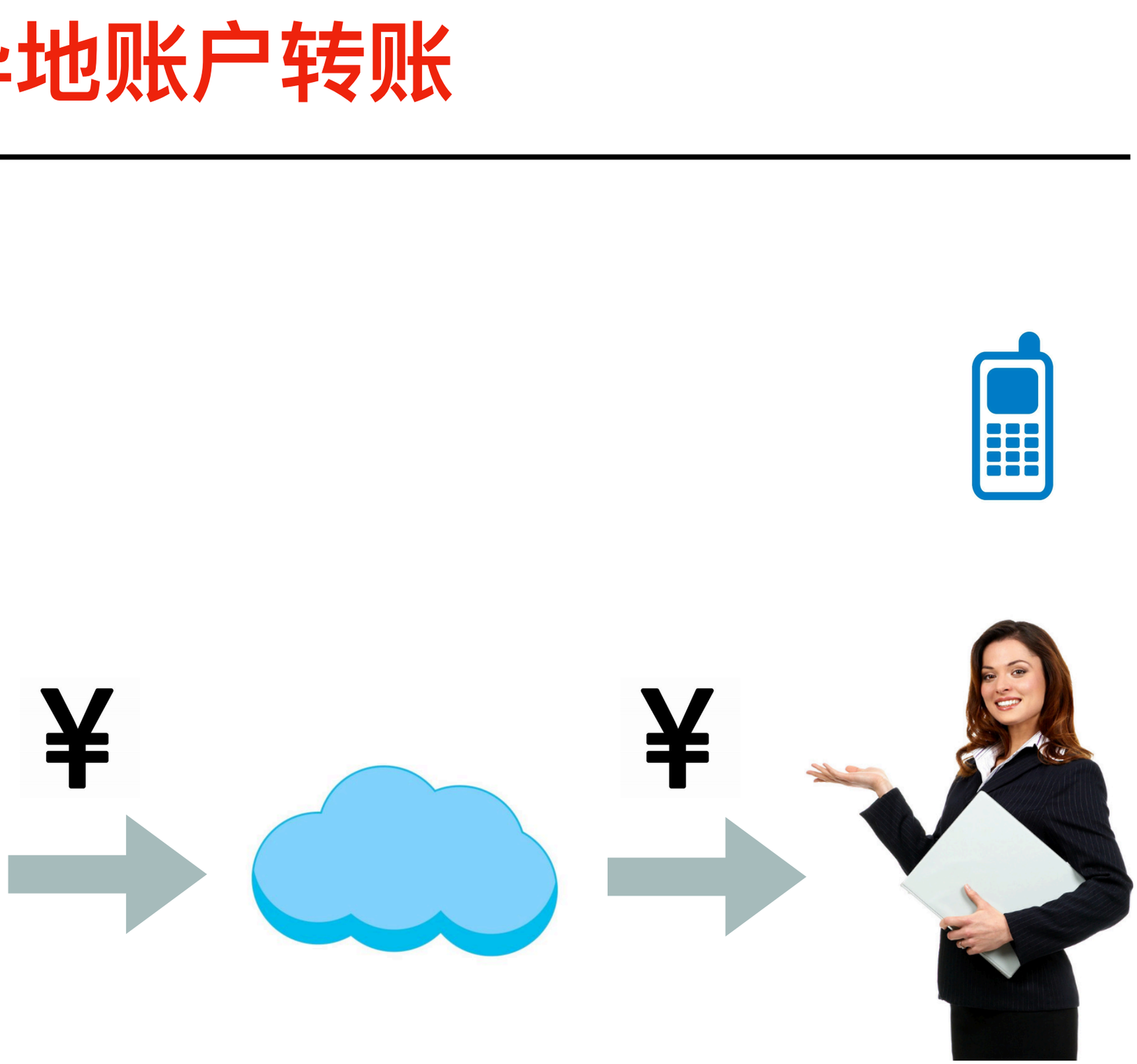


运行时视图

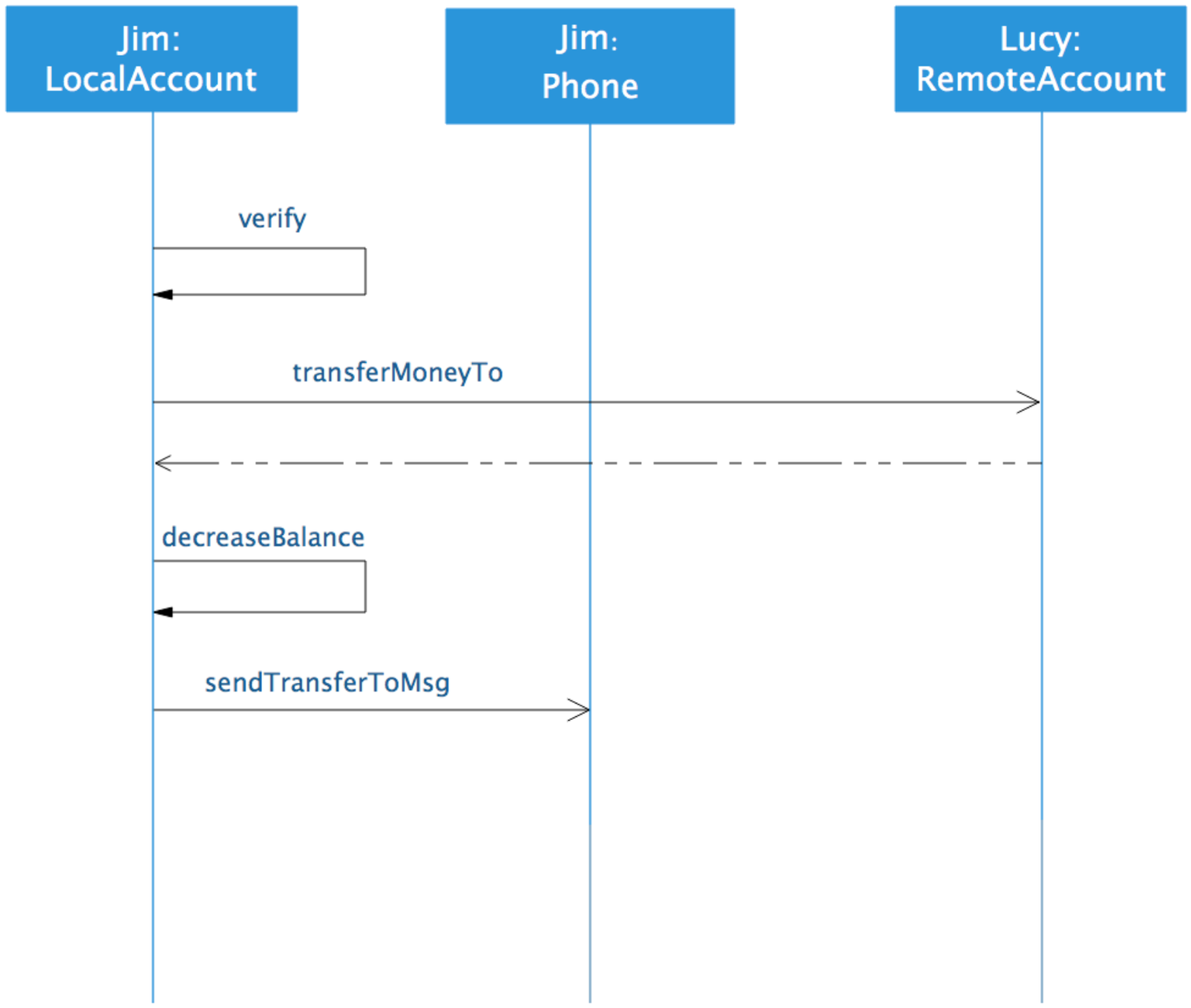

WithdrawMoneyService
TransferMoneyToLocalService
MoneyDest
MoneySrc
MoneyDest
MoneySrc
MoneyCollector
MoneyCollector
Account: Jim
Account: Lucy

需求3：向异地账户转账

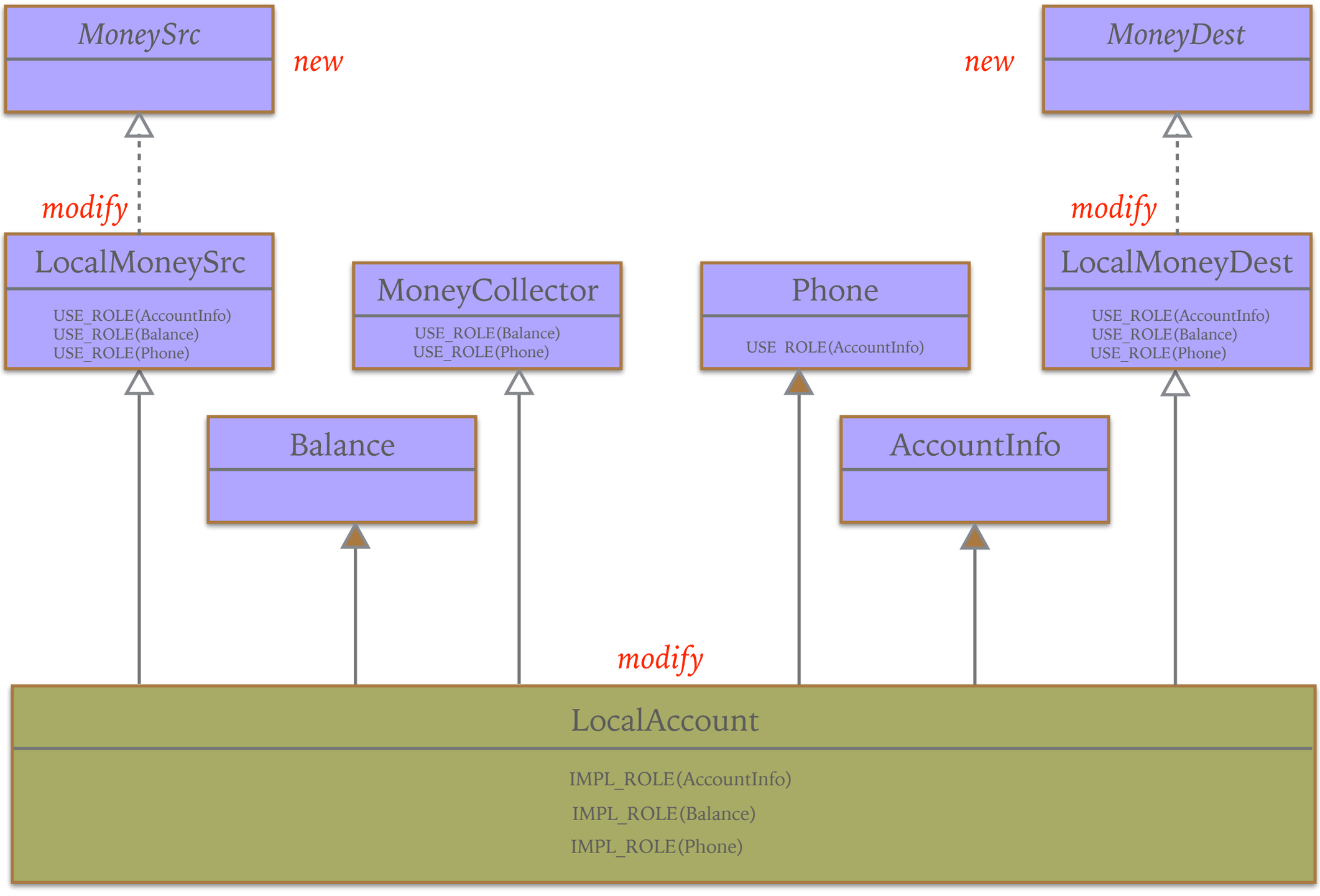




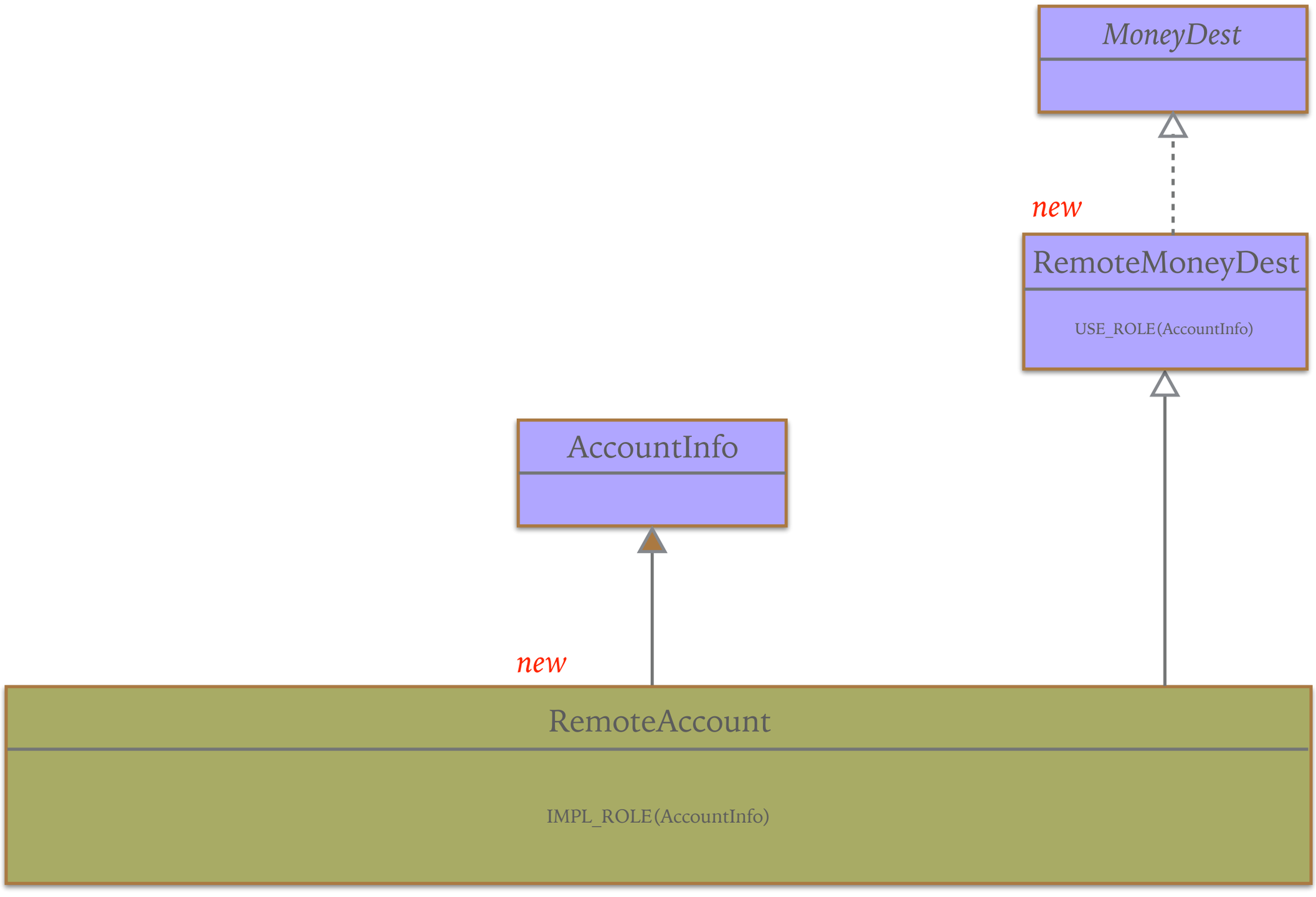
需求3：流程


Jim:
LocalAccount
Jim:
Phone
Lucy:
RemoteAccount
verify
transferMoneyTo
decreaseBalance
sendTransferToMsg

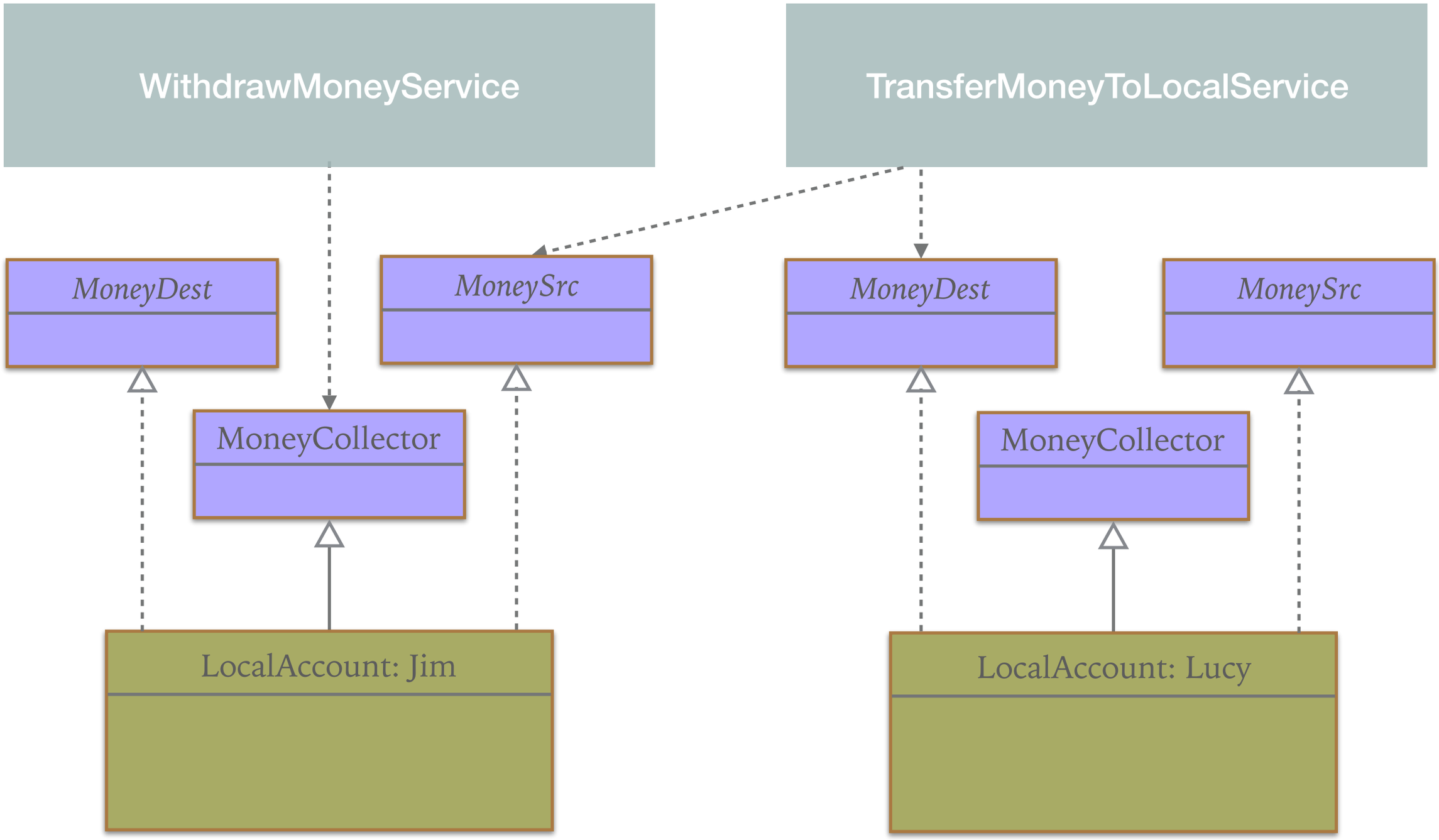
需求3：领域模型 LocalAccount



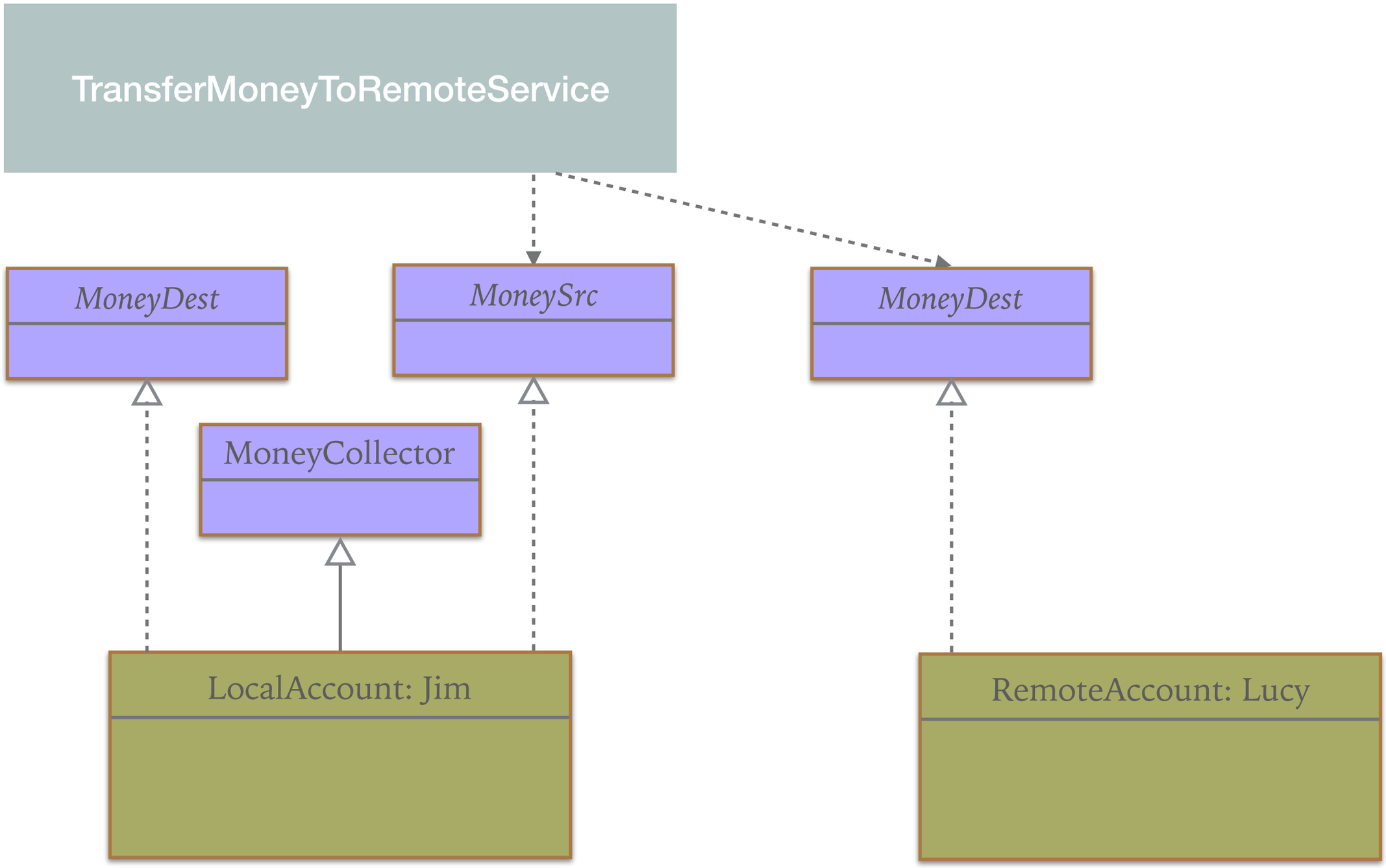
需求3：领域模型 RemoteAccount



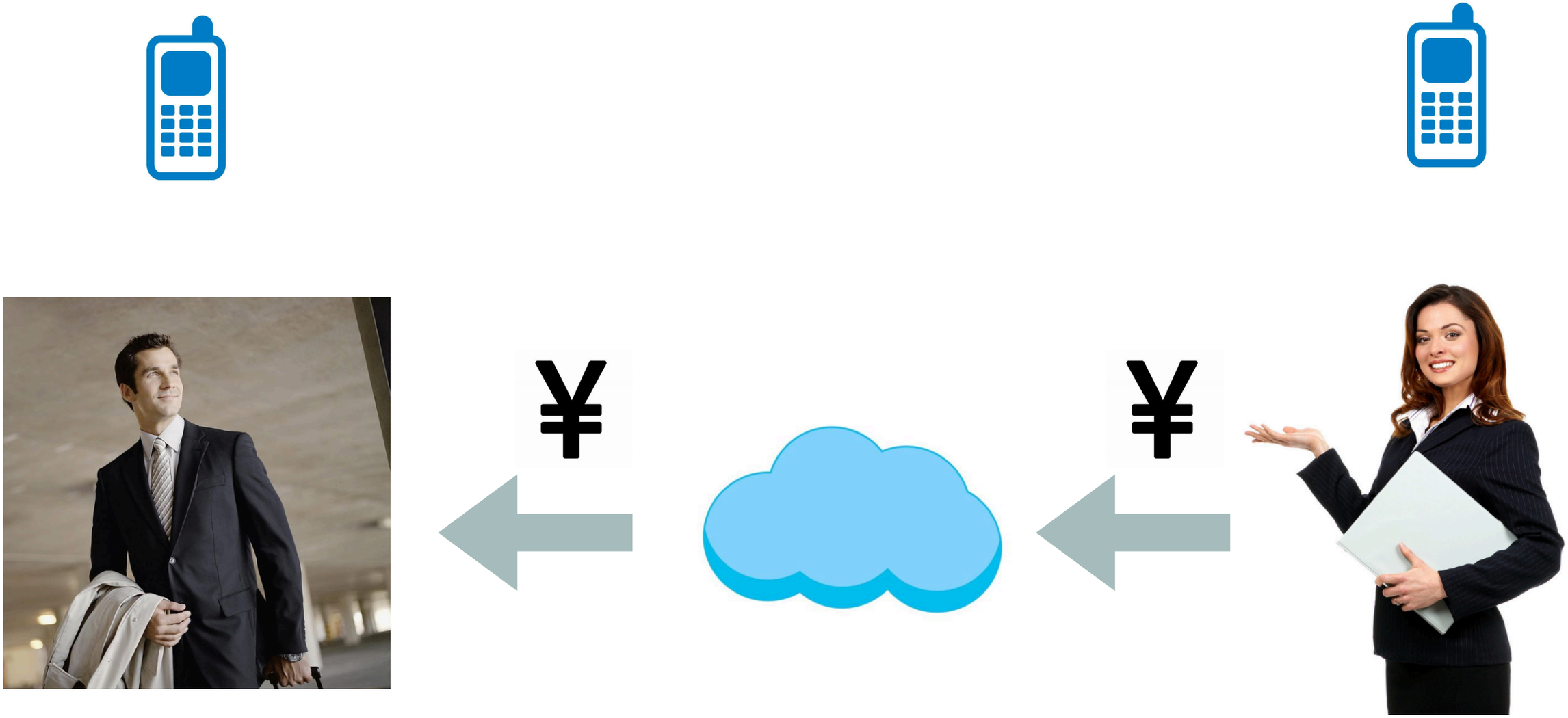
运行时视图


WithdrawMoneyService
TransferMoneyToLocalService
MoneyDest
MoneySrc
MoneyDest
MoneySrc
MoneyCollector
MoneyCollector
LocalAccount: Jim
LocalAccount: Lucy

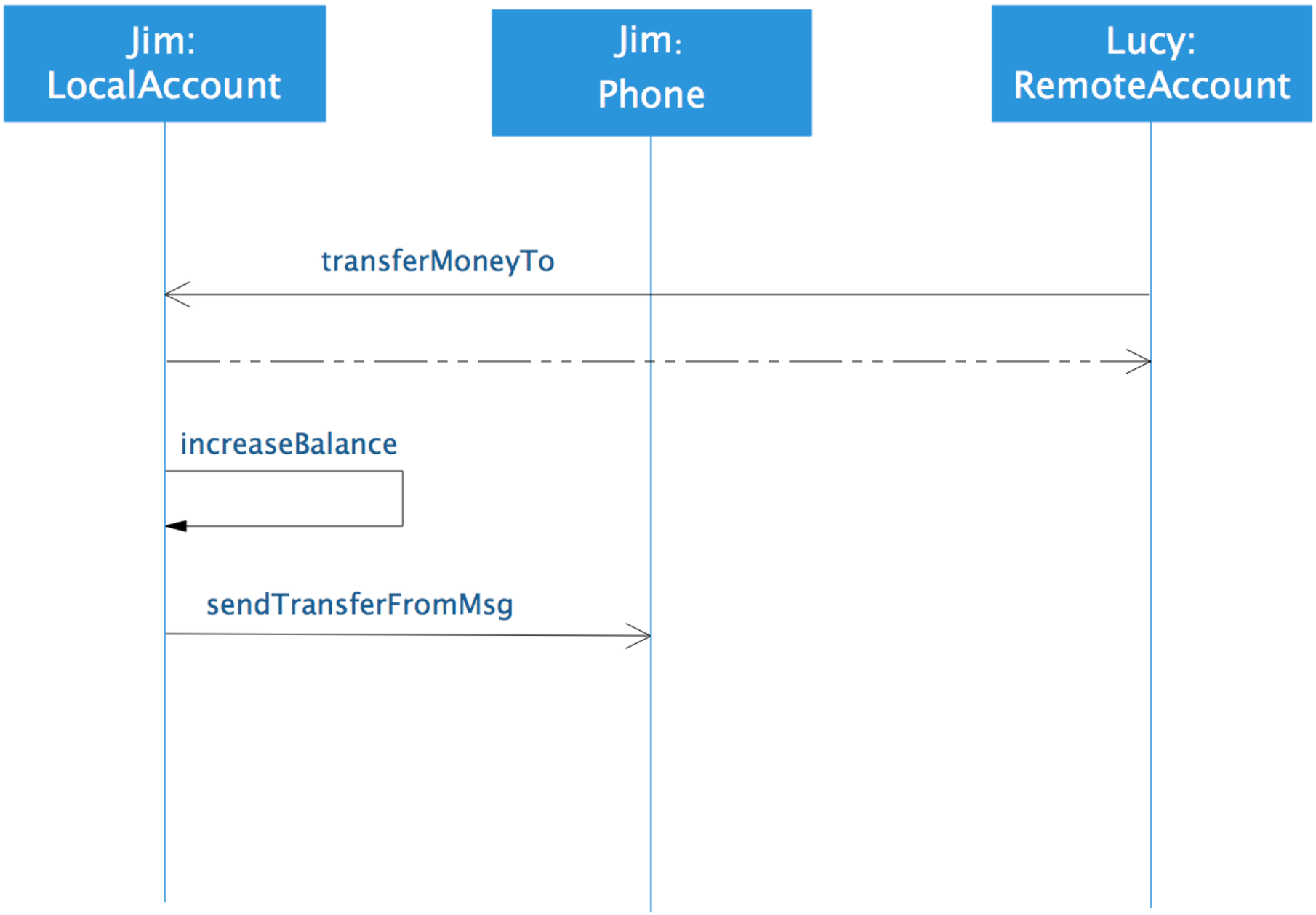
运行时视图


TransferMoneyToRemoteService
MoneyDest
MoneySrc
MoneyDest
MoneyCollector
LocalAccount: Jim
RemoteAccount: Lucy

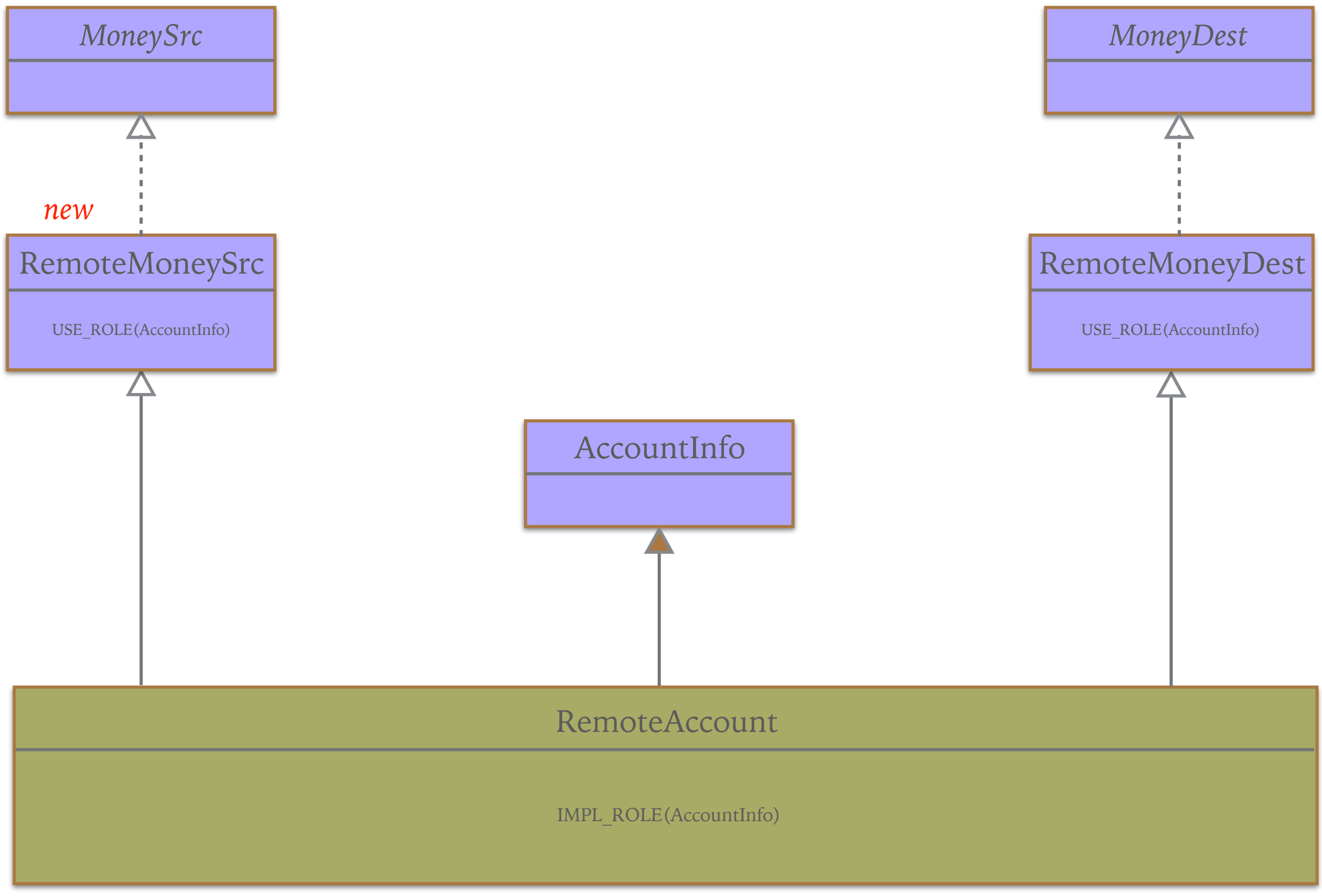
需求4：从异地账户转入



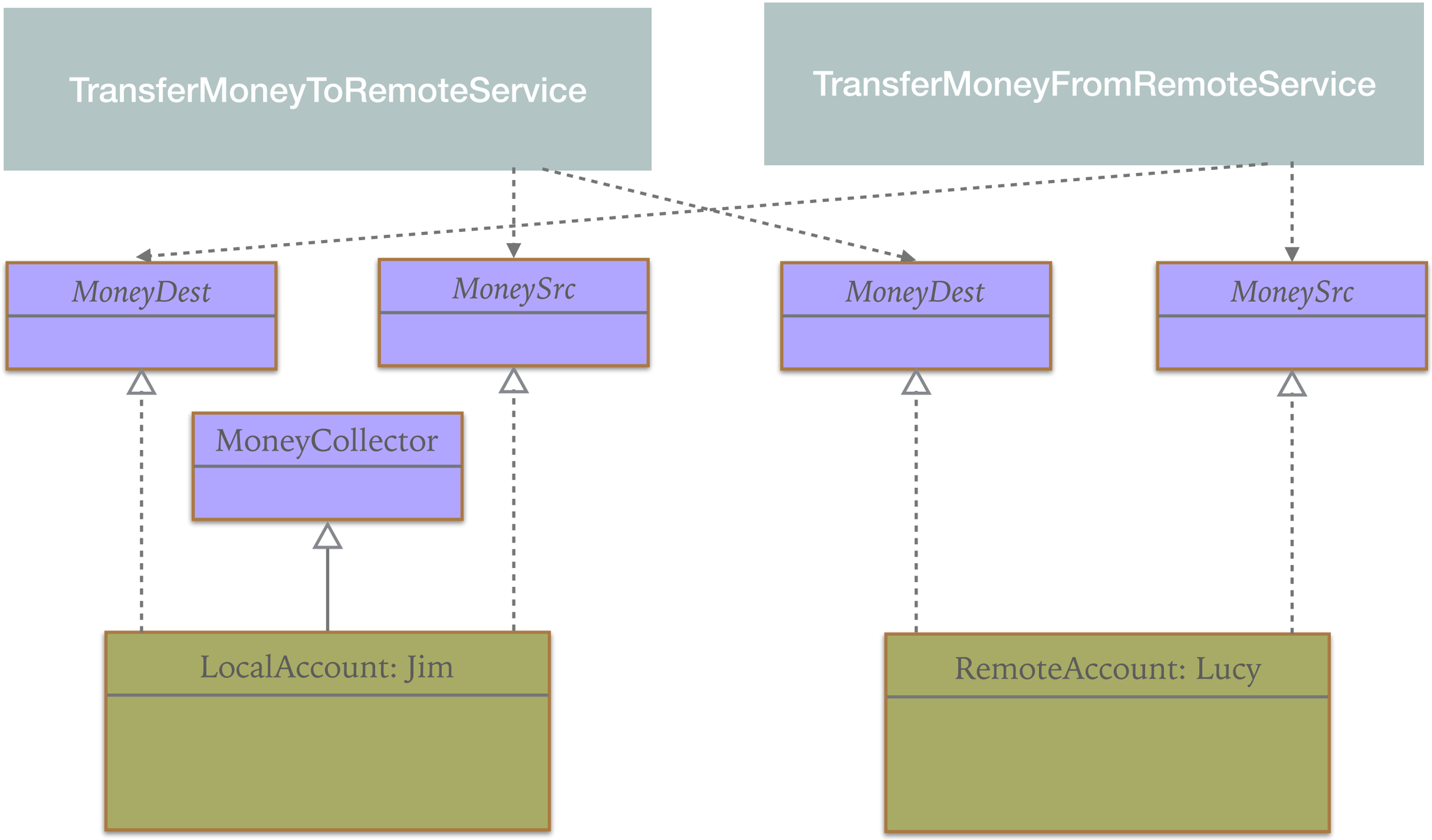
需求4：流程


Jim:
LocalAccount
Jim:
Phone
Lucy:
RemoteAccount
transferMoneyTo
increaseBalance
sendTransferFromMsg

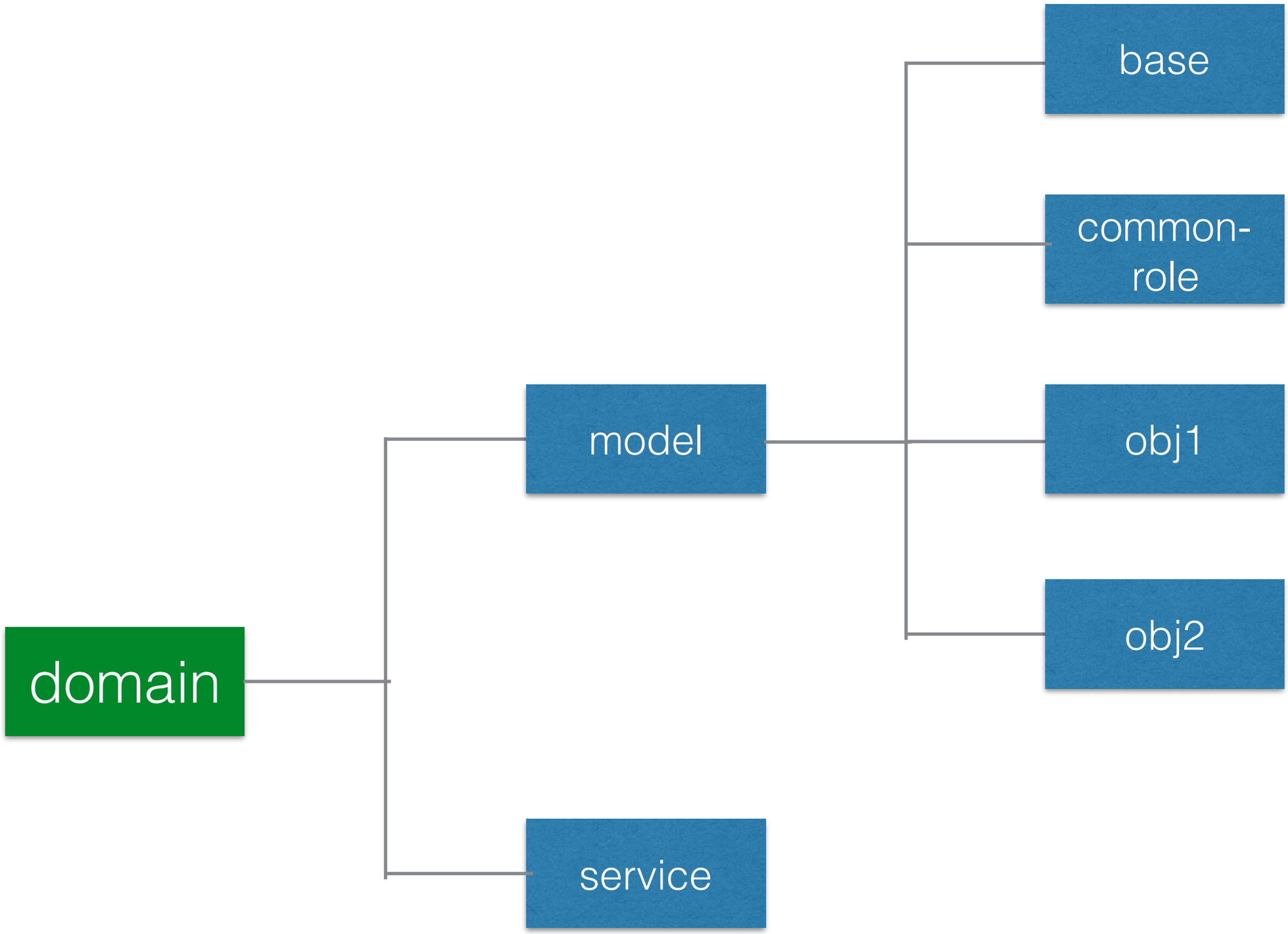
需求4：领域模型 RemoteAccount



运行时视图


TransferMoneyToRemoteService
TransferMoneyFromRemoteService
MoneyDest
MoneySrc
MoneyDest
MoneySrc
MoneyCollector
LocalAccount: Jim
RemoteAccount: Lucy

物理设计



代码示例：Aggregate 定义

```
struct LocalAccount : AggregateRoot
                    , MoneyCollector
                    , LocalMoneySrc
                    , LocalMoneyDest
                    , private AccountInfo
                    , private Balance
                    , private Phone
{
    LocalAccount(const std::string& accountId,
            const std::string& phoneNumber,
            U32 initialAmount);

private:
    IMPL_ROLE(AccountInfo);
    IMPL_ROLE(Balance);
    IMPL_ROLE(Phone);
};
```

```
struct RemoteAccount : AggregateRoot
                     , RemoteMoneySrc
                     , RemoteMoneyDest
                     , private AccountInfo
{
    RemoteAccount(const std::string& accountId);

private:
    IMPL_ROLE(AccountInfo);
};
```

代码示例：Role 定义

```
struct MoneyDest : Role
{
    ABSTRACT(void transferMoneyFrom(const std::string& fromId, U32 amount));
    ABSTRACT(std::string getAccountId() const);
};
```

```
struct MoneySrc : Role
{
    ABSTRACT(void transferMoneyTo(MoneyDest& dest, U32 amount));
};
```

```
struct LocalMoneyDest : MoneyDest
{
    OVERRIDE(void transferMoneyFrom(const std::string& fromId, U32 amount));
    OVERRIDE(std::string getAccountId() const);

private:
    USE_ROLE(AccountInfo);
    USE_ROLE(Balance);
    USE_ROLE(Phone);
};
```

```
struct RemoteMoneySrc : MoneySrc
{
    OVERRIDE(void transferMoneyTo(MoneyDest& dst, U32 amount));

private:
    USE_ROLE(AccountInfo);
};
```

```
struct MoneyCollector : Role
{
    void withdraw(U32 amount);

private:
    USE_ROLE(Balance);
    USE_ROLE(Phone);
};
```

代码示例：Role 实现

```
void LocalMoneySrc::transferMoneyTo(MoneyDest& dest, U32 amount)
{
    if (ROLE(Balance).get() < amount)
    {
        throw "insufficient money!";
        return;
    }
    dest.transferMoneyFrom(ROLE(AccountInfo).getAccountId(), amount);
    ROLE(Balance).decrease(amount);
    ROLE(Phone).sendTransferToMsg(dest.getAccountId(), amount);
}

U32 LocalMoneySrc::getAmount() const
{
    return ROLE(Balance).get();
}
```

```
void LocalMoneyDest::transferMoneyFrom(const std::string& fromId, U32 amount)
{
    ROLE(Balance).increase(amount);
    ROLE(Phone).sendTransferFromMsg(fromId, amount);
}

std::string LocalMoneyDest::getAccountId() const
{
    return ROLE(AccountInfo).getAccountId();
}
```

```
void RemoteMoneySrc::transferMoneyTo(MoneyDest& dest, U32 amount)
{
    dest.transferMoneyFrom(ROLE(AccountInfo).getAccountId(), amount);
    Response ok{false};
    sendProtocolResponse(ok);
}
```

代码示例：Domain Service

```
void WithdrawMoneyService::exec(const std::string& accountId, U32 amount)
{
    auto account = repo->get(accountId);
    SELF(*account, MoneyCollector).withdraw(amount);
}
```

```
void TransferMoneyToLocalService::exec(const std::string& fromId, const std::string& toId, U32 amount)
{
    auto fromAccount = repo->get(fromId);
    auto toAccount = repo->get(toId);
    SELF(*fromAccount, MoneySrc).transferMoneyTo(SELF(*toAccount, MoneyDest), amount);
}
```

```
void TransferMoneyToRemoteService::exec(const std::string& fromId, const std::string& toId, U32 amount)
{
    auto fromAccount = repo->get(fromId);
    auto toAccount = new RemoteAccount(toId);
    SELF(*fromAccount, MoneySrc).transferMoneyTo(SELF(*toAccount, MoneyDest), amount);
    delete toAccount;
}
```

```
void TransferMoneyFromRemoteService::exec(const std::string& fromId, const std::string& toId, U32 amount)
{
    auto fromAccount = new RemoteAccount(toId);
    auto toAccount = repo->get(toId);
    SELF(*fromAccount, MoneySrc).transferMoneyTo(SELF(*toAccount, MoneyDest), amount);
    delete fromAccount;
}
```

小结

小结

- DDD 和 DCI 的融合
- Role 的拆分原则和组合手段
- 小类大对象

案例源码地址

C++版：https://github.com/agiledragon/transfer-money
Go版：https://github.com/agiledragon/transfer-money-go
Python版：https://github.com/agiledragon/transfer-money-python

当 DDD 遇上 DCI，使得软件容易应对变化，同时解决了过多的类带来的可理解性问题，让领域可以指导设计，设计真正反映领域，而这才是领域驱动设计的真正目的和精髓

- 张晓龙 @ 中兴通讯
- 架构师，技术教练
- zhangxiaolong1980@126.com
- 简书个人主页：
  http://www.jianshu.com/u/1381dc29fed9
- GitHub个人主页：
  https://github.com/agiledragon
- GitChat认证作者，昵称：agiledragon

THANK YOU

ArchSummit
全 球 架 构 师 峰 会