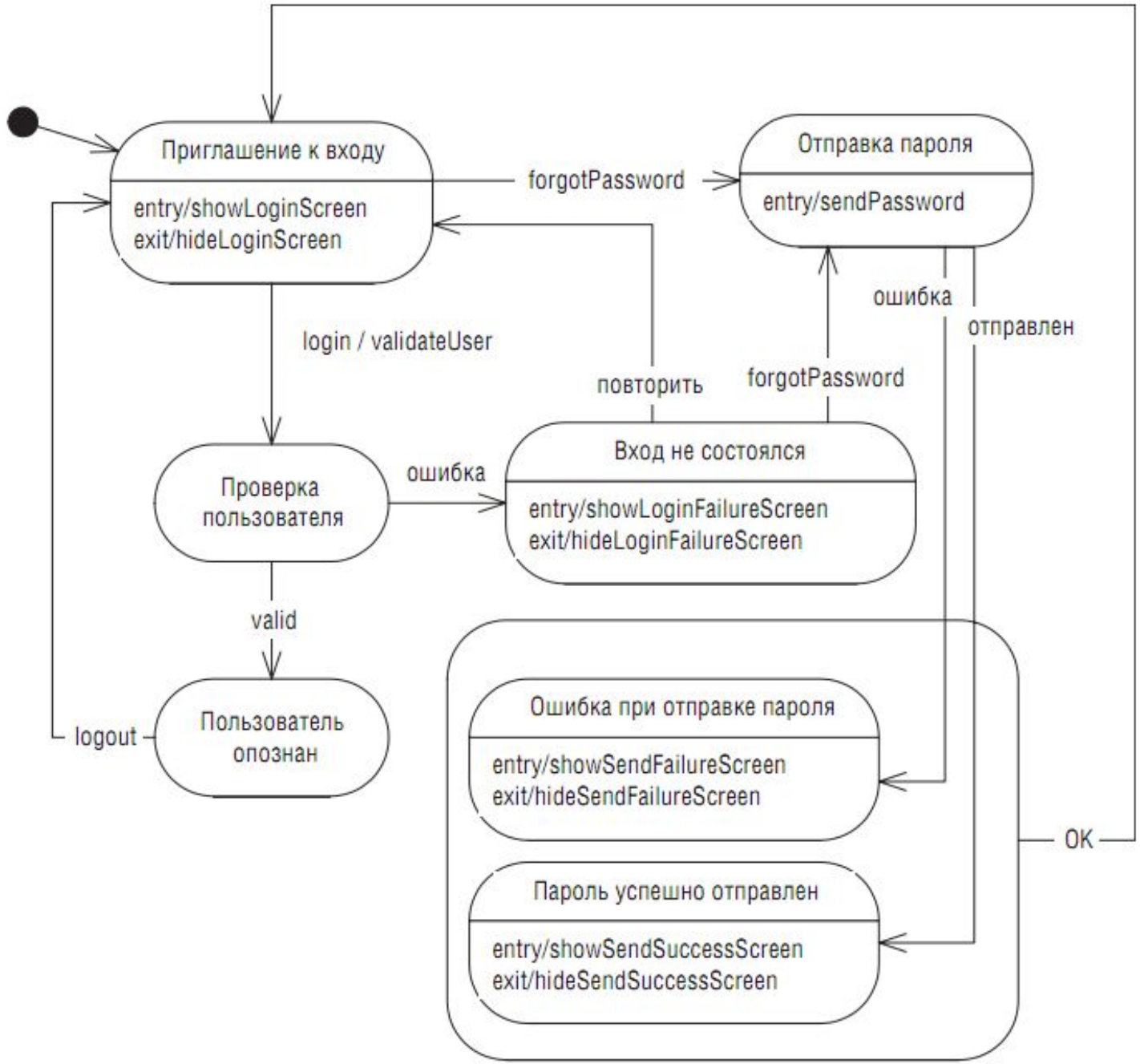
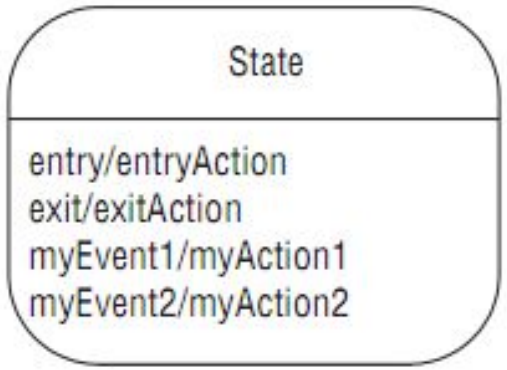
Диаграммы состояний

Основные понятия



Простой конечный автомат, описывающий вход в систему

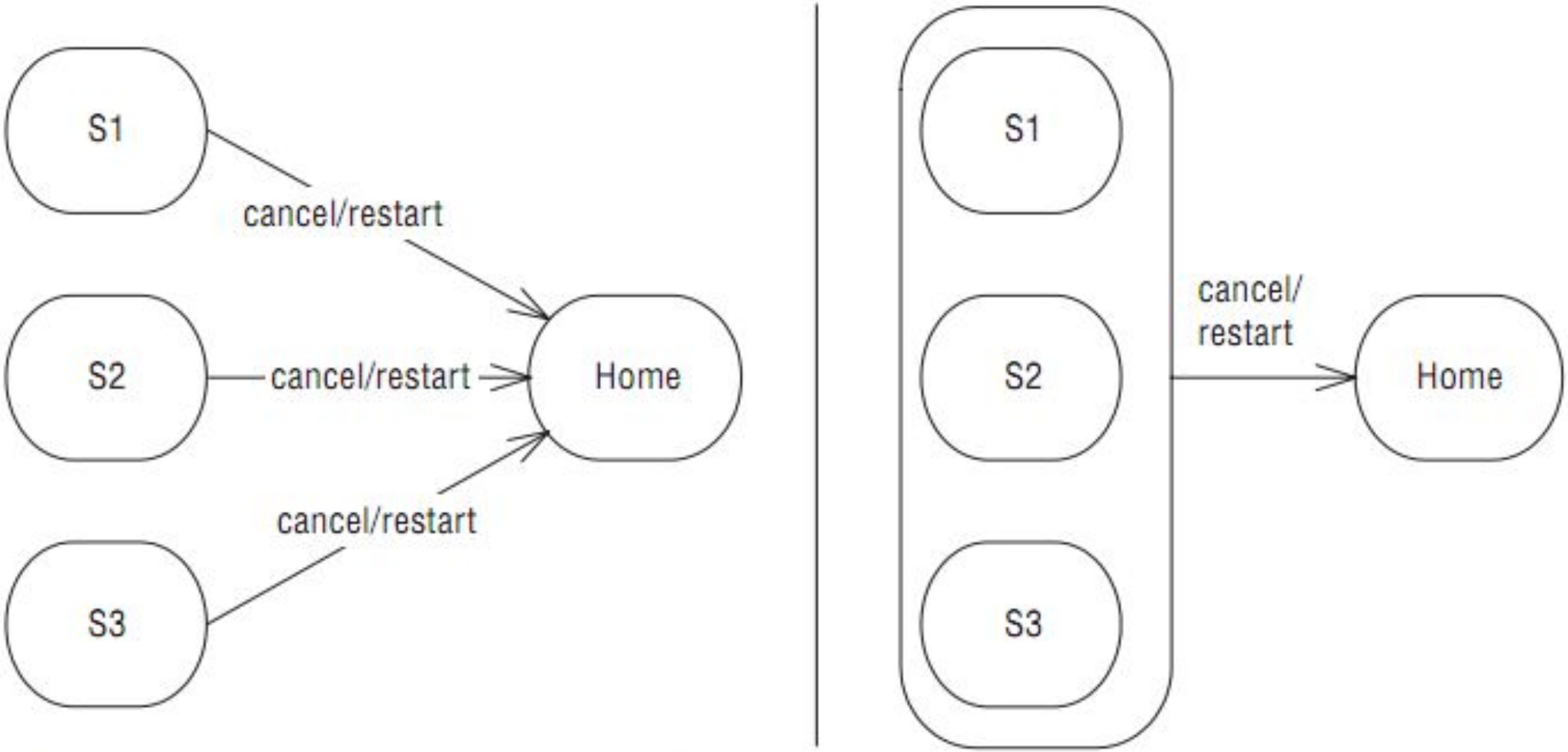
Специальные события



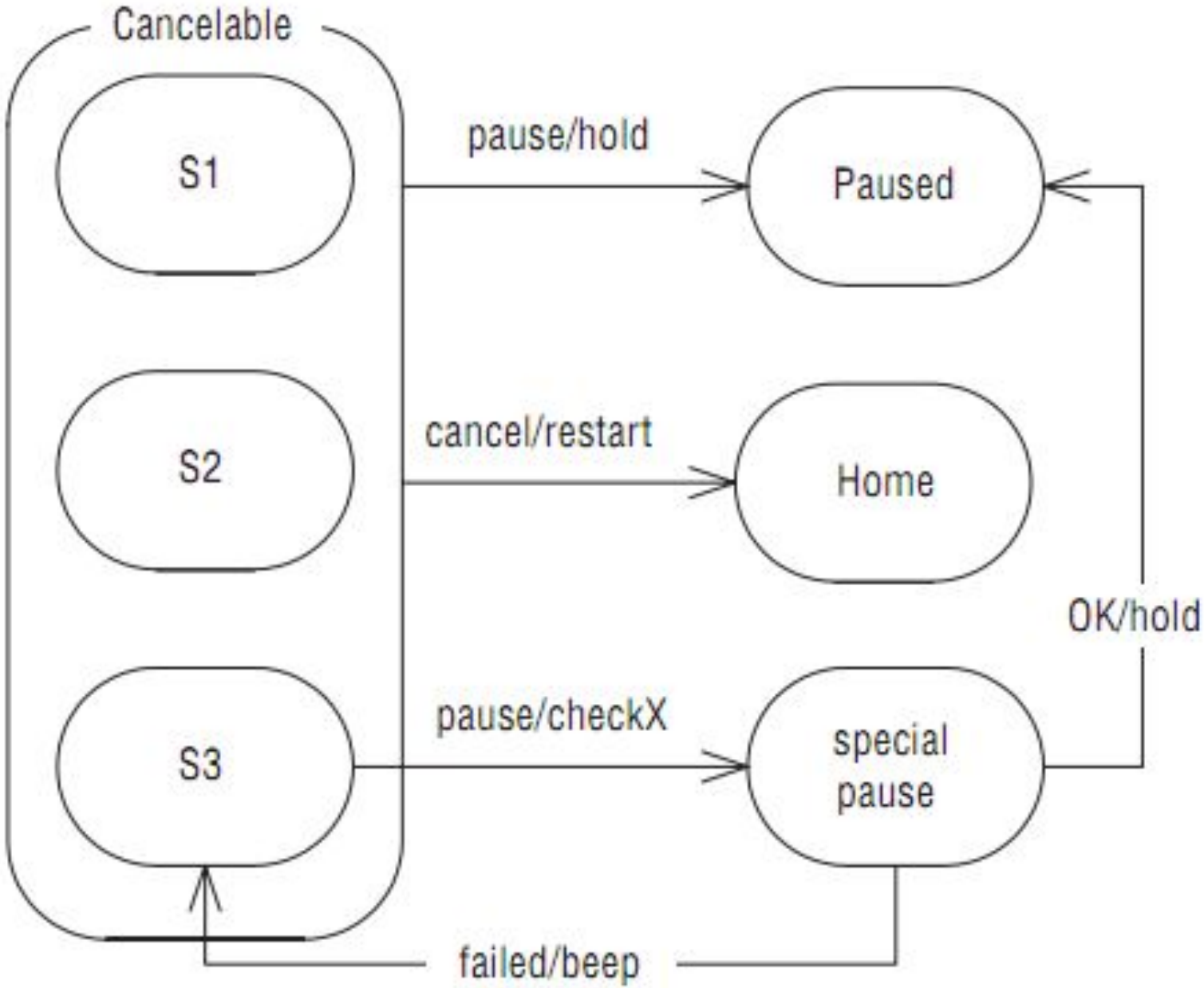
Изображение состояний и специальных событий в UML

Рефлексивный переход

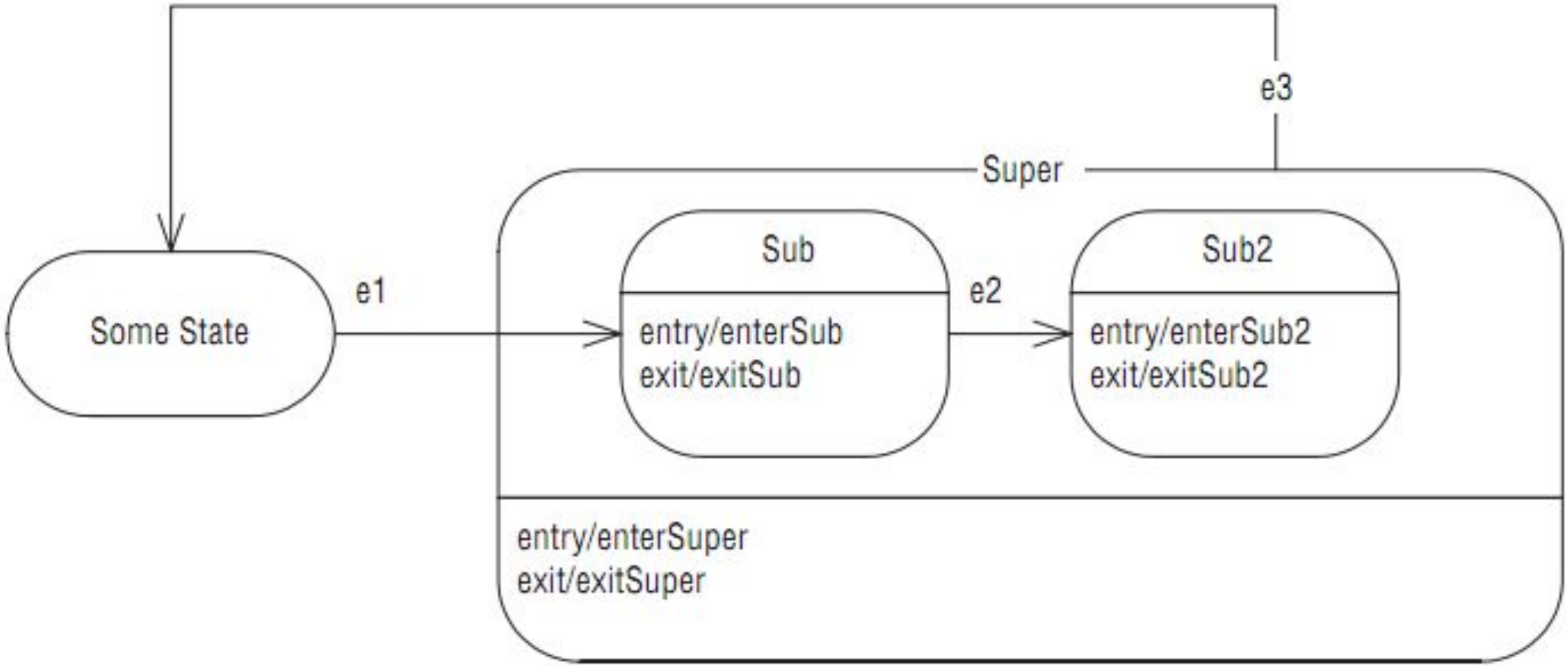
Суперстостояния



Переход: несколько состояний и одно суперсостояние

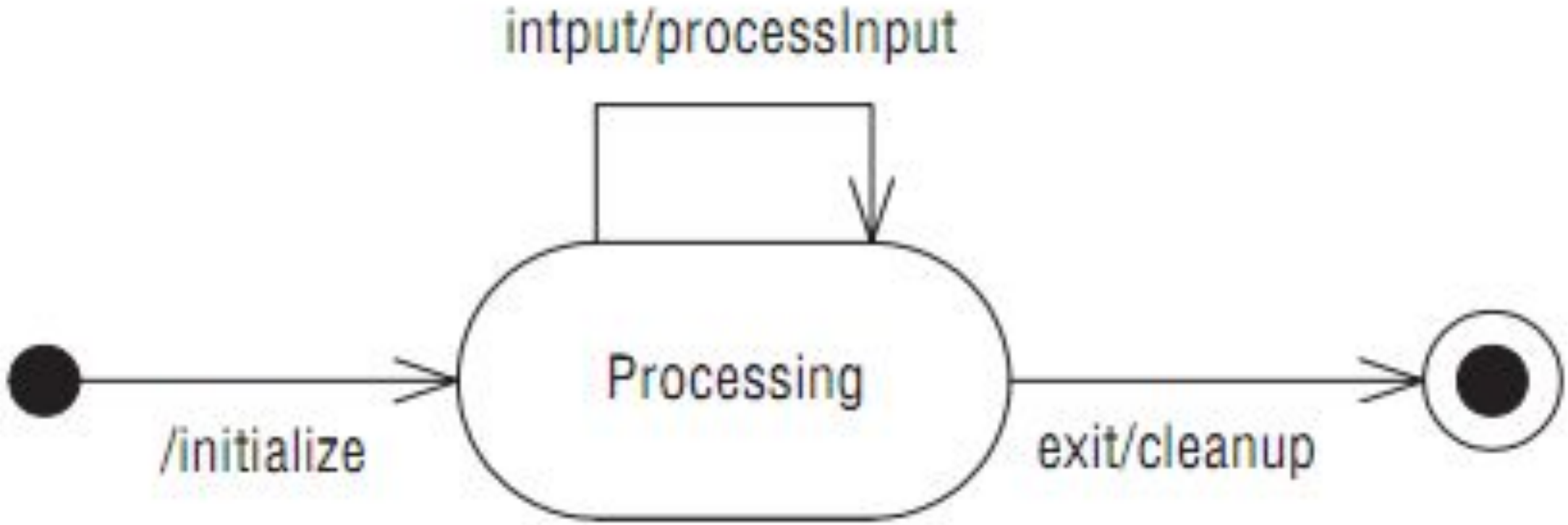


Переопределение переходов из суперсостояния



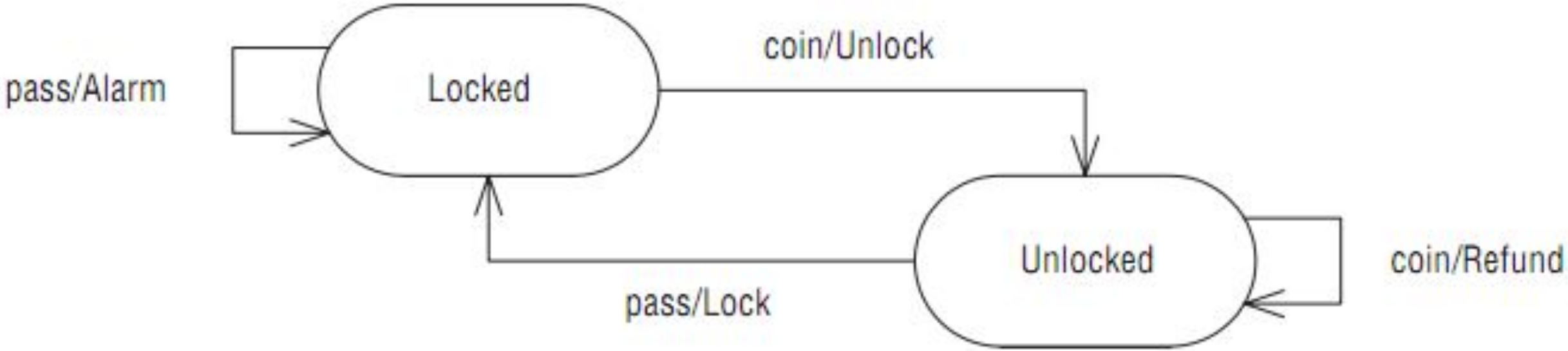
Иерархические вызовы действий при входе и выходе

Начальное и конечное псевдосостояния



Начальное и конечное псевдосостояния

Использование диаграмм состояний

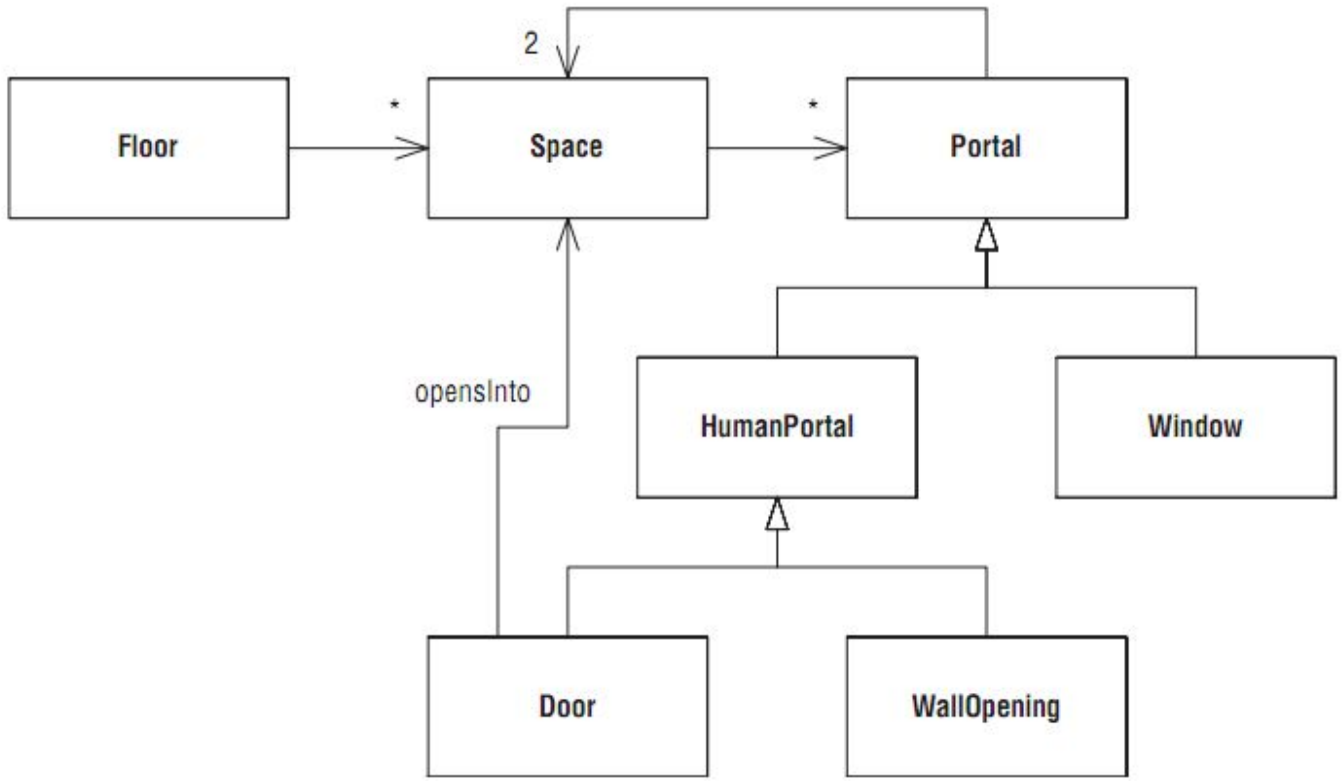


ДПС турникета в метро

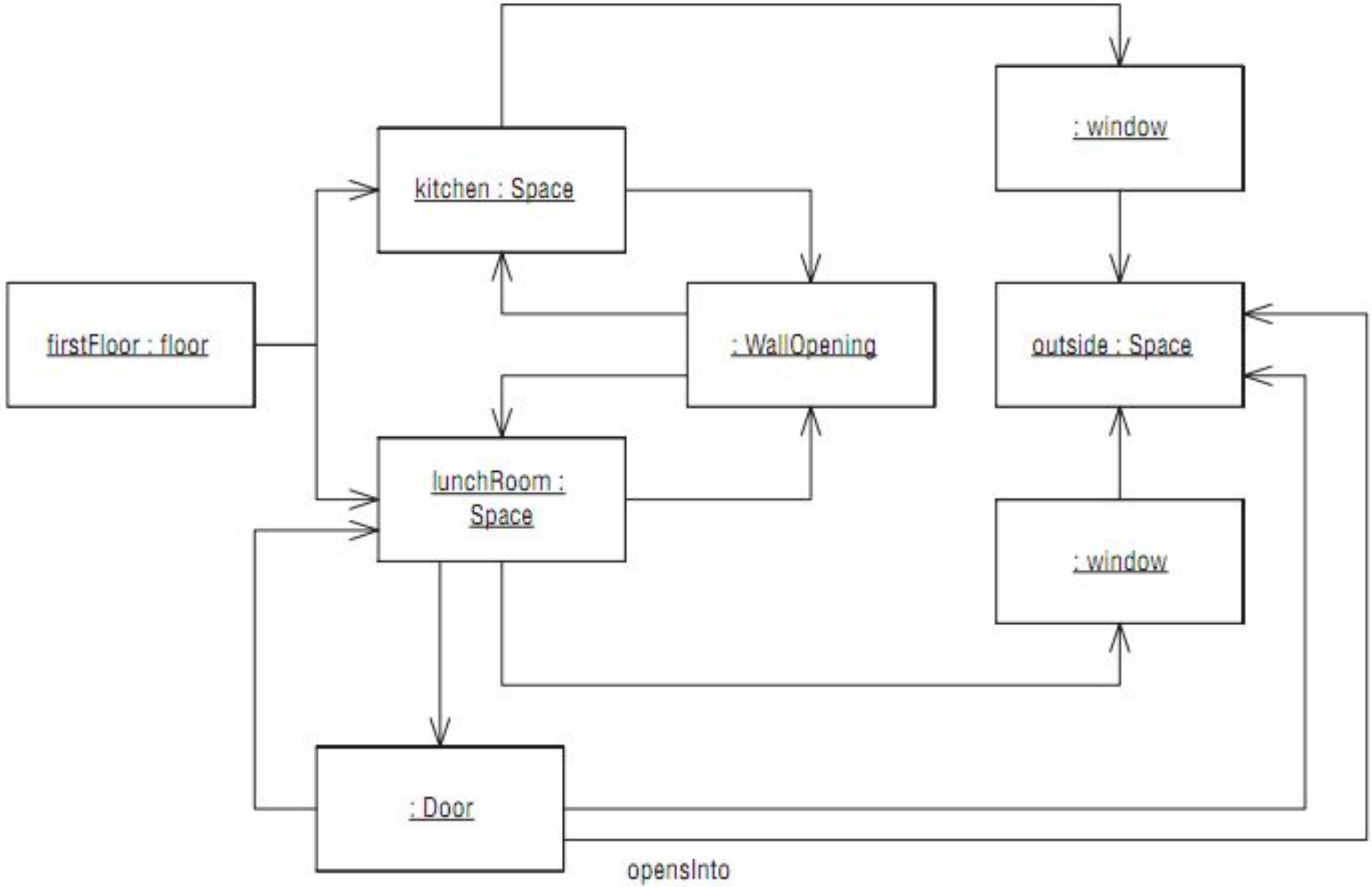
ТПС турникета в метро

Текущее состояние	Событие	Новое состояние	Действие
Locked	coin	Unlocked	Unlock
Locked	pass	Locked	Alarm
Unlocked	coin	Unlocked	Refund
Unlocked	pass	Locked	Lock

Диаграммы объектов



План этажа



Столовая и кухня

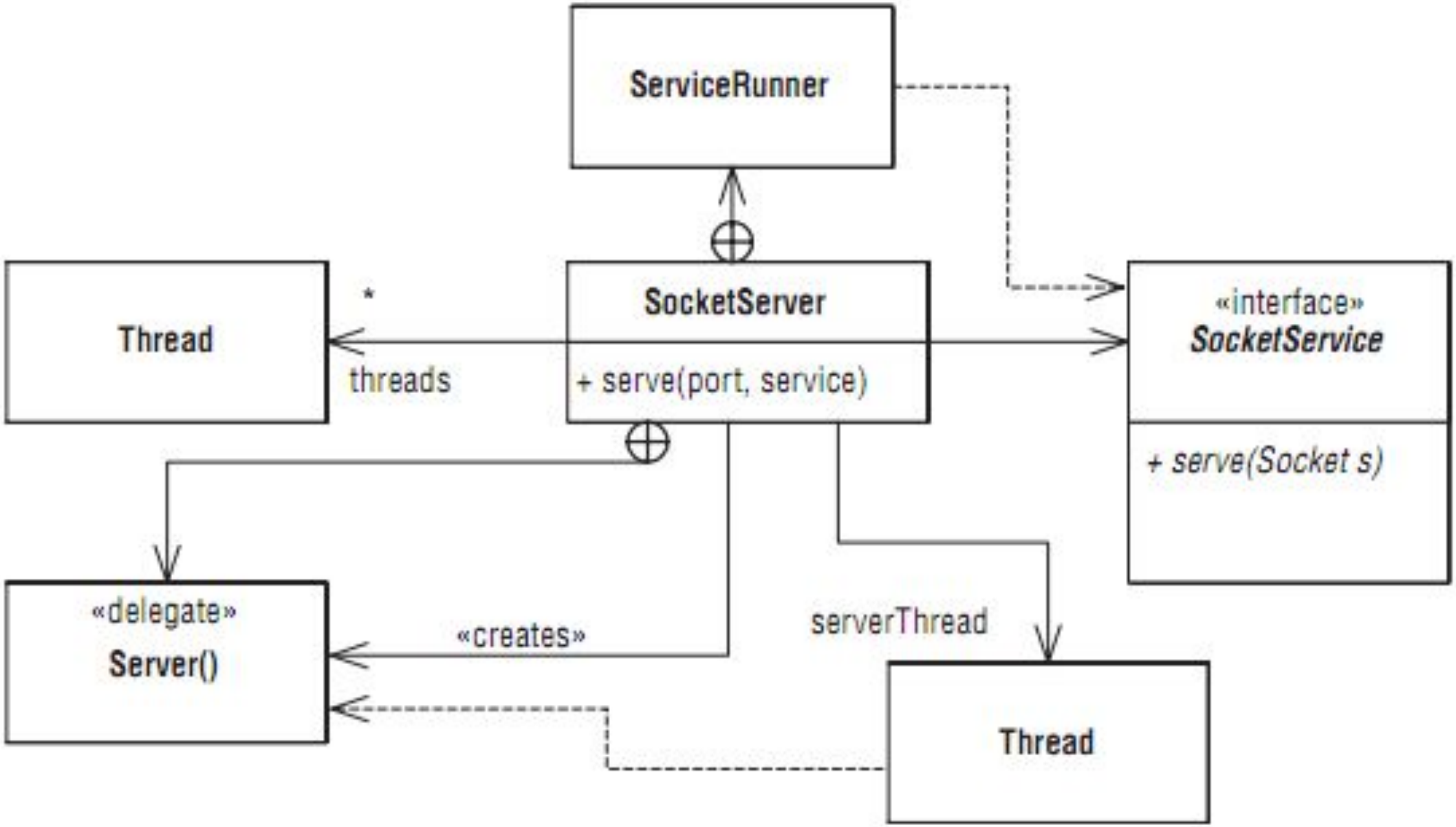


Диаграмма классов SocketServer

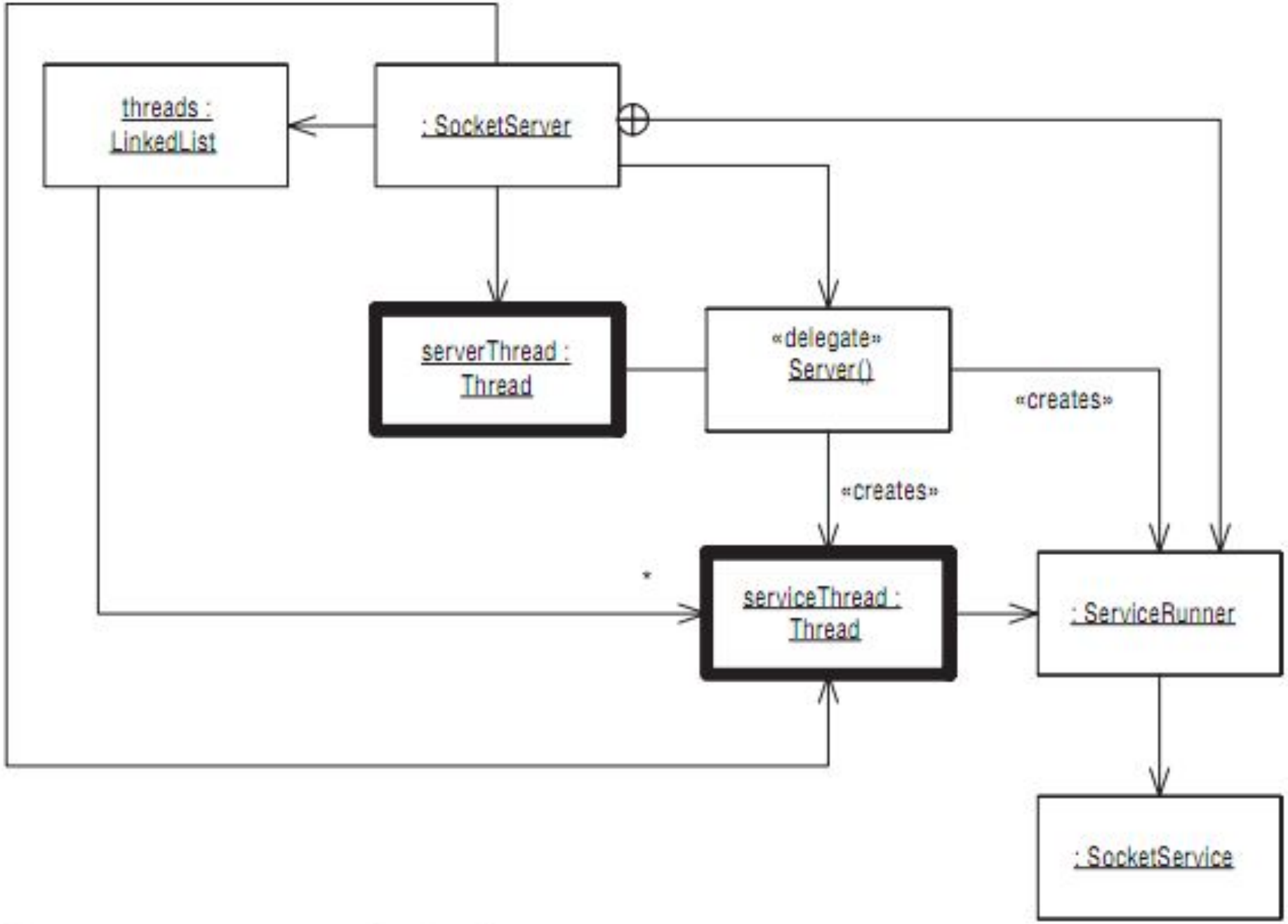


Диаграмма объектов SocketServer

Прецеденты (Use Cases)

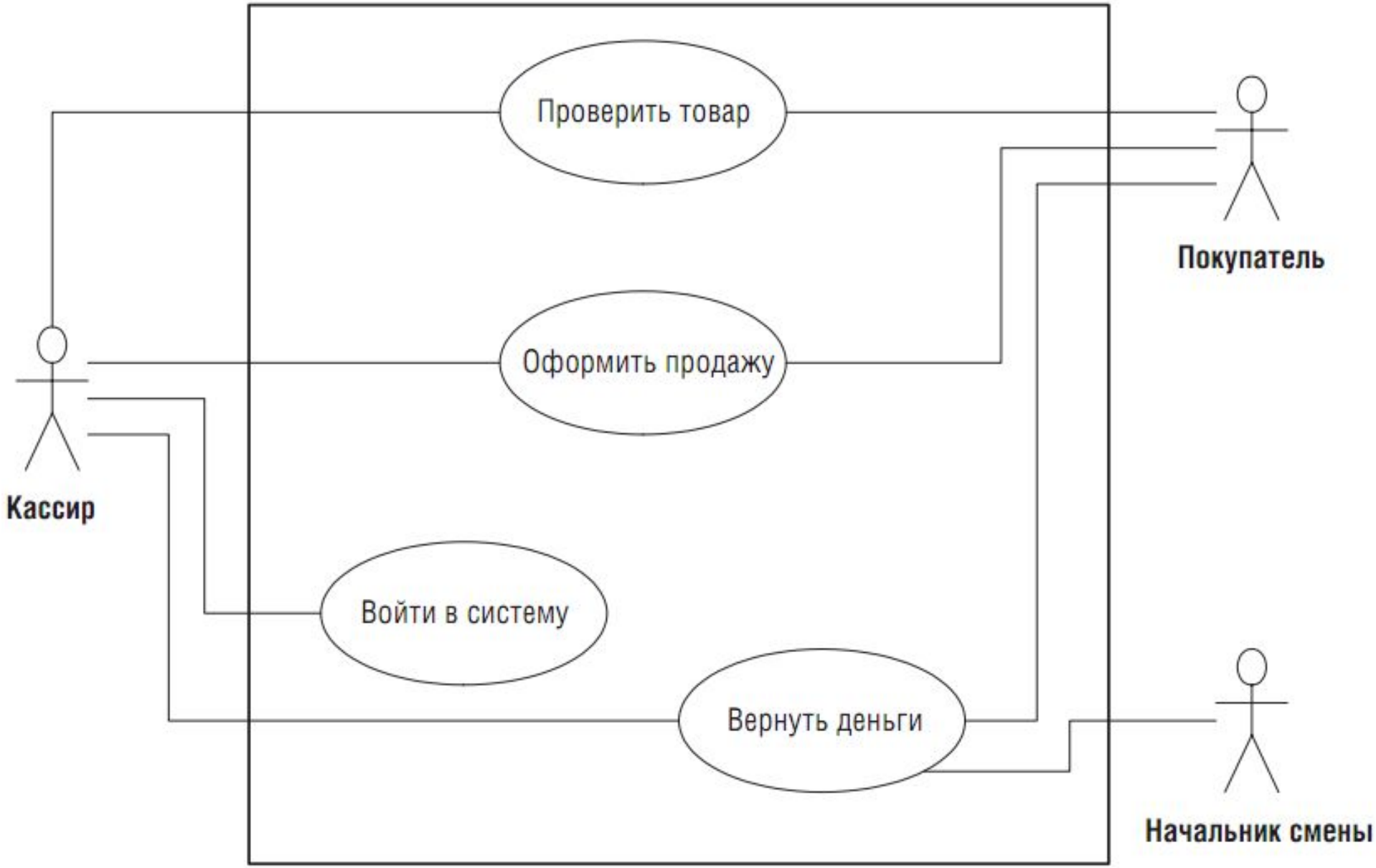
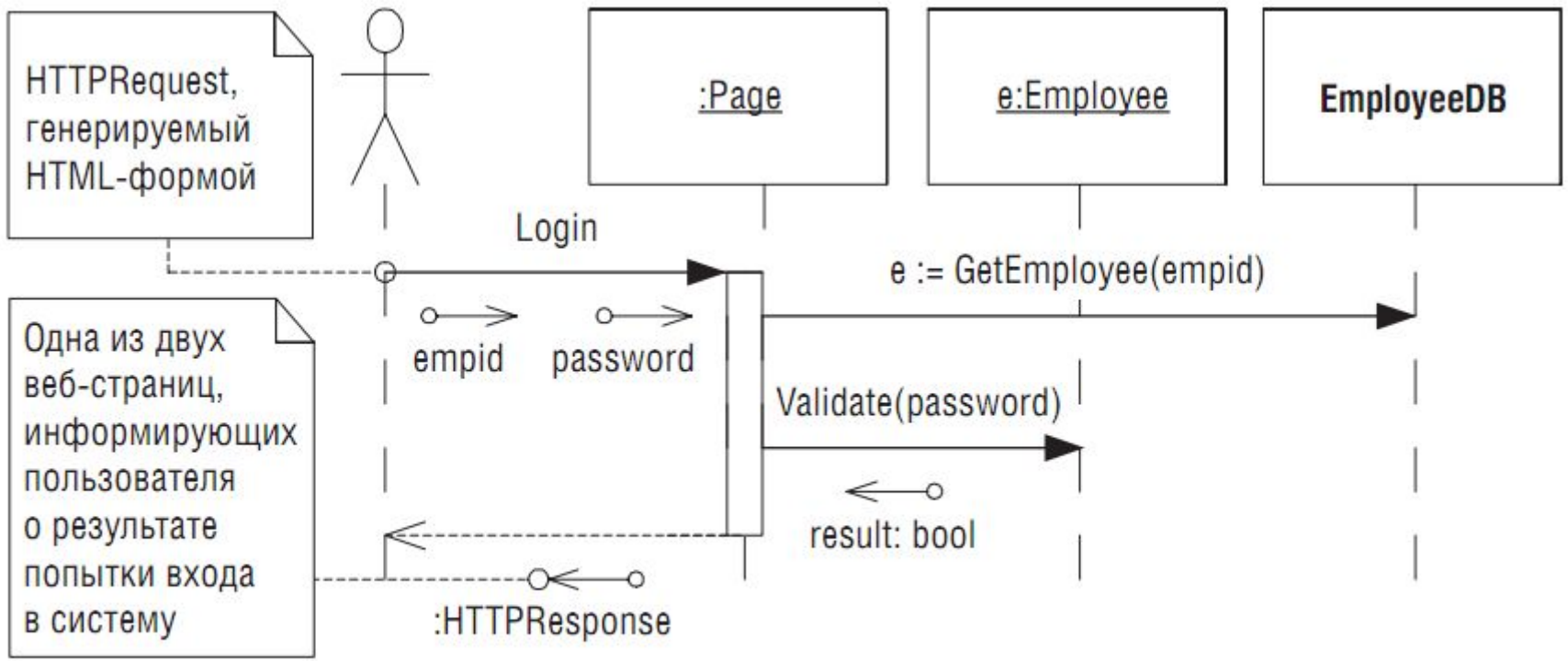


Диаграмма системных границ

Диаграммы последовательностей

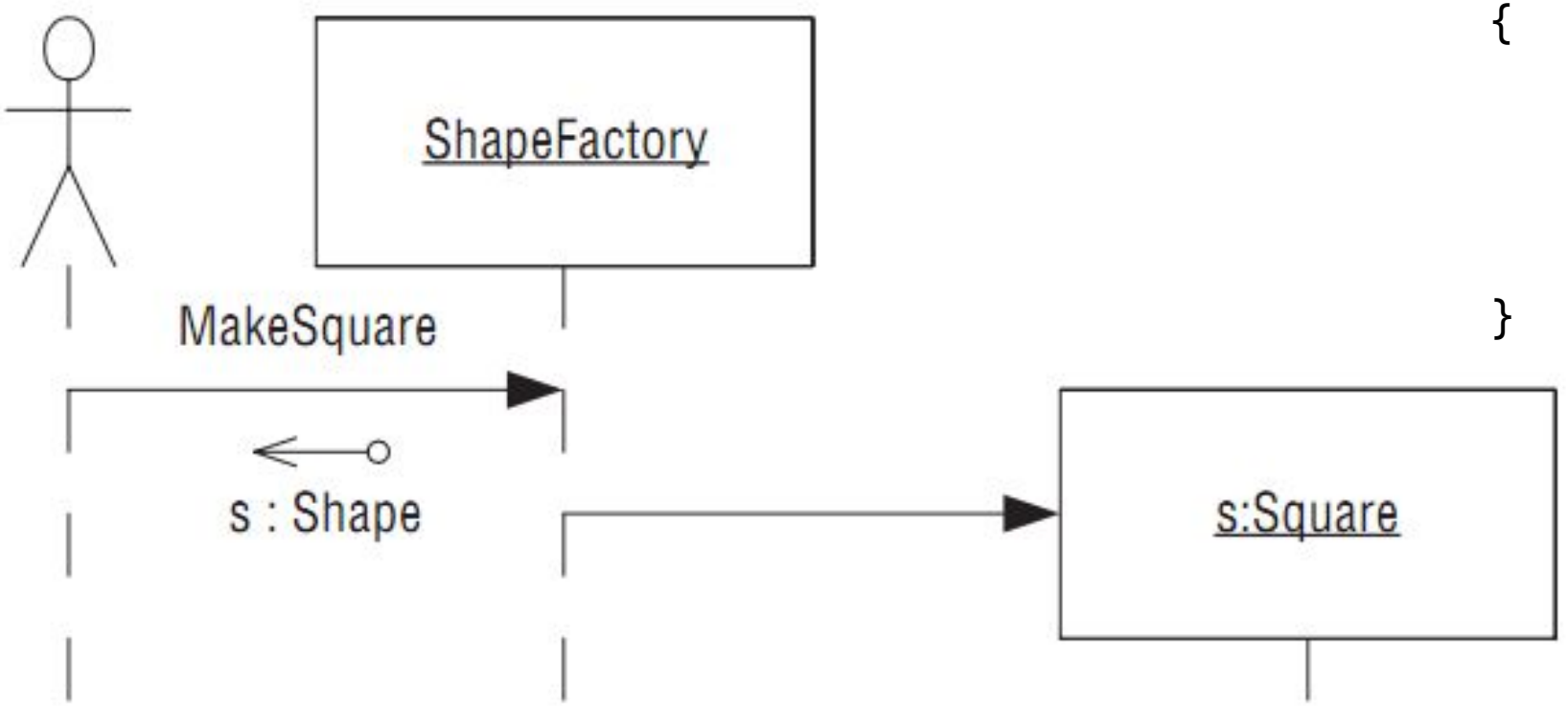
Основные понятия



Типичная диаграмма последовательности

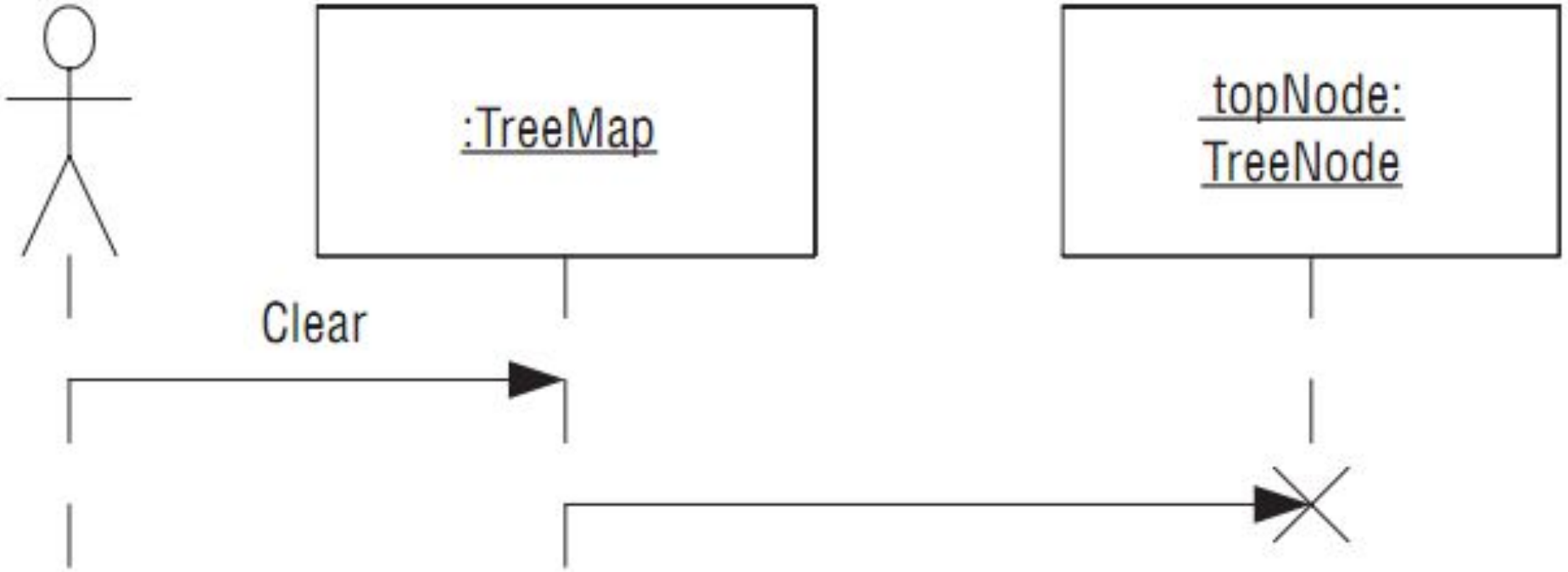
```
public class EmployeeDB
{
    public static Employee GetEmployee(string
empid)
    {
      //...
    }
    //...
}
```

Создание и уничтожение



Создание объекта

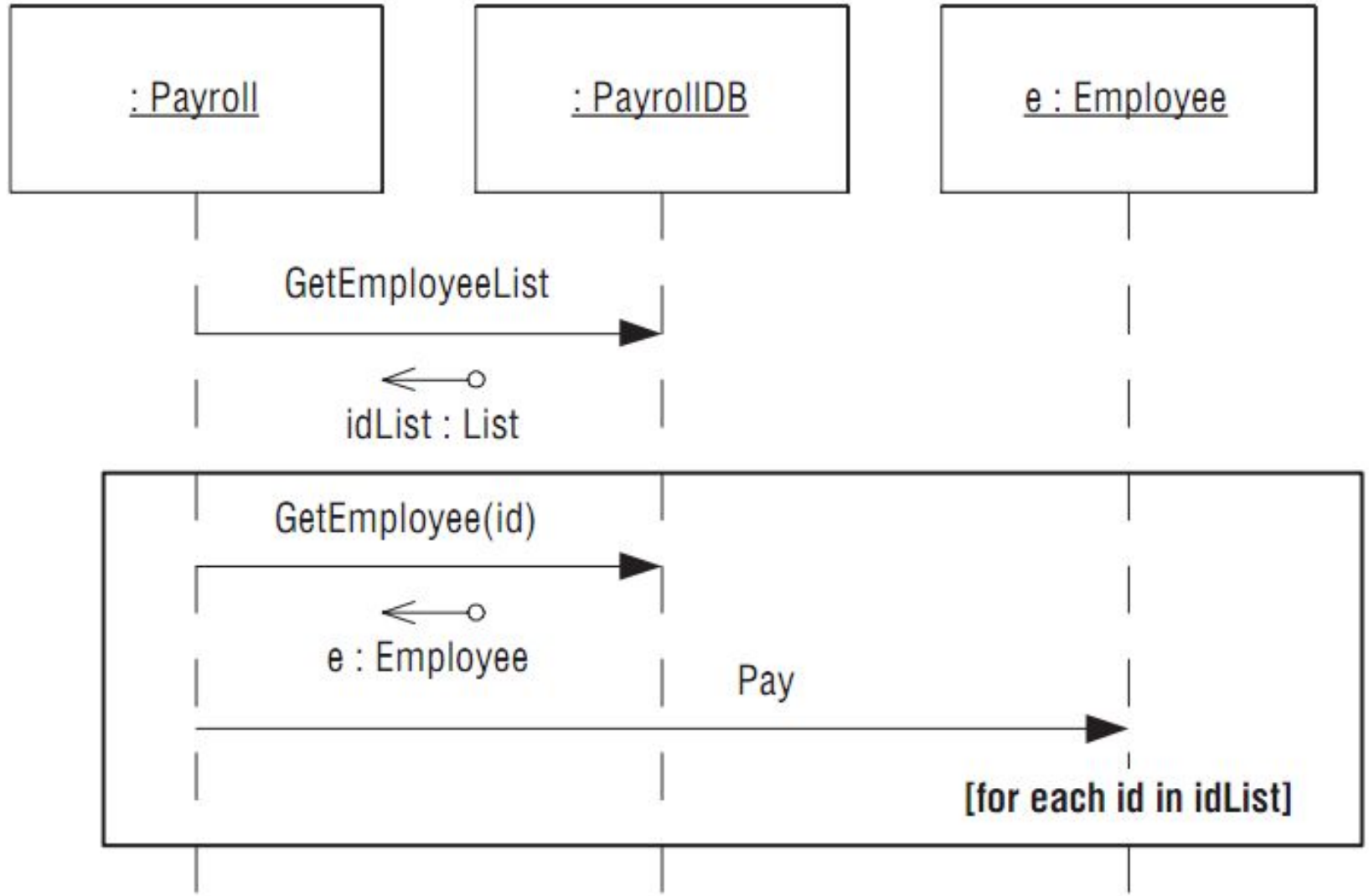
```
public class ShapeFactory
{
    public Shape MakeSquare()
    {
        return new Square();
    }
}
```



Передача объекта сборщику мусора

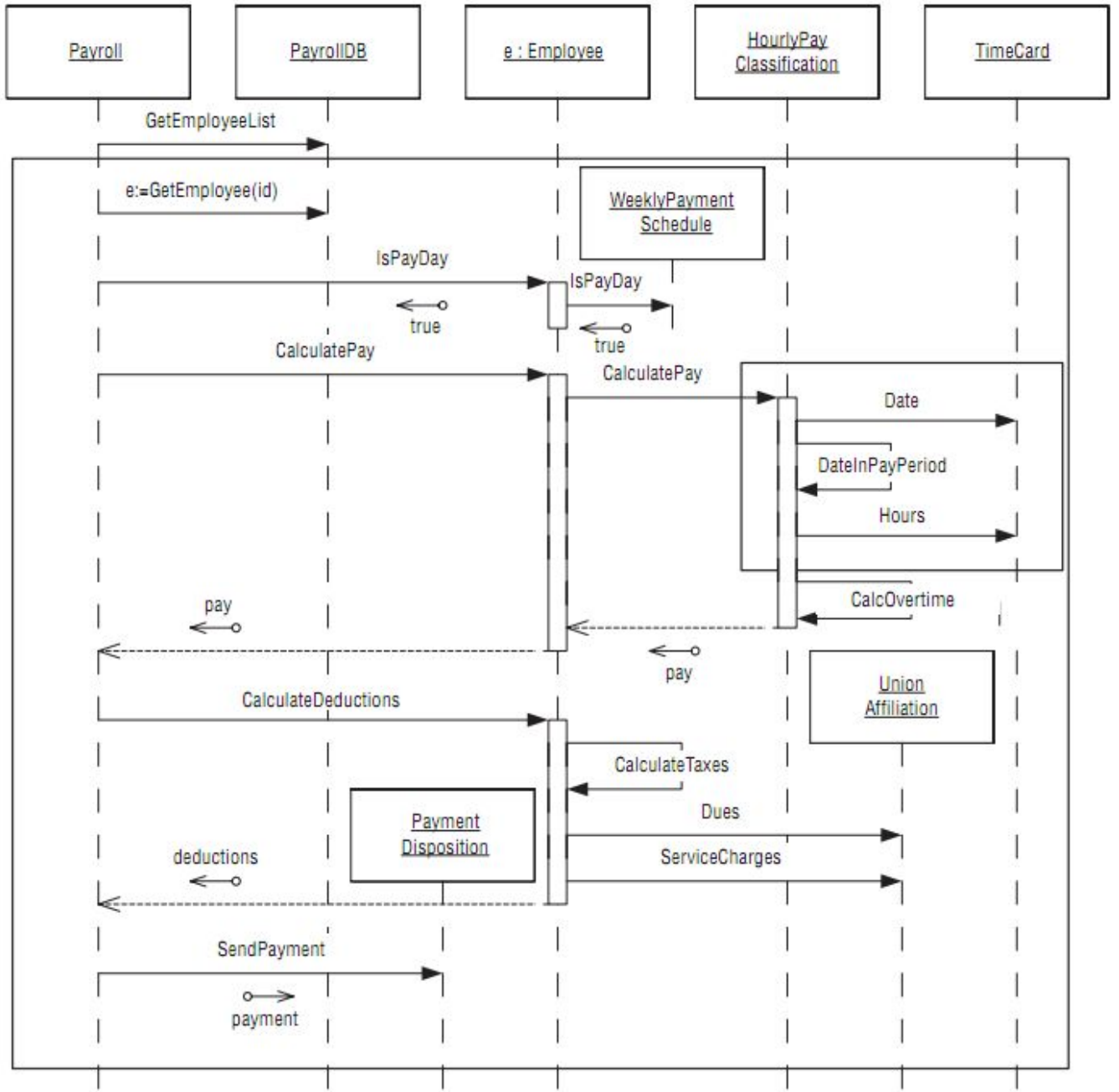
```
public class TreeMap
{
    private TreeNode topNode;
    public void Clear()
    {
        topNode = null;
    }
}
```

Простые циклы



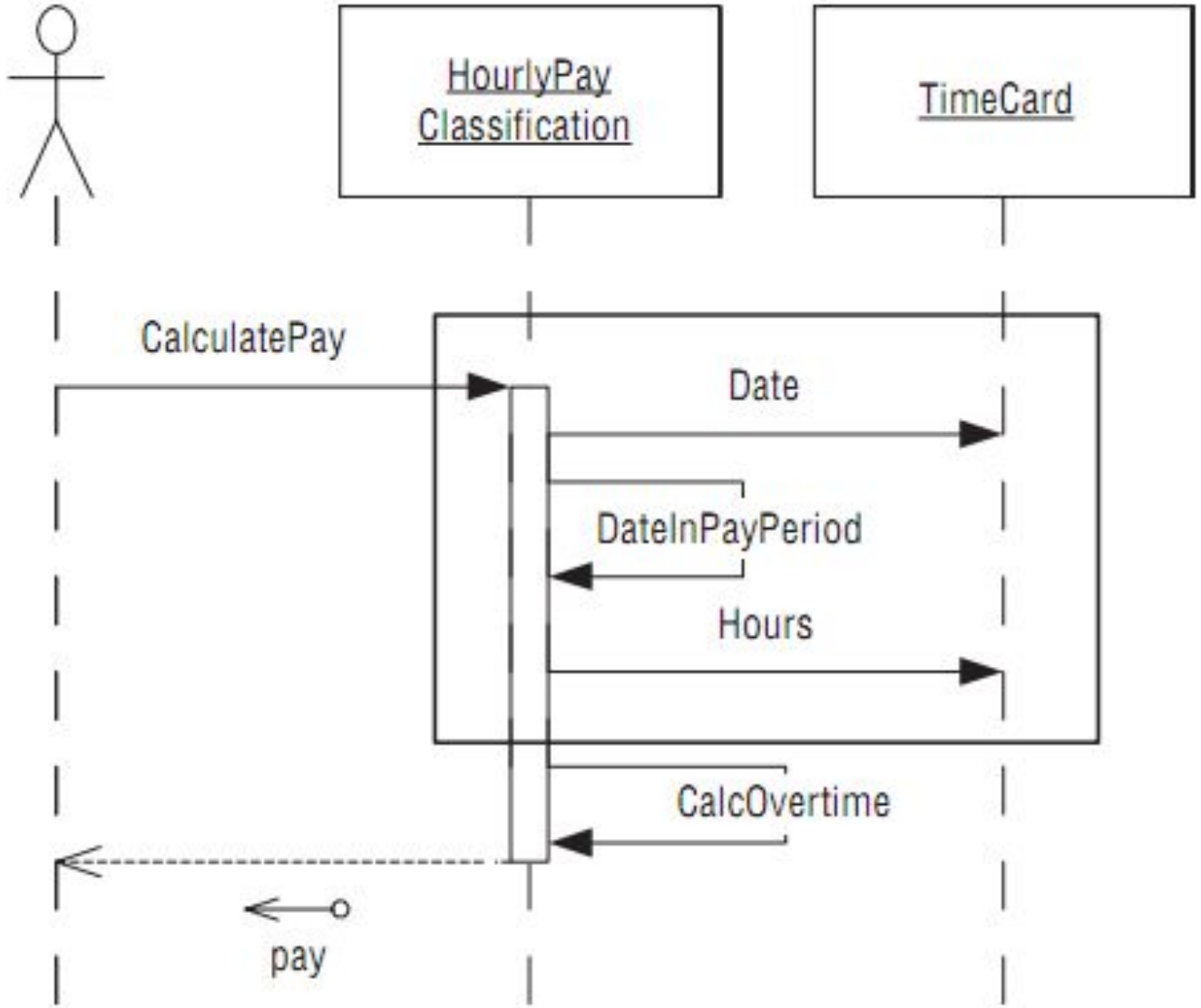
Простой цикл

Различные сценарии

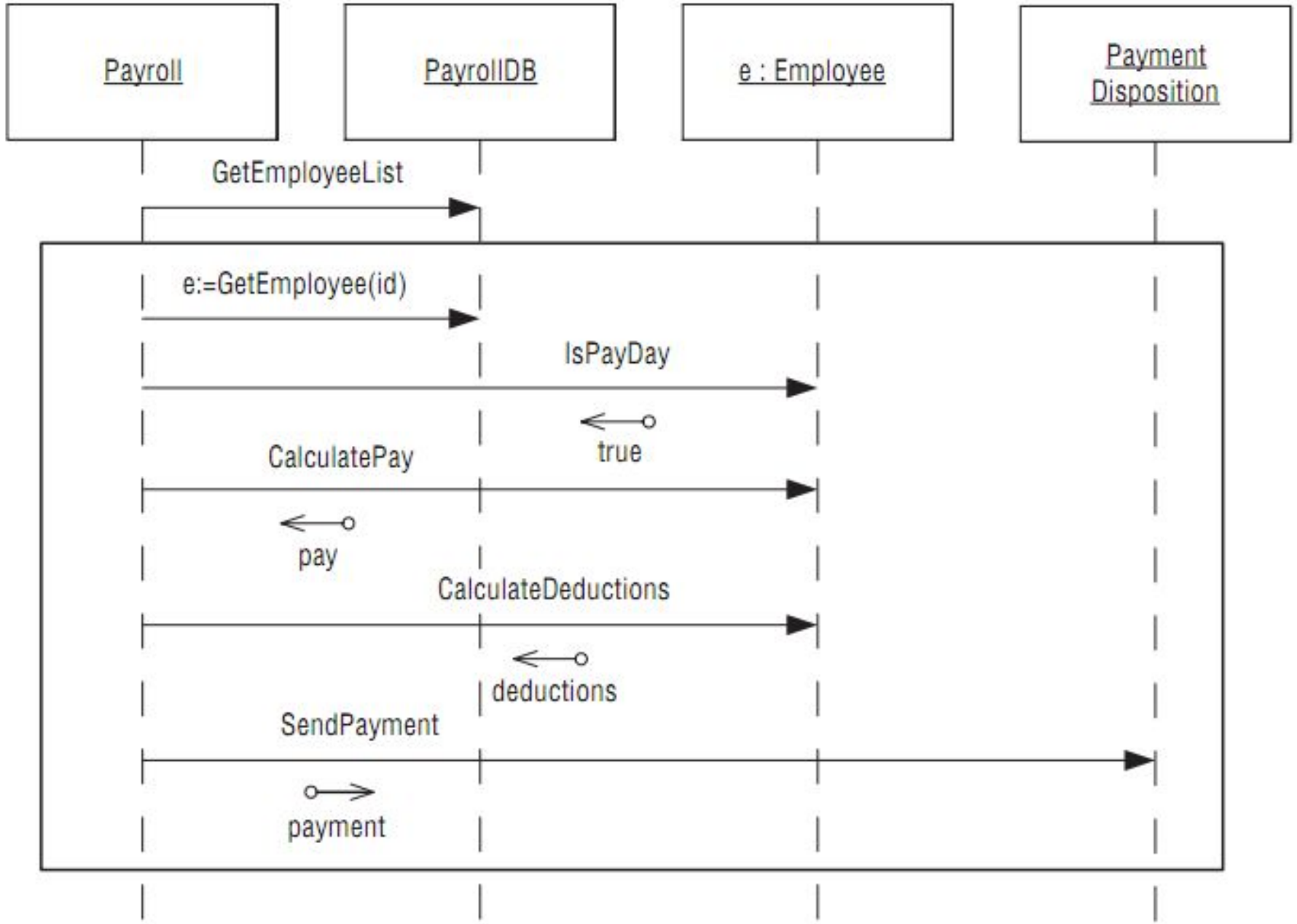


Чрезмерно сложная диаграмма последовательности

```
public class Payroll
{
    private PayrollDB itsPayrollDB;
    private PaymentDisposition itsDisposition;
    public void DoPayroll()
    {
        ArrayList employeeList = itsPayrollDB.GetEmployeeList();
        foreach (Employee e in employeeList)
        {
            if (e.IsPayDay())
            {
                double pay = e.CalculatePay();
                double deductions = e.CalculateDeductions();
                itsDisposition.SendPayment(pay - deductions);
            }
        }
    }
}
```



Один небольшой сценарий



Высокоуровневое представление

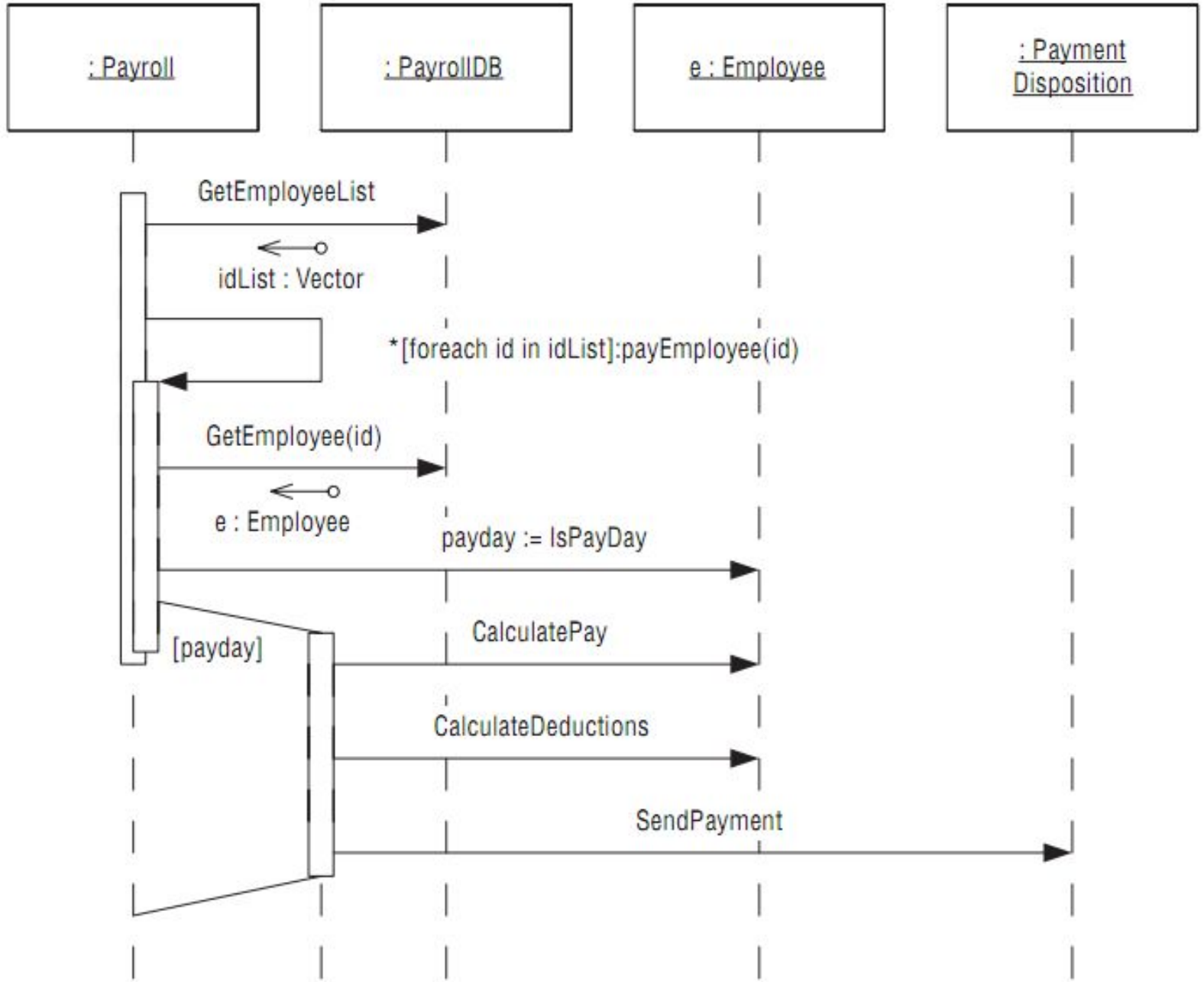
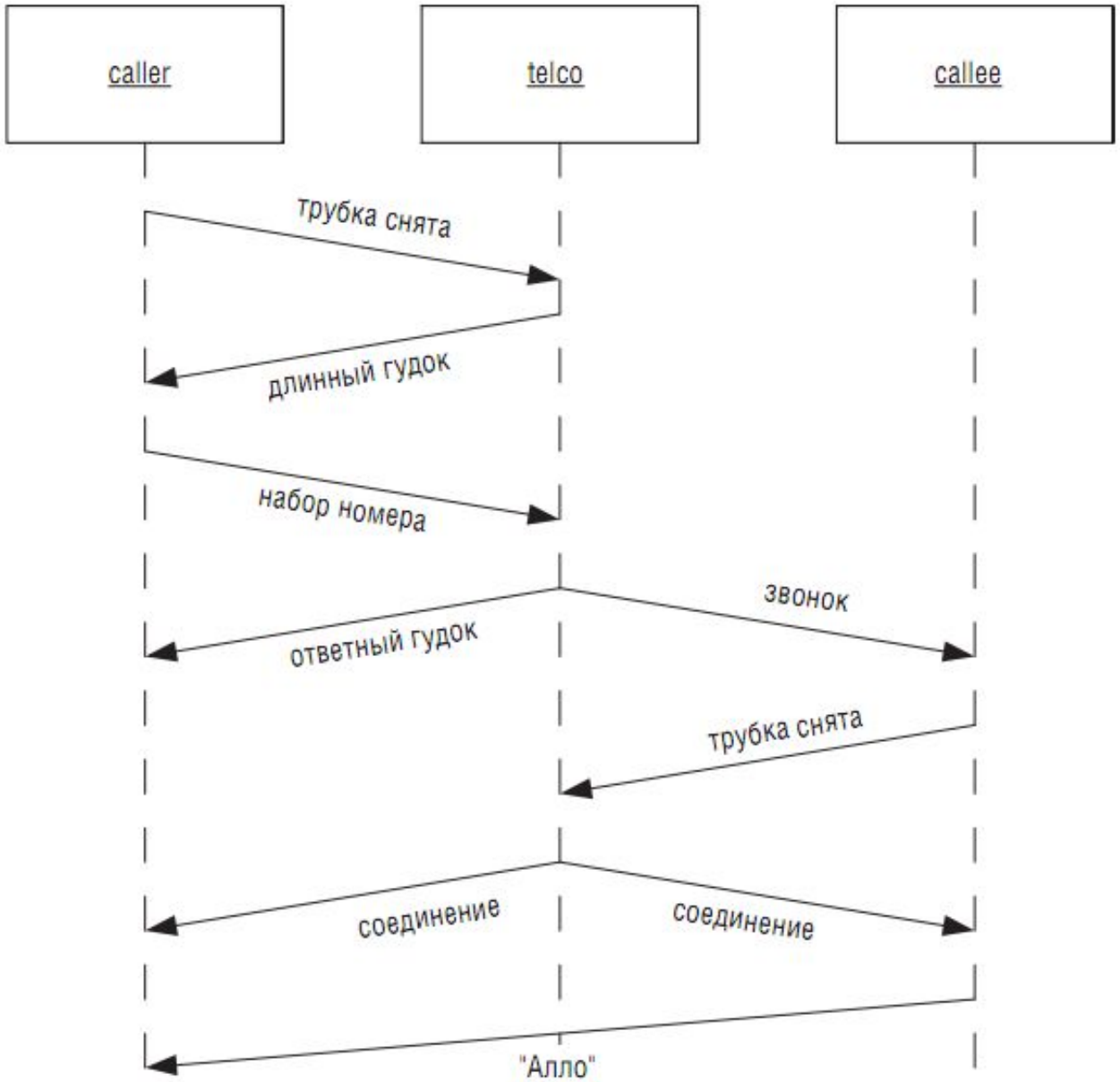
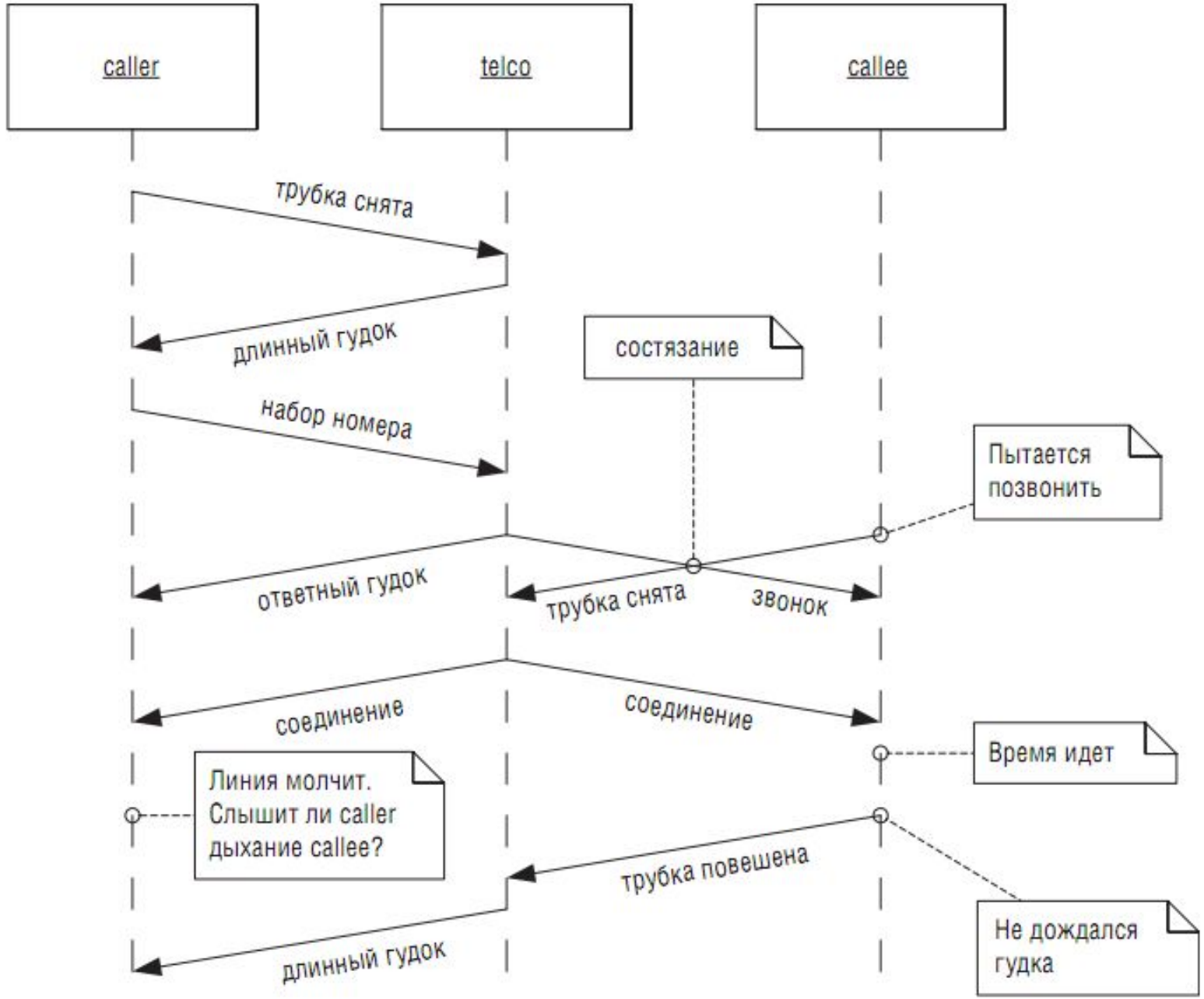


Диаграмма последовательности с циклами и условиями

Сообщения, занимающие время

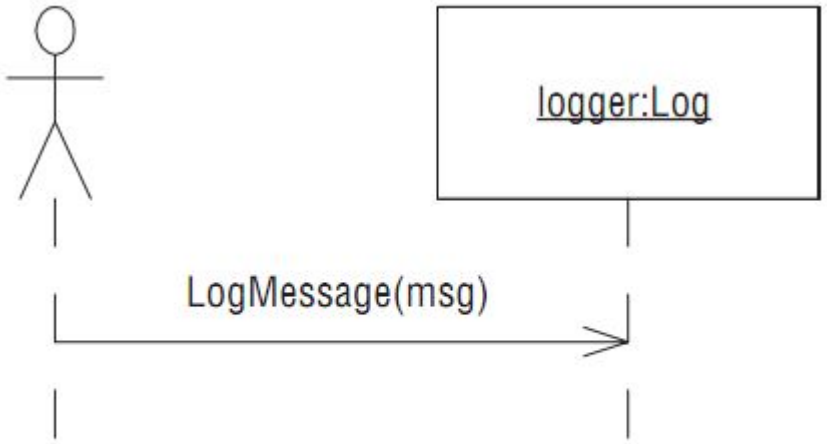


Обычный телефонный вызов

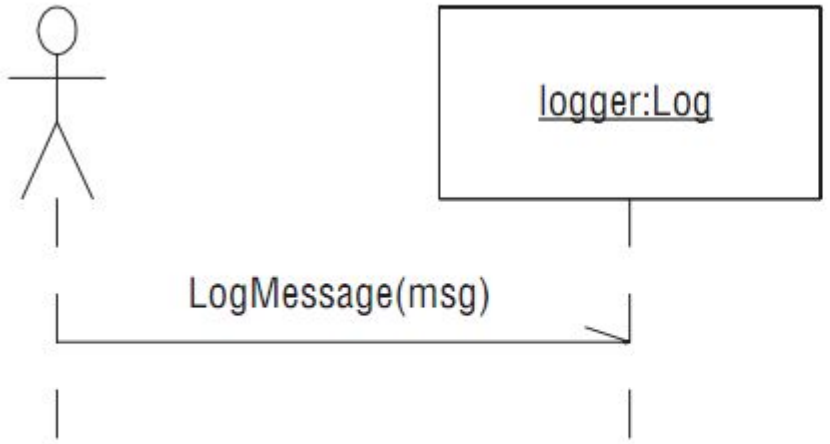


Несостоявшийся телефонный звонок

Асинхронные сообщения

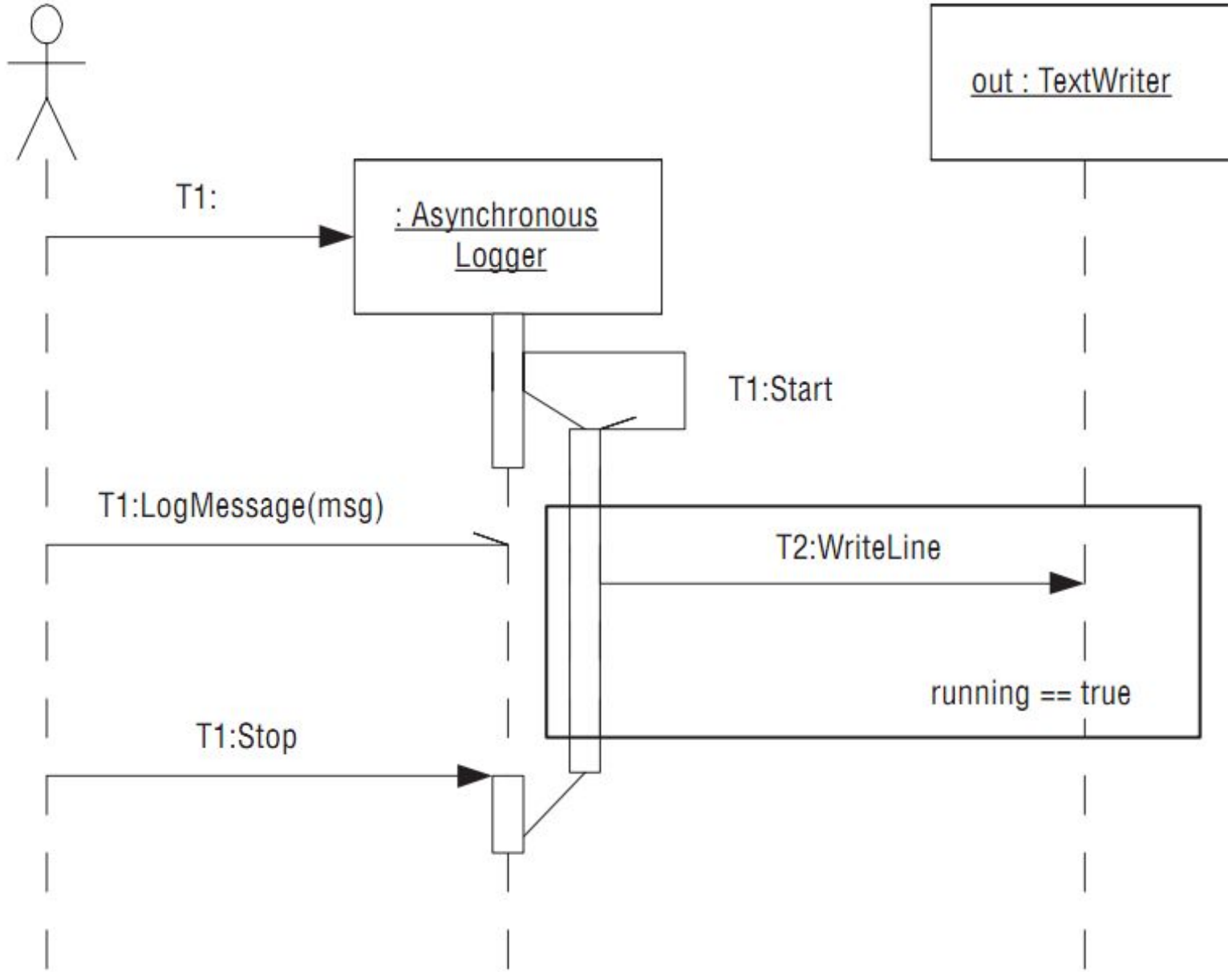


Асинхронное сообщение



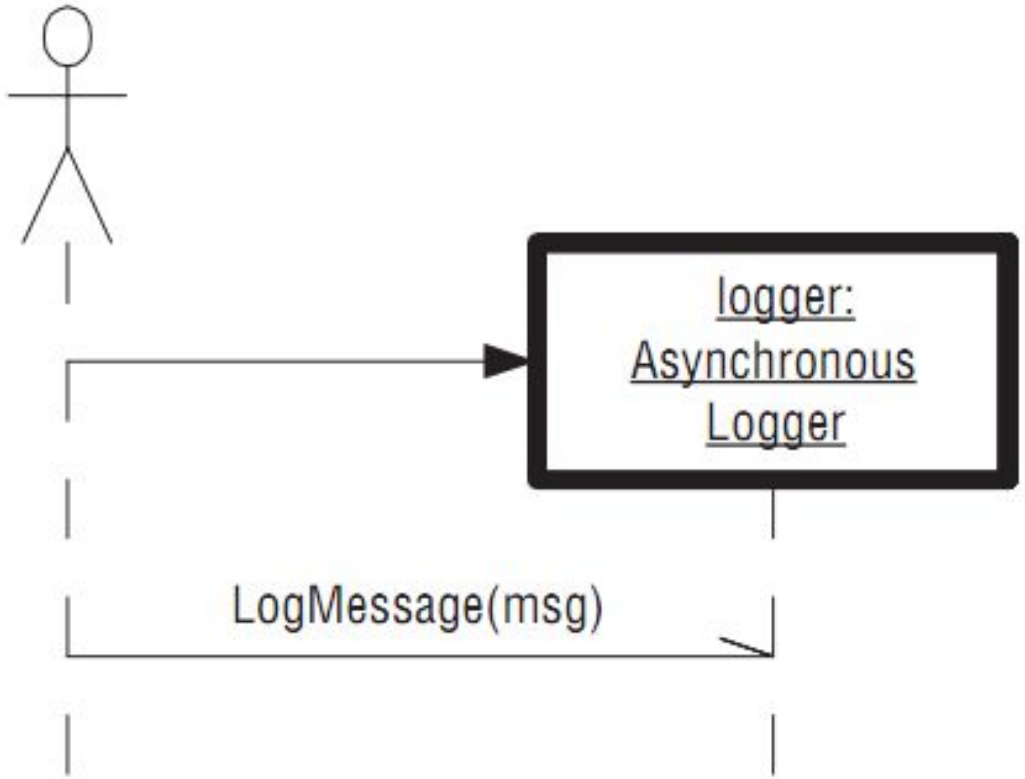
Прежний и лучший способ изображения асинхронных сообщений

Несколько потоков



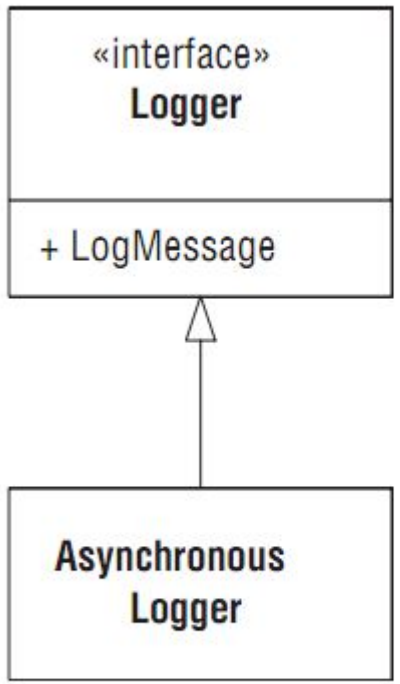
Несколько потоков управления

Активные объекты



Активный объект

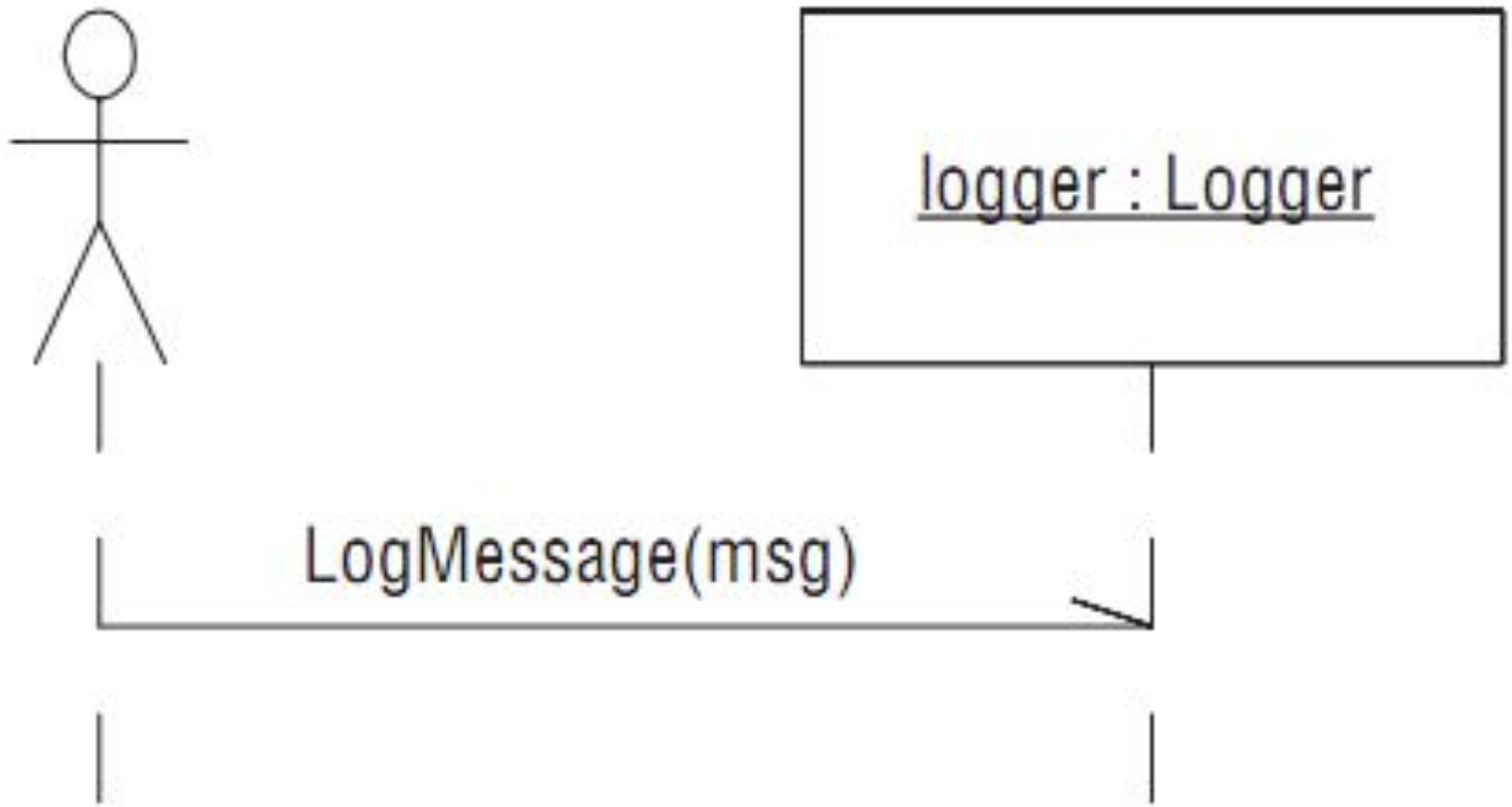
Отправка сообщений интерфейсам



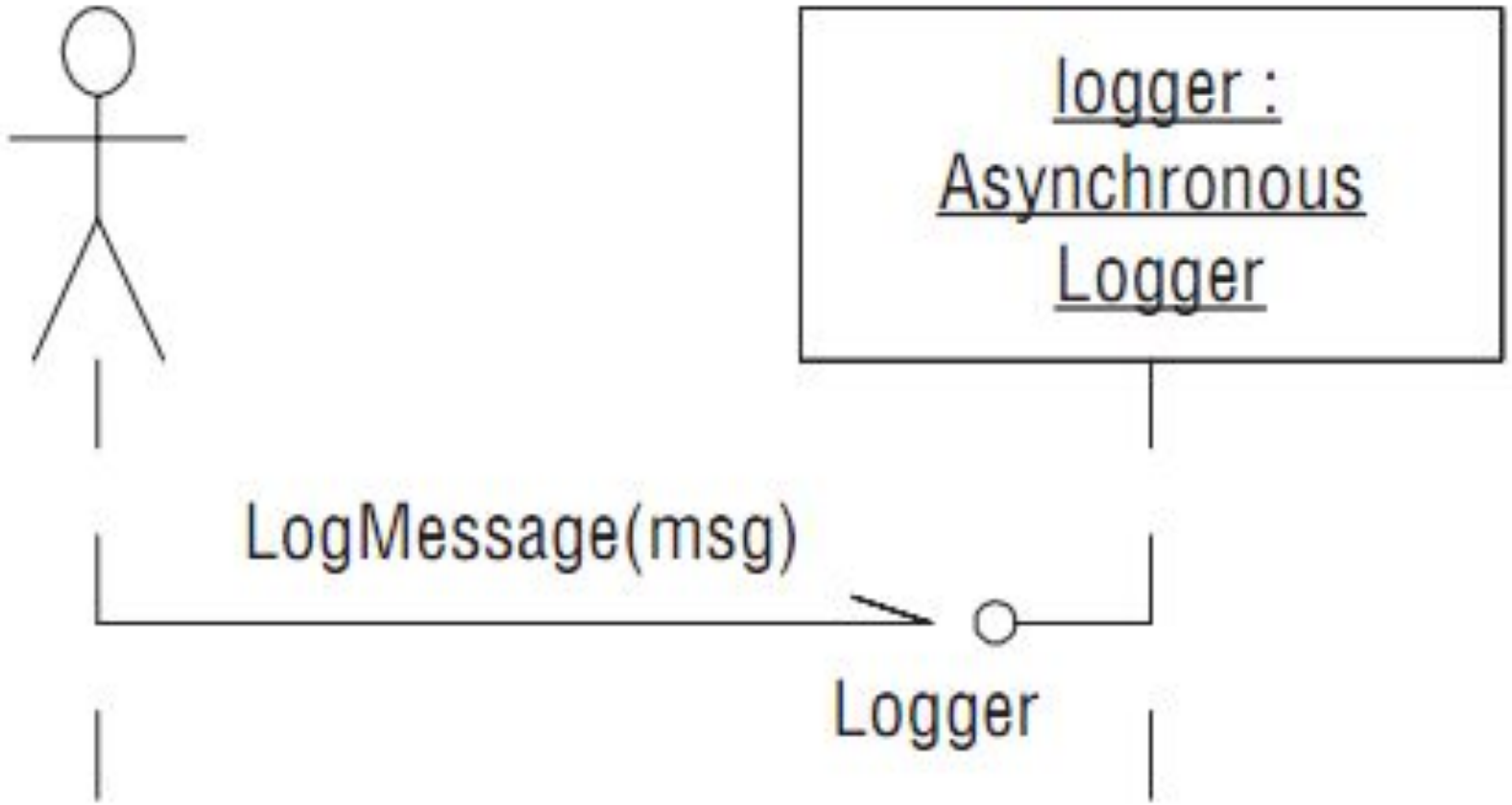

```
interface Logger {
  void LogMessage(string msg);
}

public class AsynchronousLogger : Logger {
  ...
}
```

Дизайн простого регистратора



Отправка сообщения интерфейсу



Отправка производному типу через интерфейс