4.2. Любовный треугольник MVC

Классический архитектурный паттерн и паттерн проектирования

Триада **модель-представление-контроллер (**MVC - Model-View-Controller**)** может интерпретироваться очень широко: от архитектурного принципа независимости описания бизнес-модели от ее визуализации и поведения до конкретного паттерна проектирования, например, для работы с отдельным свойством бизнес-объекта.

Рассмотрим, как выглядит паттерн MVC в его первозданном описании [3-6, 41-3] (рис.4-5):

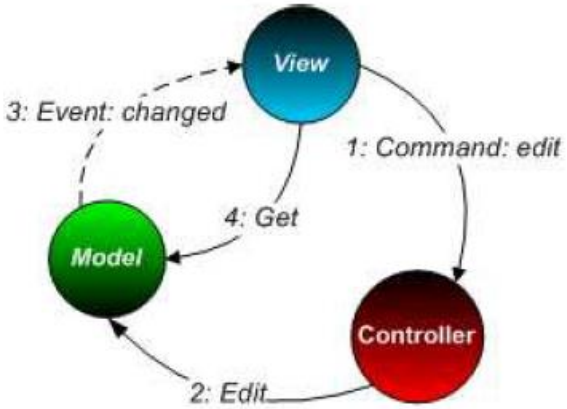

Рис.4-5. Классический MVC

- модель описывает бизнес-сущность и ее атомарное поведение, она существует автономно и способна менять внутреннее состояние под воздействием извне при помощи набора методов;
- представление может запросить и получить данные от модели;
- при изменении данных модель может уведомить об этом представление, для этого представление подписывается на соответствующие события (паттерн *Observer*, см.3.3);
- для изменения модели представление передает команду контроллеру, который разворачивает ее в набор действий над моделью.

Рисунок 4-6 показывает диаграмму последовательности для такого поведения. Очевидно, что такой вид паттерна годится, например, для отображения отдельного значения свойства какого-либо бизнес-объекта. Например, в бизнес-объекте имеется свойство – время, оставшееся до некоторого события. Пусть это будет таймер микроволновки. В представлении (экранной форме) ему будет соответствовать поле, а само представление будет подписано на событие – изменение указанного свойства. Событие будет генерироваться моделью каждую секунду, если установлен обратный отсчет. Кроме того, представление может запросить соответствующее значение у модели соответствующим get-методом.

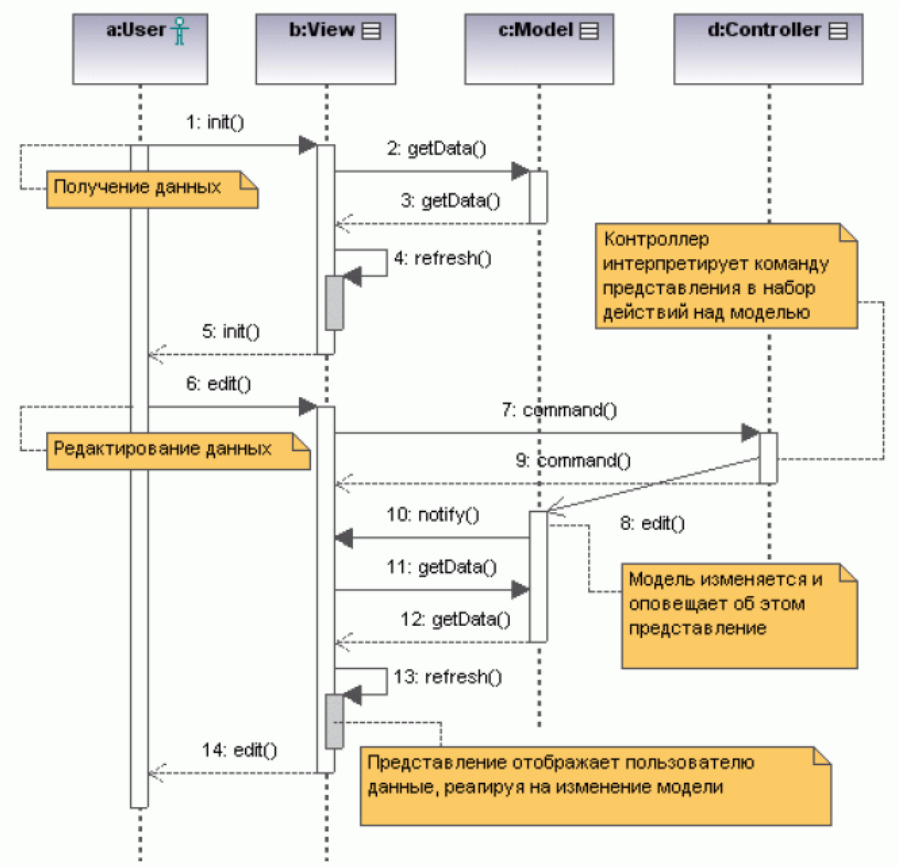

Рис.4-6. Поведение классического MVC

Изменить значение указанного свойства прямым присваиванием нельзя. Для этого служит контроллер, который проверяет возможность такого изменения и выполняет последовательность команд, например, сбрасывает модель и заново запускает ее с новым интервалом.

В таком виде паттерн далеко не идеален. Отметим некоторые недостатки:

- если представление содержит значительное количество полей отображения или связано с несколькими бизнес-объектами, то получится настоящий клубок ссылок;
- контроллер не всегда отслеживает состояние модели, а инициируется только командами представления, события модели контроллеру не передаются.

Архитектурный MVC: контроллер – управление долгосрочным поведением представления

Указанные недостатки шаблона MVC отмечены также в [41-4]. Там предлагается сделать контроллер посредником между моделью и представлением, отвечающим за *binding* – взаимное соответствие содержания полей представления и свойств бизнес-объекта.

Если же рассматривать MVC на уровне архитектуры, то под представлением следует понимать экранную форму (диалог) или набор основных форм, в которых осуществляется весь бизнес-процесс. Тогда структура и поведение шаблона должны коренным образом измениться. Ключевые идеи такого архитектурного MVC (рис.4-7):

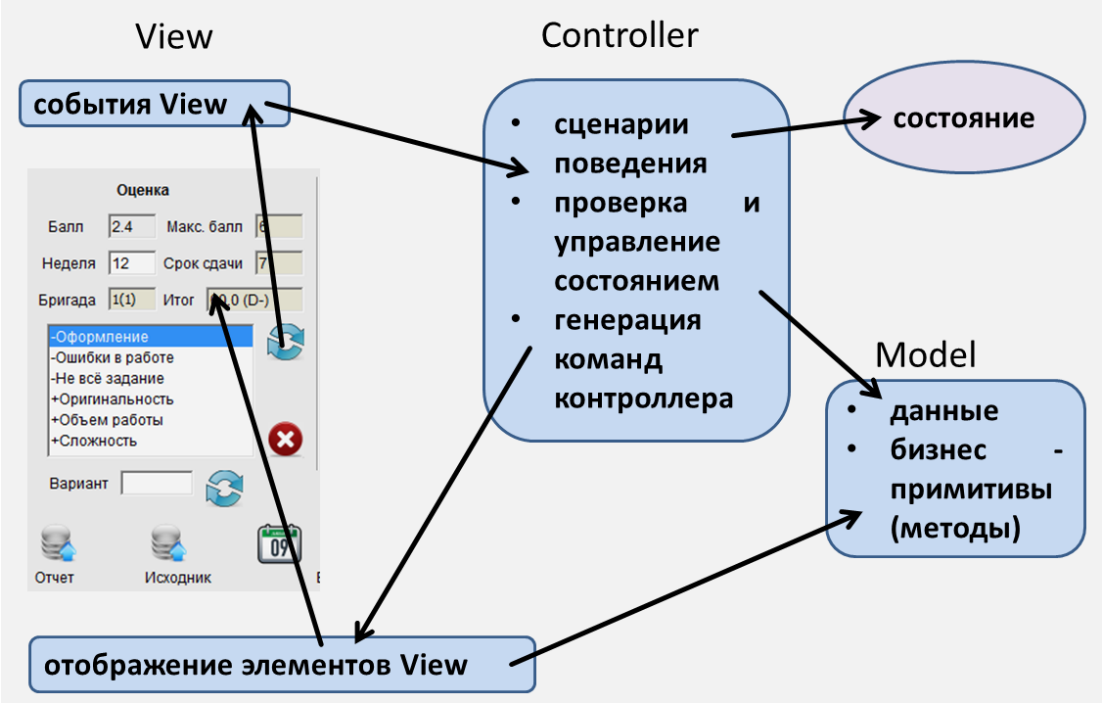

Рис.4-7. Архитектурный паттерн MVC

- под представлением понимается система основных экранных форм (диалогов), в рамках которых осуществляется бизнес-процесс;
- модель представляет собой набор необходимых бизнес-объектов, которые содержат в себе данные и методы, определяющие их краткосрочное, атомарное поведение. Таким образом, модель не определяет бизнес-процесс в целом, а только набор возможных действий в его рамках;
- контроллер управляет долгосрочным поведением представления в соответствии с бизнес-процессом. Для этого он отслеживает его текущее состояние на основе автоматной модели (рис.4-8).

Рис.4-8. Состояния контроллера при работе с бизнес-моделью

Технологически паттерн работает таким образом:

- главная форма создает контроллер в контексте приложения, при переходе от формы к форме передается контекст, а также интерфейс слушателя событий контроллера. Таким образом, контроллер передается по цепочке активных форм, с которыми он взаимодействует;
- контроллер имеет два встречных интерфейса: интерфейс событий, передаваемых от представления контролеру и интерфейс команд контроллера. Оба интерфейса функционально ориентированы на бизнес-логику и содержат набор событий и команд, необходимых для некоторого абстрактного представления. Их методы передают события и команды, которые могут сопровождать протекание бизнес-процесса, например, *установить видимость группы элементов управления, загрузить вектор имен в выпадающий список*

студентов и т.п.. Таким образом, контроллер может *управлять абстрактным тонким клиентом на уровне элементарных шагов бизнес-процесса*.

- контроллер создает модель, связывает ее с источником данных и в дальнейшем обрабатывает события модели, вызывает в ней методы, соответствующие шагам бизнес-процесса.

Помимо основных функций, контроллер в таком виде может выполнять много других общесистемных:

- кэширование данных - создание дополнительной бизнес-модели, настроенной на локальный источник данных, использование ее при отсутствии соединения с сервером, синхронизация локальных изменений. Все это в режиме, прозрачном для представления;
- обработка исключений, возникающих в модели и представлении, информирование о них представления через интерфейс событий контроллера в приемлемом для пользователя виде.

Архитектурный MVC обеспечивает максимальное повторное использование кода для приложений, работающих на разных платформах, либо в разных вариациях внешнего вида (см.7.2).