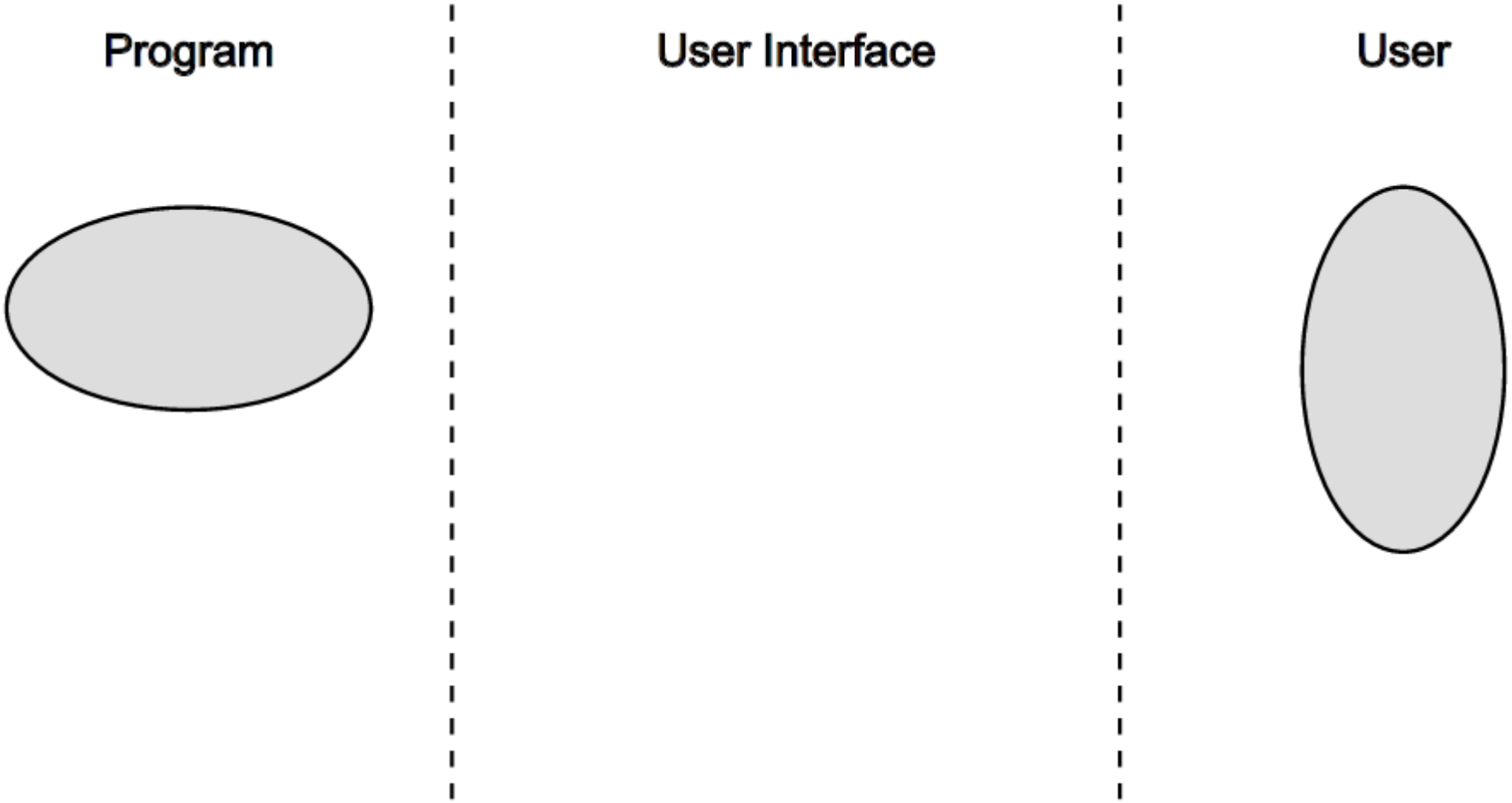
Что такое MVC?

Итак, MVC — это про пользовательский интерфейс (UI). Не обязательно графический, голосовое управление тоже годится. Не забудем, что программа может не иметь пользовательского интерфейса, может иметь программный интерфейс (API) или вообще никакого не иметь и всё ещё быть полезной.

Но если у нас есть пользователь, значит должен быть пользовательский интерфейс. Что же такое интерфейс? Это смежная граница между двумя системами. В нашем случае: с одной стороны — программа, с другой — пользователь. Вот они.



Программа совершенно абстрактная, любой предметный код. Она умеет делать что-то полезное, и у пользователя есть нужды, которые можно удовлетворить с помощью этой программы. Тогда появляются кусочки логики, которые «знают», как, используя эту программу, сделать непосредственно то, что хочет пользователь. Кусочки — не предметные, предметная логика в программе. Они больше относятся к пользователю с его конкретными потребностями, и представляют собой комбинации вызовов и обращений к программе.

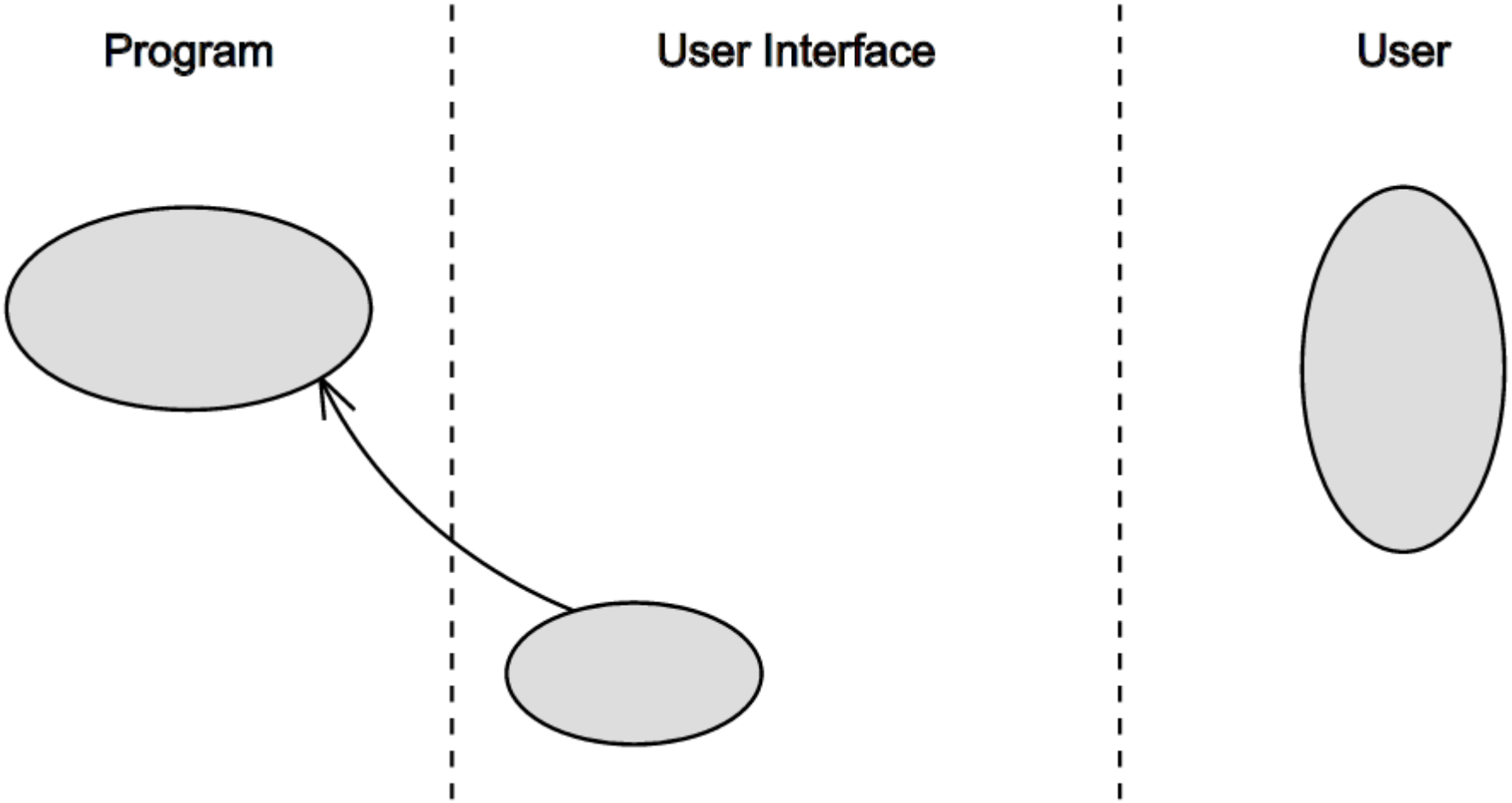
Юзкейсы

В качестве примера представьте терминал для торговли на бирже. Пользователь терминала выставляет заявку, в которой указывает, что он хочет купить акции компании «Светлый путь» в количестве 20 штук по цене 1500 рублей за акцию. Также указывает, что заявка действительна в течение четырёх часов, и с какого из его счетов списать деньги, в случае успешной сделки.

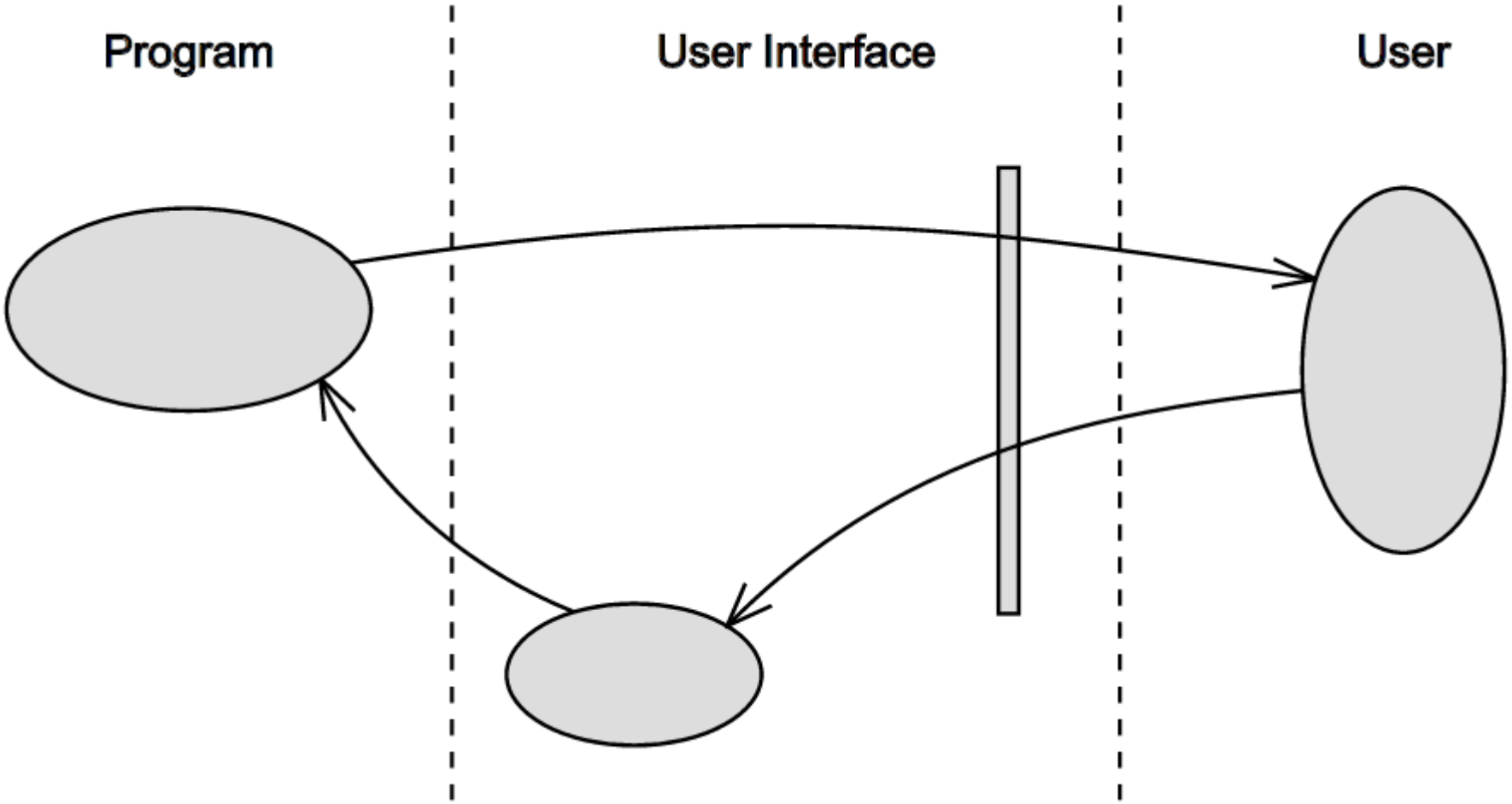
Ощутимое количество атрибутов. Проходит некоторое время, и он понимает, что по такой цене купить не удастся и готов поднять цену до 1550 рублей, оставив все остальные значения. Тогда он выбирает эту заявку, нажимает кнопку «изменить», указывает новую цену, да. Это удобно.

Но на бирже нельзя изменить заявку, в предметной области нет такого понятия. Заявку можно только выставить и отменить. Чтобы дать пользователю возможность в один клик менять заявку, надо запоминать старые значения, снимать заявку, давать редактировать то, что запомнили, и выставлять новую заявку. Такая комбинация. Но для пользователя она выглядит как одно простое действие: изменение заявки. Это называется — use case.

Дополним нашу диаграмму местом под юзкейсы.



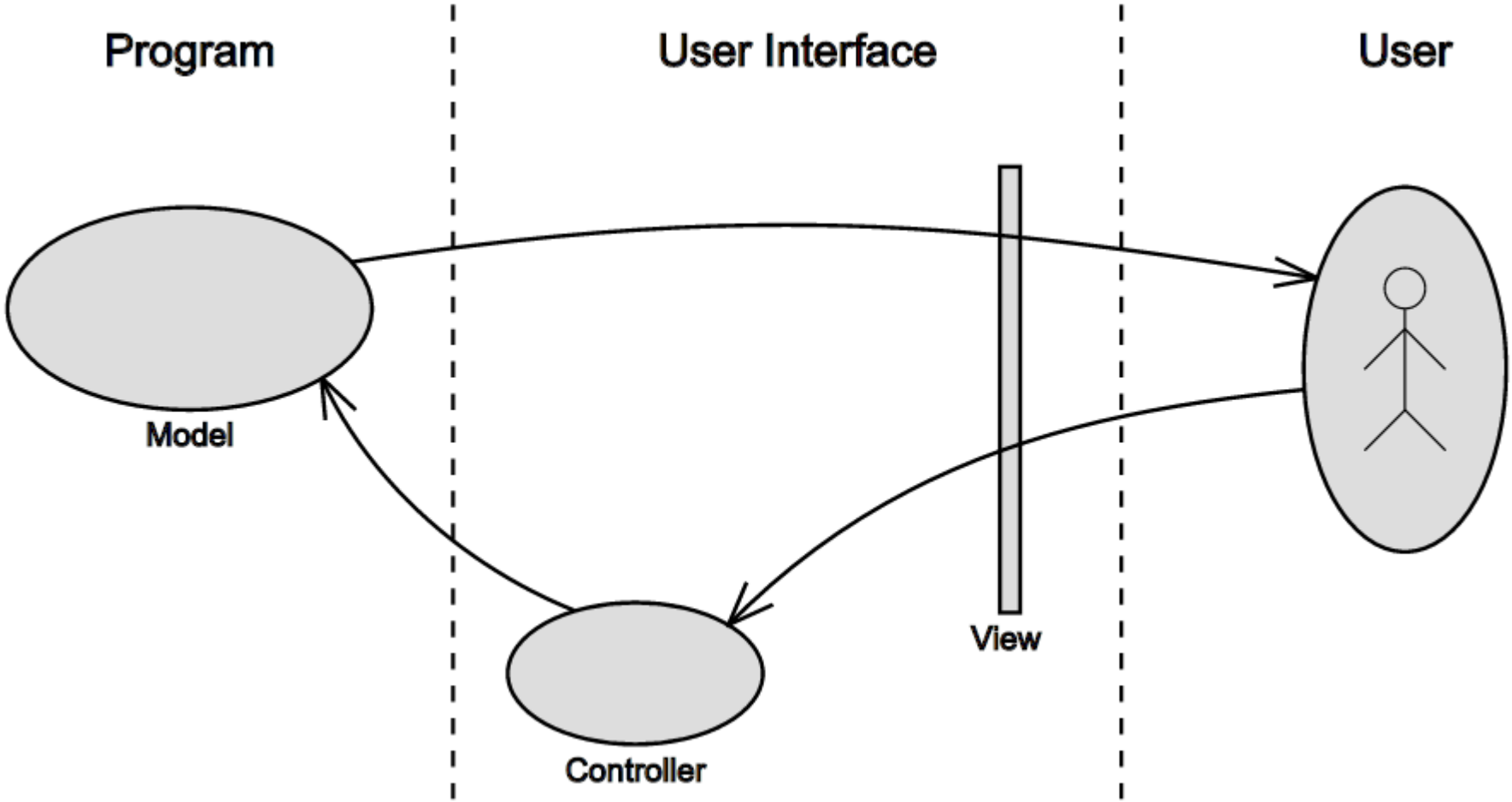
Ещё пользователю надо дать возможность дёргать эти юзкейсы и получать результат. Это могут быть кнопки и другие графические элементы ввода-вывода, жесты, распознавание и синтез речи. Любой вариант обмена данными и командами. Вуаля:



Пользователь дёргает какой-то из юзкейсов, который, в свою очередь, производит манипуляции над программой. Программа публикует результат или изменения в её состоянии.

Так где же тут все-таки MVC?

Все, что осталось — это только раздать знакомые имена образовавшимся компонентам.



Когда модель публикует изменения, её не волнует для кого, она ничего не знает про View. Вместо или вместе со View на том конце может быть другая подсистема.

Теперь немного частностей.

Это был классический вариант MVC — Active Model. Бывает и так, что модель не оповещает об изменениях. Тогда эту обязанность берёт на себя контроллер. Он знает, какие манипуляции производит над моделью, и, очевидно, знает, какие изменения в состоянии модели могут последовать. Это Passive Model.

И ещё один момент. Деление кода на предметный и не предметный — условное и зависит от того, насколько педантично мы хотим смоделировать предметную область. Иногда это рациональное решение — включить какой-то юзкейс в модель. Возможно, это уменьшит количество кода в целом и упростит его.