4. Построение модели анализа

Рассмотрим построение модели анализа на примере разработки ПО для Интернет-магазина.

4.1. Постановка задачи

Производитель компьютеров предлагает возможность приобретения своей продукции через Internet-магазин. Клиент может выбрать компьютер на Web-странице производителя. Компьютеры подразделится на серверы, настольные и портативные. Заказчик может выбрать стандартную конфигурацию или посмотреть требуемую конфигурацию в диалоговом режиме. Компоненты для построения конфигурации предоставляются как список для выбора из доступных альтернатив. Для каждой новой конфигурации система подсчитывает цену.

Чтобы оформить кредит клиент должен заполнить информацию по доставке и оплате. В качестве платежных средств допускается использование банковских карточек или наличный расчет. После ввода заказа система отправляет клиенту по электронной почте сообщение с подтверждением получения заказа вместе с относящимися к нему деталями. Пока клиент ожидает прибытия компьютера, он может проверить состояние заказа в любое время в диалоговом режиме.

Серверная часть обработки заказа состоит из заданий, необходимых для проверки кредитоспособности и способа расчета клиента за покупку, получения заказанной конфигурации со склада, печати счета и подачи заявки на склад о доставке компьютера клиенту.

4.2.Функциональные требования:

1. Для знакомства со стандартной конфигурацией выбираемого компьютера (сервера, настольного, портативного), клиент заходит на Web-страницу Internet-магазина. При знакомстве приводится цена конфигураций.
2. Клиент выбирает детали конфигурации, с которыми он хочет познакомиться, возможно, с намерением купить готовую или составить более подходящую конфигурацию. Цена конфигурации может быть подсчитана по требованию пользователя.
3. Клиент может выбрать вариант заказа по Internet или попросить, чтобы продавец связался с ним для уточнения деталей заказа, прежде, чем заказ будет фактически размещен.
4. Для размещения заказа клиент должен заполнить электронную форму с адресами для доставки товара и отправки счет-фактуры, а также деталями, касающимися оплаты (оплата по карточке или наличный расчет).
5. После ввода заказа в систему продавец отправляет на склад электронное требование, содержащее детали заказанной конфигурации.
6. Детали сделки (номер заказа, номер счета клиента) отправляются по электронной почте клиенту , так что заказчик может проверить состояние заказа через Internet.
7. Склад получает счет-фактуру от продавца и отгружает компьютер клиенту.

4.2.1. Прецеденты

Прецедент (use case) представляет собой некий целостный набор функций, имеющих определенную ценность для субъекта. Прецеденты можно вывести в результате идентификации задач для субъекта. Для этого следует задаться вопросом: “Каковы **обязанности субъекта по отношению к системе** и чего **он ожидает** от системы?”

Прецеденты также можно определить в результате непосредственного анализа функциональных требований. Во многих случаях *функциональное требование* отображается непосредственно в *прецедент*:

1. Показать стандартную конфигурацию компьютера.
2. Построить свою конфигурацию компьютера
3. Заказать конфигурацию компьютера
4. Связаться с продавцом.
5. Проверка способа оплаты.
6. Передача заказа на склад.
7. Обновление статуса заказа.
8. Печать счета-фактуры и отправка заказа.

Можно построить таблицу, которая распределяет функциональные требования по субъектам и прецедентам (табл. 4.1).

Таблица 4.1. Распределение требований по субъектам и прецедентам

№	Требование	Субъект	Прецедент
1	Для знакомства со стандартной конфигурацией выбираемого компьютера (сервера, настольного, портативного), клиент заходит на Web-страницу Internet-магазина. При знакомстве приводится цена конфигураций	Клиент	Отображение стандартной конфигурации компьютера
2	Клиент выбирает детали конфигурации, с которыми он хочет познакомиться, возможно, с намерением купить готовую или составить более подходящую конфигурацию. Цена конфигурации может быть подсчитана по требованию пользователя.	Клиент	Составление конфигурации
3	Клиент может выбрать вариант заказа по Internet или попросить, чтобы продавец связался с ним для уточнения деталей заказа, прежде, чем заказ будет фактически размещен.	Клиент, Продавец	Заказ выбранной конфигурации, Связаться с продавцом
4	Для размещения заказа клиент должен заполнить электронную форму с адресами для доставки товара и отправки счет-фактуры, а также деталями, касающимися оплаты (оплата по карточке или наличный расчет).	Клиент	Проверка и прием платежа
5	После ввода заказа в систему продавец отправляет на склад электронное требование, содержащее детали заказанной конфигурации.	Продавец, Склад	Передача заказа на склад
6	Детали сделки (номер заказа, номер счета клиента) отправляются по электронной почте клиенту , так что заказчик может проверить состояние заказа	Клиент, Продавец	Обновление статуса заказа

	через Internet.		
7	Склад получает счет-фактуру от продавца и отгружает компьютер клиенту.	Продавец, Склад	Печать счета-фактуры

Диаграмма прецедентов (рис. 4.1.) приписывает прецеденты к субъектам. Она также позволяет пользователю установить отношения между прецедентами, конечно, если такие отношения существуют. Чтобы представить полную модель прецедентов необходимо более подробное описание элементов диаграммы (прецедентов и субъектов). Смысл отношения <<extend>> (расширяет) состоит в том, что прецедент Заказ выбранной конфигурации может быть расширен субъектом Customer с помощью прецедента Связаться с продавцом.

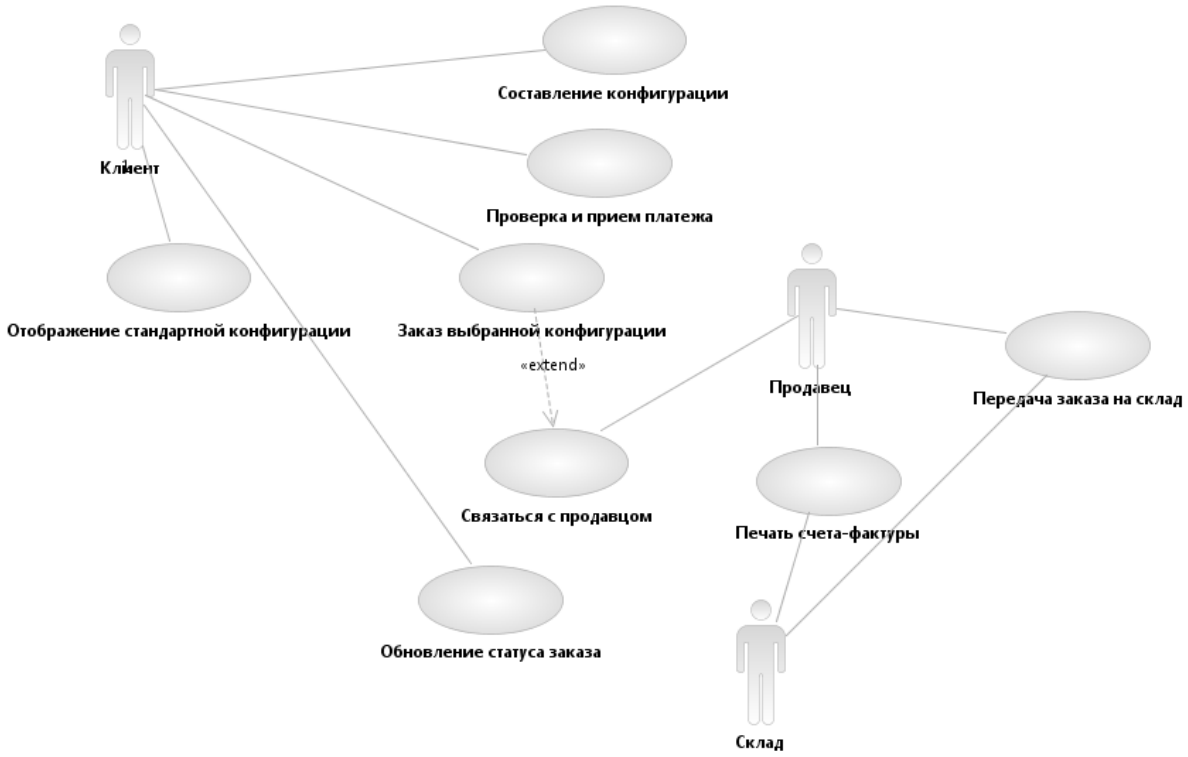

Рисунок 4.1. Диаграмма прецедентов для Интернет-магазина

4.2.2. Документирование прецедентов

Каждый прецедент должен быть описан с помощью документально зафиксированного потока событий (flow of events). Соответствующий текстовый документ определяет, что должна делать система, когда субъект инициирует прецедент. Структура документа, описывающего прецеденты, может варьироваться, однако типичное описание должно содержать следующие разделы:

- *Краткое описание.*
- Участвующие *субъекты.*
- *Предусловия*, необходимые для инициирования прецедента.
- *Детализированное описание* потока событий, которое включает:
  - *основной поток*, который можно разбить для того, чтобы показать *подчиненные потоки* событий (подчиненные потоки могут быть разделены дальше на еще более мелкие потоки, с целью улучшить удобочитаемость документа);
  - *альтернативные потоки* для определения исключительных ситуаций.
- *Постусловия*, определяющие состояние системы, по достижении которого прецедент завершается.

Документ, содержащий описания прецедента, развивается по ходу разработки. На ранней стадии определения требований составляется только краткое описание. Остальные части документа создаются постепенно и итеративно. Полный документ возникает в конце этапа спецификации требований. На этой стадии документ может быть дополнен прототипами GUI_экранов. Позднее документ по прецедентам используется для создания пользовательской документации для реализуемой системы.

Описательная спецификация прецедента Заказ сконфигурированного компьютера (Интернет-магазин) приведена в таблице 4.2.

Таблица 4.2. Прецедент Заказ сконфигурированного компьютера

Краткое описание	• Прецедент дает возможность Клиенту ввести заказ на покупку. Заказ включает адреса доставки товара и оплаты счета, а также детали условий оплаты
Актеры	• Клиент
Предусловия	• Клиент с помощью Internet-браузера выбирает страницу производителя компьютеров для ввода заказа. На Web-странице отображается подробная информация о сконфигурированном компьютере вместе с его ценой
Основной поток	• Начало прецедента совпадает с решением клиента заказать сконфигурированный компьютер с помощью выбора функции Продолжить (или аналогичной функции) при отображении на экране детализированной информации, относящейся к заказу. • Система просит клиента ввести детали покупки, в том числе: имя продавца (если оно известно); детали, касающиеся доставки (имя и адрес клиента); детальную информацию по оплате (если она отличается от информации по доставке); способ оплаты (карточка или наличные) и произвольные комментарии. • Клиент выбирает функцию Покупка (или аналогичную функцию) для отправки заказа производителю. • Система присваивает уникальный номер заказа и клиентский учетный номер заказу на покупку и запоминает информацию о заказе в базе данных. • Система отправляет клиенту по электронной почте номер заказа и • клиентский номер клиенту вместе со всеми деталями, относящимися к заказу, в качестве подтверждения принятия заказа.
Альтернативные потоки	• Клиент инициирует функцию Покупка до того, как введет всю обязательную информацию. Система отображает на экране сообщение об ошибке и просит ввести пропущенную информацию. • Клиент выбирает функцию Сброс (или аналогичную) для того, чтобы вернуться к исходной форме заказа на покупку. Система дает возможность клиенту вновь ввести информацию
Постусловия	• Если прецедент был успешным, заказ на покупку записывается в базу данных. В противном случае состояние системы остается неизменным

4.3. Моделирование видов деятельности

Модель видов деятельности (activity model) может представлять в графической форме поток событий для прецедента. Этот тип модели был введен только в более поздние версии UML и позволил преодолеть разрыв между высокоуровневым представлением поведения системы с помощью моделей прецедентов и намного более

низким уровнем представления поведения с помощью моделей взаимодействий (диаграмм последовательностей и диаграмм кооперации). Каждый шаг соответствует состоянию (state), в котором что-либо выполняется. Поэтому шаги выполнения называются состояниями вида деятельности. Диаграмма описывает, какие шаги выполняются последовательно, а какие — параллельно. Передача управления от одного состояния вида деятельности к другому называется переходом (transition).

Модель видов деятельности можно построить по описанию основного и альтернативных потоков в спецификации прецедентов. Однако, между описанием прецедентов и моделью видов деятельности существует важное различие. Описание **прецедента с**оздается с точки зрения **внешнего** субъекта. **Модель видов деятельности** отражает **внутрисистемную** точку зрения.

4.3.1. Виды деятельности

Состояние вида деятельности можно установить на основе документа описания прецедента. Имена действиям следует присваивать, исходя из системных соображений, а не с точки зрения субъекта.

Состояние вида деятельности представляется в UML в виде прямоугольника с закругленными углами. Следует сразу уточнить, что один и тот же графический символ используется для визуализации состояния вида деятельности (activity state) и состояния действия (action state). Различие между деятельностью и действием заключается в их временном масштабе. Для осуществления деятельности требуется определенное время; действие же завершается столь быстро, что может считаться происходящим мгновенно.

Следовательно, в модели состояний могут быть определены только в рамках состояния объекта, а действия могут появляться также при переходе между состояниями объекта.

4.3.2. Состояния действия и состояния деятельности

В потоке управления, моделируемом диаграммой деятельности, происходят различные события. Могут вычисляться выражения, в результате чего изменяется значение некоторого атрибута или возвращается некоторое значение. Также, например, можно выполнить операцию над объектом, послать ему сигнал, создать или уничтожить объект. Все эти атомарные вычисления называются состояниями действия. Состояния действия атомарные и не могут быть подвергнуты декомпозиции.

В противоположность этому состояние деятельности могут быть подвергнуты дальнейшей декомпозиции, вследствие чего выполняемую деятельность можно представить с помощью других диаграмм деятельности.

Можно считать, что состояние действия - это частный случай состояние деятельности, то есть такое состояние, которое не может быть подвергнуто дальнейшей декомпозиции. А состояние деятельности можно представить как составное состояние, поток управления которого включает другие состояния деятельности и действий.

Таблица 4.3. Установление действий в основном и альтернативных потоках

№	***Формулировка прецедента***	***Состояние вида деятельности***
1	Начало прецедента совпадает с решением клиента заказать сконфигурированный компьютер с помощью выбора функции Продолжить (или аналогичной функции) при отображении на экране детализированной информации, относящейся к заказу	Отображение текущей конфигурации, Получение запроса на заказ
2	Система просит клиента ввести детализированную информацию о покупке, в том числе: имя продавца (если оно известно); детали, касающиеся доставки (имя и адрес клиента);	Отображение закупочной Формы

	детальную информацию по оплате (если она отличается от информации по доставке); способ оплаты (карточка или наличная) и произвольные комментарии	
3	Клиент выбирает функцию Покупка (или аналогичную функцию) для отправки заказа производителю	Детализировать информацию о покупке
4	Система присваивает уникальный номер заказа и клиентский учетный номер заказу на покупку и запоминает информацию о заказе в базе данных	Запомнить заказ
5	Система отправляет клиенту по электронной почте номер заказа и клиентский номер клиенту вместе со всеми деталями, относящимися к заказу, в качестве подтверждения принятия заказа	Отправить детальную информацию по заказу
6	Клиент инициирует функцию Покупка до того, как введет всю обязательную информацию. Система отображает на экране сообщение об ошибке и просит ввести пропущенную информацию	Детализировать информацию о покупке, Отображение закупочной Формы
7	Клиент выбирает функцию Сброс (или аналогичную) для того, чтобы вернуться к исходной форме заказа на покупку. Система дает возможность клиенту вновь ввести информацию	Отображение закупочной формы

4.3.3. Диаграмма видов деятельности

Диаграмма видов деятельности (activity diagram) показывает переходы между видами деятельности. Если вид деятельности не представляет собой замкнутого цикла, то диаграмма содержит начальное состояние вида деятельности и одно или более конечных состояний вида деятельности.

Переходы могут разветвляться по условию и объединяться. В результате возникают альтернативные (alternative) вычислительные потоки (thread). Условие ветвления обозначается ромбом. Переходы могут также разделяться и сливаться. В результате возникают параллельные (одновременно выполняемые) (concurrent) потоки. Распараллеливание и воссоединение переходов представляется в виде жирной линии или полосы. Заметим, что диаграмма вида деятельности, в которой отсутствуют параллельные процессы, похожа на обычную блок_схему.

Начальное состояние деятельности — Отображение текущей конфигурации. Рекурсивный переход на этом состоянии служит признанием того факта, что отображение непрерывно обновляется до тех пор, пока не сработает следующий переход (переход в состояние **Получение запроса на заказ**). Этот факт может интерпретироваться как осознание того, что это состояние является деятельностью, а не действием.

Если при нахождении модели в состоянии **Получение закупочной формы** сработает условие [время вышло], то выполнение модели видов деятельности завершится. Иначе, активизируется состояние **Детализировать информацию о покупке**. Если детальные данные относительно покупки неполны, система вновь переходит в состояние **Отображение закупочной формы.** В противном случае система переходит в состояние **Запомнить заказ**, а затем — в состояние **Отправить детальную информацию по заказу** (конечное состояние).

Обратите внимание, что на диаграмме показаны только те условные переходы, которые появляются на выходах из состояния вида деятельности. Условные переходы, которые являются внутренними для состояния вида деятельности, не показаны явно на диаграмме. Об их существовании можно догадаться по наличию нескольких исходящих

переходов, которые, возможно, сопровождаются именем условия в квадратных скобках (например, таким как [время вышло] на выходе из состояния Отображение закупочной формы).

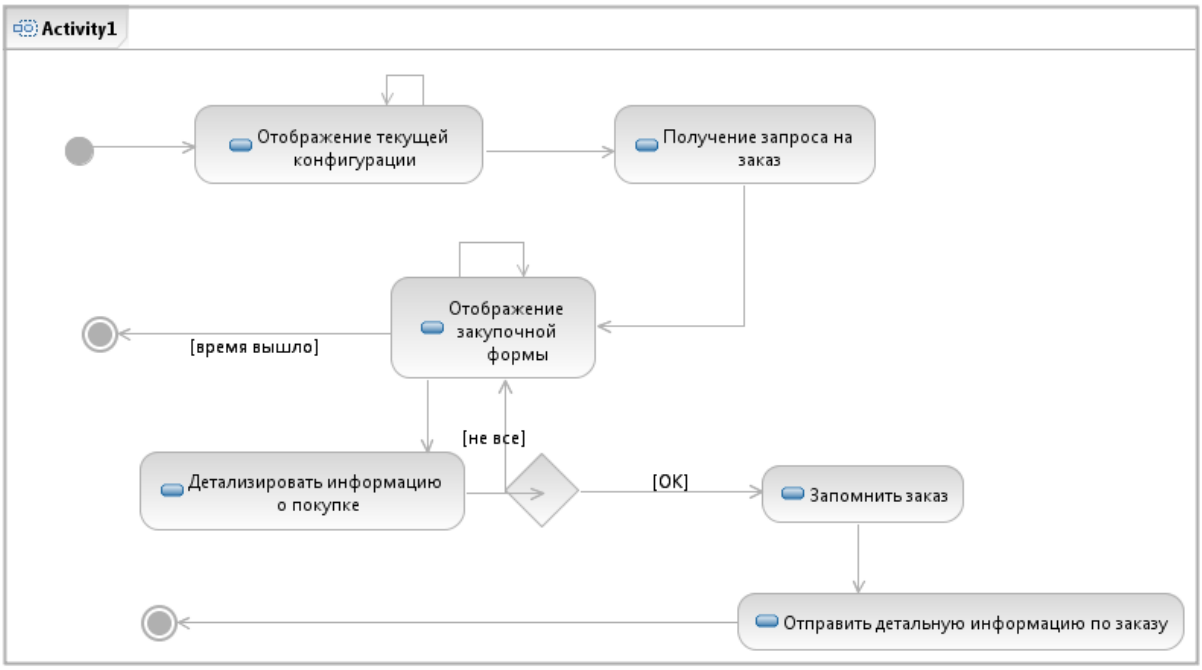

Рисунок 4.2. Диаграмма видов деятельности для прецедента Заказ сконфигурированного компьютера

4.4. Моделирование классов

Классы-сущности определяют существо любой информационной системы. Анализ требований направлен преимущественно на выявление классов-сущностей. Однако, для функционирования системы требуются также классы другого типа. Пользователям системы необходимы классы, которые определяют объекты графического интерфеса (например, такие как экранные формы), называемые пограничными классами (boundary classes) Чтобы функционировать надлежащим образом, системе также необходимы классы, которые управляют программной логикой — управляющие классы (control classes)

В зависимости от конкретного подхода к моделированию пограничные и управляющие классы могут рассматриваться или не рассматриваться на некотором уровне представления в ходе анализа требований. Моделирование классов этого типа может быть отложено до этапа проектирования системы.

Можно построить таблицу, которая поможет выявить классы в результате анализа функциональных требований.

Таблица 4.4. Соответствие функциональных требований и классов-сущностей

№	*Требование*	*Класс-сущность*
1	Для знакомства со стандартной конфигурацией выбираемого сервера, настольного или портативного компьютера клиент использует Web-страницу Internet-магазина. При этом также приводится цена конфигурации	• Customer (Клиент); • Computer (Компьютер); • StandardConfiguration (Стандартная конфигурация); • Product (Товар)
2	Клиент выбирает детали конфигурации, с которыми хочет познакомиться, возможно, с намерением купить готовую или составить более	• Customer, • ConfiguredComputer (Сконфигурированный

	подходящую конфигурацию. Цена для каждой конфигурации может быть подсчитана по требованию пользователя	компьютер); • ConfiguredProduct (Укомплектованный товар); • ConfigurationItem (Элемент конфигурации)
3	Клиент может выбрать вариант заказа компьютера по Internet либо попросить, чтобы продавец связался с ним для объяснения деталей заказа, договорился о цене и т.п. прежде, чем заказ будет фактически размещен	• Customer, ConfiguredComputer, • Order (Заказ), • Salesperson (Продавец)
4	Для размещения заказа клиент должен заполнить электронную форму с адресами для доставки товара и отправки счета-фактуры, а также деталями, касающимися оплаты (кредитная карточка или чек)	• Customer; • Order; • Shipment(Поставка); • Invoice (Счет_фактура); • Payment (Платеж)
5	После ввода заказа клиента в систему продавец отправляет на склад электронное требование, содержащее детали, касающиеся заказанной конфигурации	• Customer; • Order; • Salesperson; • ConfiguredComputer; • ConfigurationItem
6	Детали сделки, включая номер заказа, номер счета клиента, отправляются по электронной почте клиенту, так что заказчик может проверить состояние заказа через Internet	• Order; • Customer; • OrderStatus • (Состояние заказа)
7	Склад получает счет-фактуру от продавца и отправляет компьютер клиенту	• Invoice; • Shipment

Выделение классов представляет собой итеративную задачу, и первоначальный перечень предполагаемых классов, как правило, претерпевает изменения. При определении того, являются ли понятия, присутствующие в требованиях, искомыми классами, могут помочь ответы на некоторые вопросы. Вот эти вопросы.

1. Является ли понятие “вместилищем” данных?
2. Обладает ли оно отдельными атрибутами, способными принимать разные значения?
3. Можно ли создать для него множество объектов-экземпляров?
4. Входит ли оно в границы прикладной области?

При анализе таблицы возникают следующие вопросы:

1. В чем различие между классами ConfiguredComputer (Сконфигурированный компьютер) и Order (ЗаказБудем ли мы хранить информацию о сконфигурированном компьютере (ConfiguredComputer), до того как заказ (объект Order) размещен или нет?
2. Совпадает ли смысл понятия Shipment (Поставка), приведенный в требованиях №4 и №7?
3. Если нет, тонужен ли нам класс Shipment, если нам известно, что поставка является обязанностью склада и, таким образом, выходит за рамки приложения?
4. Не является ли понятие элемента конфигурации (ConfigurationItem) просто атрибутом понятия сконфигурированного компьютера (ConfiguredComputer)?
5. Является понятие состояние заказа (OrderStatus) самостоятельным классом или это атрибут понятия заказа (Order)?
6. Является ли понятие продавца (Salesperson) самостоятельным классом или это атрибут понятий Order и Invoice?

Ответив на эти вопросы (дать ответы на эти и аналогичные вопросы не легко, для этого требуется глубокое знание требований прикладной области), можно получить следующие классы:

- Customer (Клиент) – с точки зрения прецедента (Клиент появился в качестве субъекта на диаграмме прецедента).
- Computer (Компьютер).
- ConfiguredComputer (Сконфигурированный компьютер);
- ConfigurationItem (Элемент конфигурации).
- Order (Заказ).
- Payment (Платеж).
- Invoice (Счет_фактура).

При дальнейшем анализе предметной области определяем атрибуты классов.

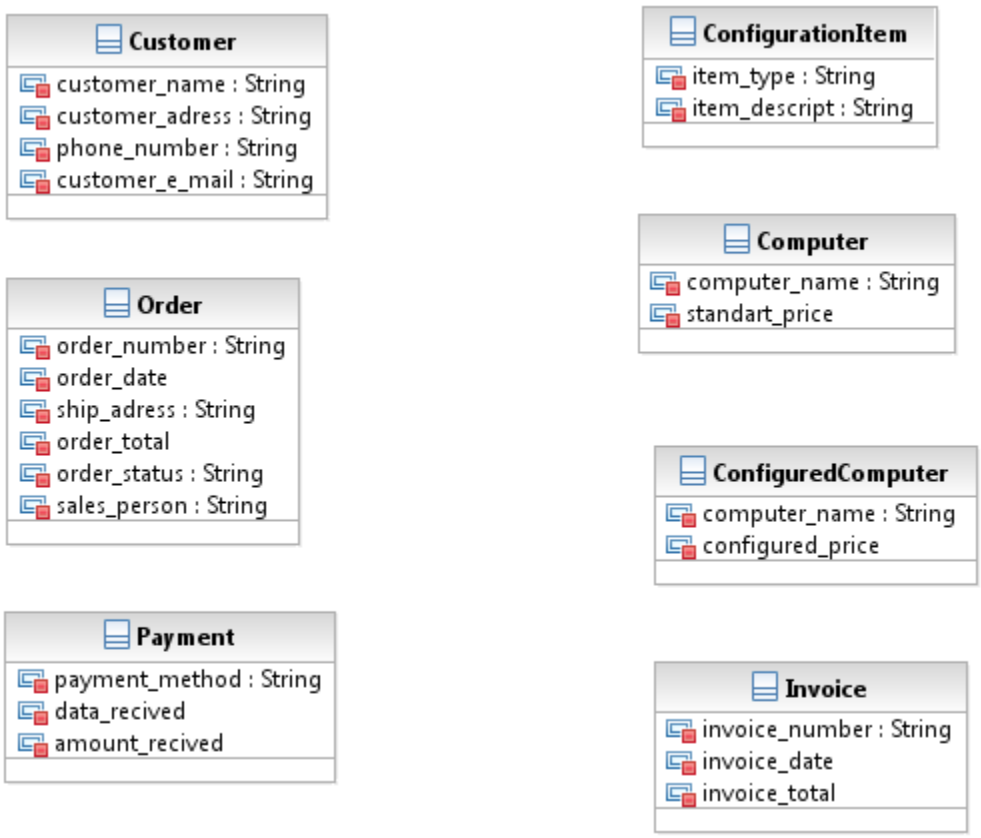

Рисунок 4.3. Классы и элементарные атрибуты (Интернет-магазин)

Рассмотрим связи, которые можно установить между классами.

Ассоциации, связывающие классы, устанавливают пути, облегчающие взаимодействие объектов. В модели анализа ассоциации представлены соединительными линиями. На рис. 4.4. показаны наиболее очевидные ассоциации между классами. При определении кратности ассоциаций был сделан ряд предположений. Заказ (Order) поступает от одного клиента (Customer), однако клиент может разместить несколько заказов. Заказ не принимается до тех пор, пока не определены реквизиты платежа (Payment) (отсюда, ассоциация типа “один к одному”). Заказ не должен обладать связанной с ним счетом-фактурой (Invoice), однако счет-фактура всегда связана с единственным заказом. Заказ делается на один или несколько сконфигурированных компьютеров (ConfiguredComputer). А компьютер с данной конфигурацией может заказываться многократно или не заказываться вовсе.

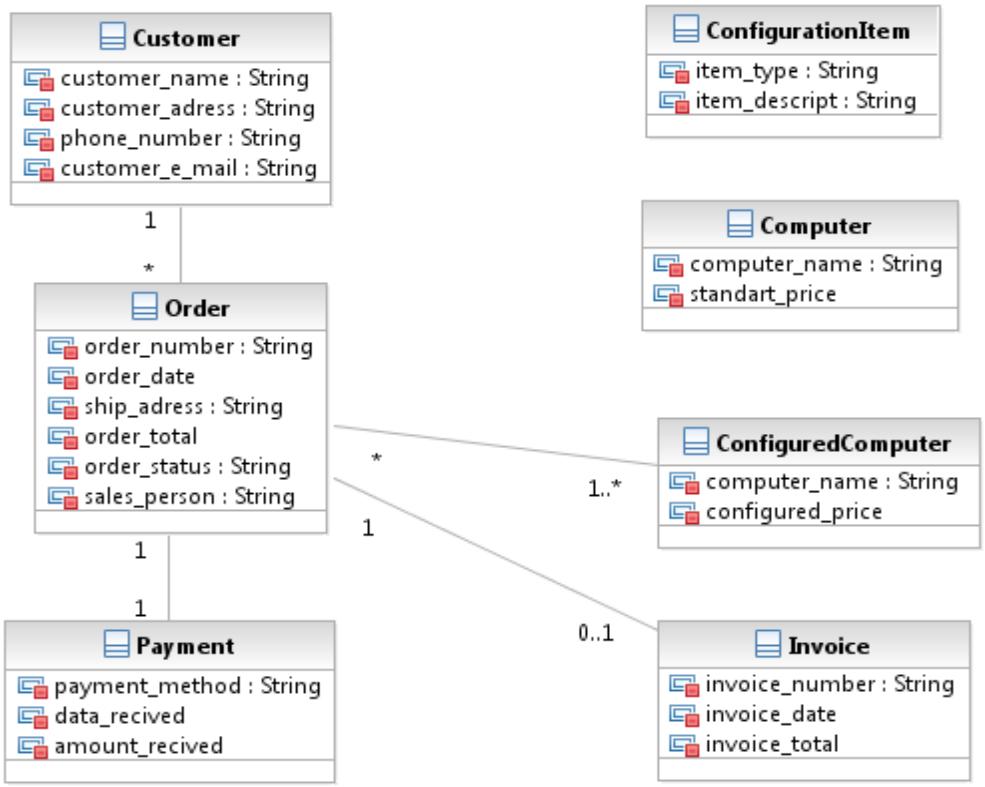

Рисунок 4.4. Ассоциации (Интернет-магазин)

Агрегация и композиция являются более сильной формой ассоциативной связи, которой присуща семантика принадлежности. На рис. 4.5. к модели добавлены отношения агрегации. Компьютер (Computer) обладает одним или более элементами конфигурации (ConfigurationItems). Подобно этому, сконфигурированный компьютер (ConfiguredComputer) состоит из одного или нескольких элементов (ConfigurationItems).

Обобщение представляет собой средство повторного использования ПО, кроме того, оно также способно в значительной мере упростить и прояснить модель. Обобщение обычно используют как метод создания дополнительных базовых классов (как правило, абстрактных). Упрощение достигается за счет увеличения точности, с которой существующий класс может быть ассоциирован с наиболее подходящими (т.е. на наиболее удобном уровне абстракции) классами в иерархии обобщения.

На рис. 4.6. показана модифицированная модель, в которой класс изменен на абстрактный класс Computer, являющийся базовым для двух производных классов - StandardComputer и ConfiguredComputer. Теперь классы Order и ConfigurationItem связаны с классом Computer, который может быть либо классом StandardComputer, либо классом ConfiguredComputer.

В результате мы получили **диаграмму классов**, которая составляет основу объектно-ориентированной системы. В классах пока не определены операции. Операции принадлежат больше к сфере проектирования, а не анализа. После того, как операции в конечном итоге вводятся в классы, модель классов в явном виде определяет поведение системы.

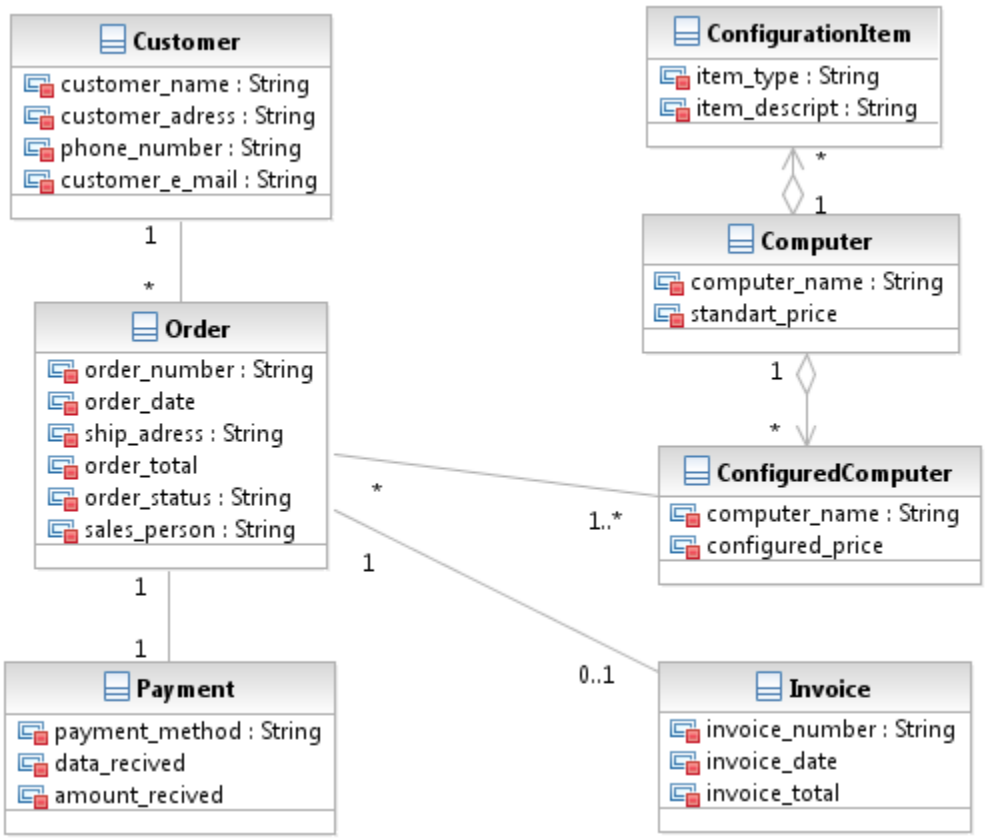

Рисунок 4.5. Агрегации (Интернет-магазин)

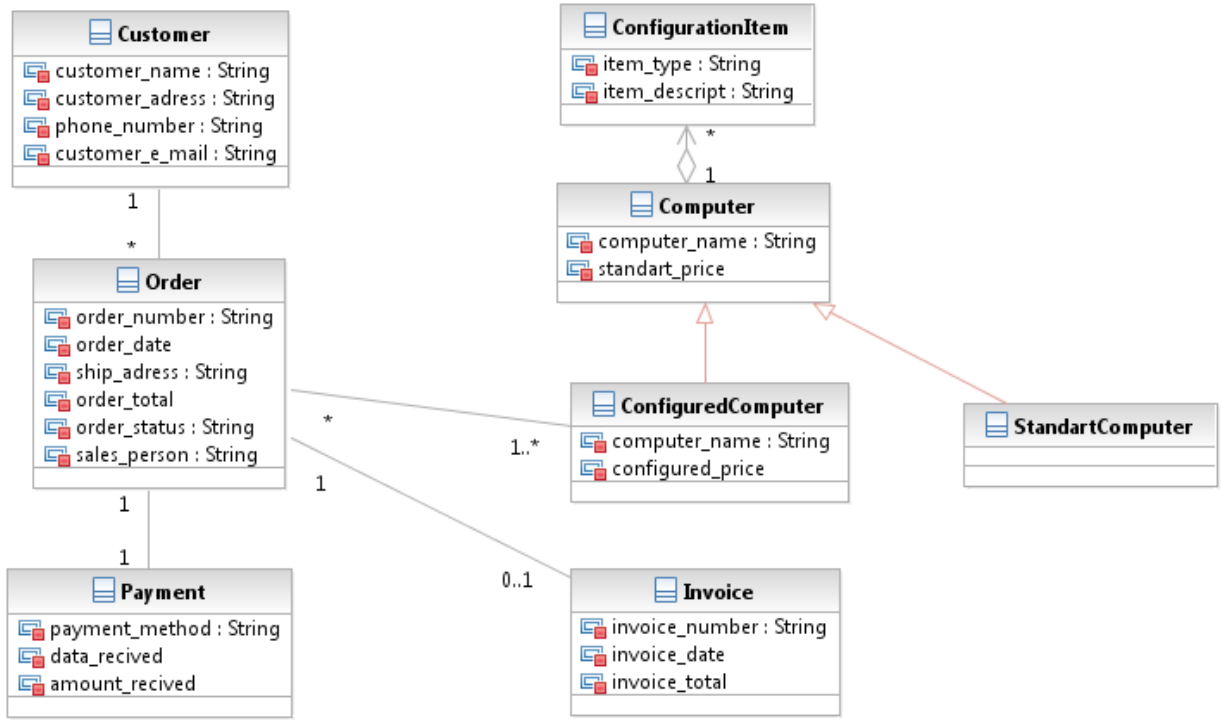

Рисунок 4.6. Диаграмма классов (Интернет-магазин)

4.5. Моделирование взаимодействий

Моделирование взаимодействий рассматривает вопросы взаимодействия между объектами, необходимыми для выполнения прецедента. Модели взаимодействия

используются на более развитых стадиях анализа требований, когда становится известной модель классов, так что ссылки на объекты опираются на модель классов.

Диаграммы взаимодействий разделяются на два вида - **диаграммы последовательностей и диаграммы кооперации**. Они могут использоваться как взаимозаменяемые, и многие CASE-средства поддерживают автоматическое преобразование одной модели в другую. Разница между моделями заключается в акцентах. Модели последовательностей концентрируются на временных последовательностях событий, а в моделях кооперации основное внимание уделяется отношениям между объектами.

Диаграммы последовательностей, как правило, используются на этапе анализа требований, а диаграммы кооперации — системного проектирования. Этот выбор соответствует общепринятой практике разработки ИС.

Взаимодействие представляет собой набор сообщений, свойственных поведению некоторой системы, которыми обмениваются объекты в соответствии с установленными между ними связями. Диаграмма последовательностей представляется двумерным графом. Объекты располагаются по горизонтали. Последовательности сообщений располагаются сверху вниз по вертикали. Каждая вертикальная линия называется линией жизни (lifeline) объекта.

Стрелки представляют каждое сообщение, направляемое от вызывающего объекта (отправителя) к операции (методу) вызываемого объекта (получателя). Для каждого сообщения, как минимум, указывается его имя. Кроме того, для сообщения могут быть указаны фактические аргументы сообщения и другая управляющая информация. Фактические аргументы соответствуют формальным аргументам метода объекта получателя.

Фактические аргументы могут быть входными аргументами (передаются от отправителя к получателю) или выходными аргументами (передаются от получателя назад к отправителю). Входные аргументы могут быть обозначены ключевым словом in (если ключевое слово отсутствует, то предполагается, что аргумент — входной). Выходные аргументы обозначаются ключевым словом out. Допускаются также аргументы типа inout (“входные/выходные”), однако, для объектно-ориентированного подхода они не характерны.

Показывать на диаграмме возврат управления от объекта-получателя объекту-отправителю не обязательно. Стрелка, указывающая на объект-получатель, предполагает автоматический возврат управления отправителю. Получатель знает уникальный идентификатор объекта (Object Identifier, OID) отправителя. Сообщение может быть отправлено коллекции объектов (коллекция может быть набором, списком, массивом объектов и т.д.). Довольно частой является ситуация, когда вызывающий объект связан с несколькими объектами-получателями (поскольку кратность ассоциации указана как “один ко многим” или “многие ко многим”). Итеративный маркер - звездочка перед обозначением сообщения - указывает на процесс итерации сообщения по всей коллекции.

Диаграмма последовательностей для “отображения текущей конфигурации” показана на рис. 4.7. Внешний субъект - клиент (Customer) принимает решение об отображении конфигурации компьютера. Сообщение openNew (открыть новое [окно]) отправляется объекту confWin класса ConfWin ([диалоговое] окно конфигурации). В результате создается новый объект confWin. Класс ConfigurationWindow - пограничный класс.

Объекту ConfWin необходимо “отобразить себя” вместе с данными, относящимися к конфигурации. С этой целью он отправляет сообщение объекту aComp:Computer. В действительности, aComp — это объект класса StandardComputer или ConfiguredComputer. Класс Computer — абстрактный класс. Объект aComp использует выходной аргумент для того, чтобы “собрать себя” из элементов конфигурации — объектов ConfigurationItem. Затем он “оптом” отсылает элементы конфигурации объекту aConfWin в качестве

аргумента i_recset сообщения displayComputer. Теперь объект aConfWin может отобразить себя на экране.

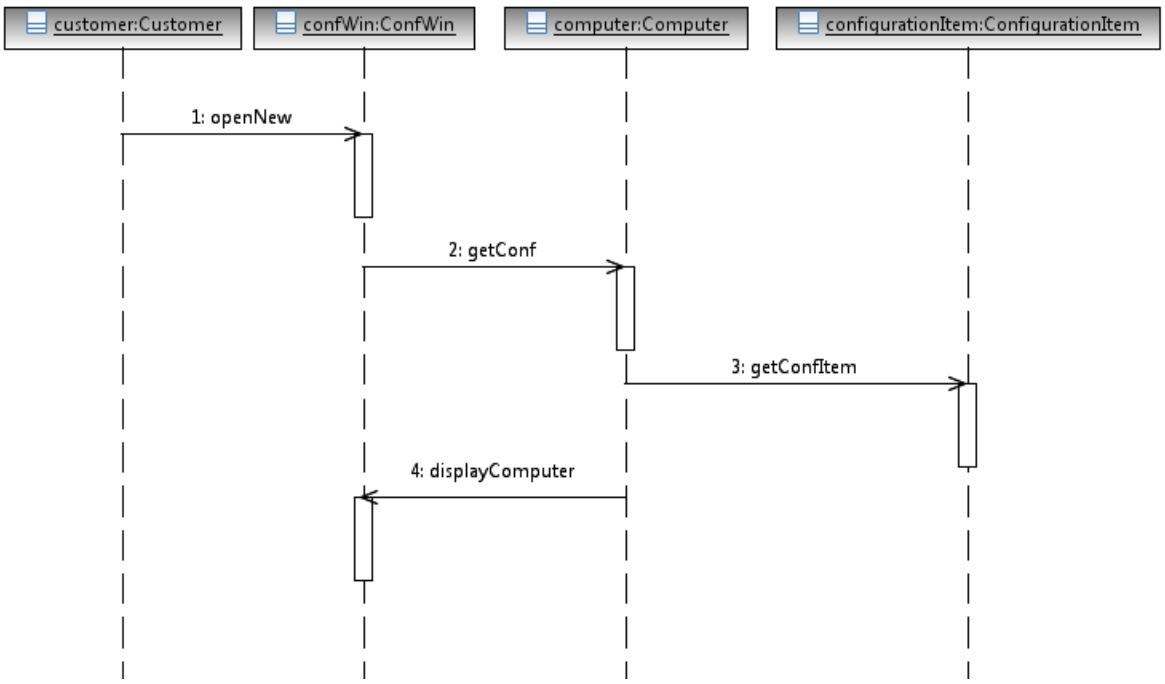

Рисунок 4.7. Диаграмма последовательностей для вида деятельности Отображение текущей конфигурации.

Хотя введение операций в классы часто откладывается до этапа проектирования, исследование взаимодействий между классами может привести к выявлению **операций**. Между взаимодействиями и операциями существует прямая зависимость. Каждое сообщение обращается к операции в вызываемом объекте. Имена операции и сообщения совпадают. Для каждого сообщения на диаграмме последовательностей нужно ввести операцию на диаграмме классов.

Изменениям вносятся в три класса. Класс ConfigurationWindow — это пограничный класс. Два других класса — это классы-сущности, представляющие постоянные объекты базы данных. Операция openNew выбрана в качестве шаблона операции конструктора. Это означает, что операция openNew будет реализована в качестве метода конструктора, который порождает новые объекты класса. (В RSA – операции добавляются в классы автоматически).

Для каждого прецедента, как правило, строится отдельная диаграмма последовательностей. Т.к. каждый прецедент должен быть выражен через диаграмму видов деятельности, диаграмма последовательностей строится для каждой диаграммы видов деятельности. Использование нескольких взаимоувязанных точек зрения на одну и ту же систему является краеугольным камнем моделирования.

Диаграмма последовательностей во многом является самоочевидной. Сообщение acceptConf вызывает отправку сообщения prepareForOrder объекту :Order. Это приводит к созданию временного объекта :Order, отображаемого в “объекте-окне” :OrderWindow.

В ответ на принятие клиентом детализированных условий заказа (submitOrder) объект :OrderWindow инициирует (storeOrder) создание постоянного объекта :Order. После этого объект-заказ :Order устанавливает связь с заказанным компьютером (:Computer), а также соответствующими клиентом (:Customer) и платежом :Payment. После того, как эти объекты постоянно связаны в базе данных, объект :Order отправляет электронное сообщение emailOrder клиенту (Customer).

Обратите внимание на двойное использование сущности :Customer - в качестве внешнего субъекта, представленного объектом, и внутреннего объекта-класса. Подобная двойственность довольно часто встречается в моделировании. Клиент (Customer) является по отношению к системе одновременно и внешней, и внутренней сущностью. Он представляет собой внешнюю сущность, поскольку взаимодействует с системой извне. Однако информация о клиенте должна храниться в системе, чтобы иметь возможность установить идентичность внешнего клиента и внутренней сущности, о которой знает система.

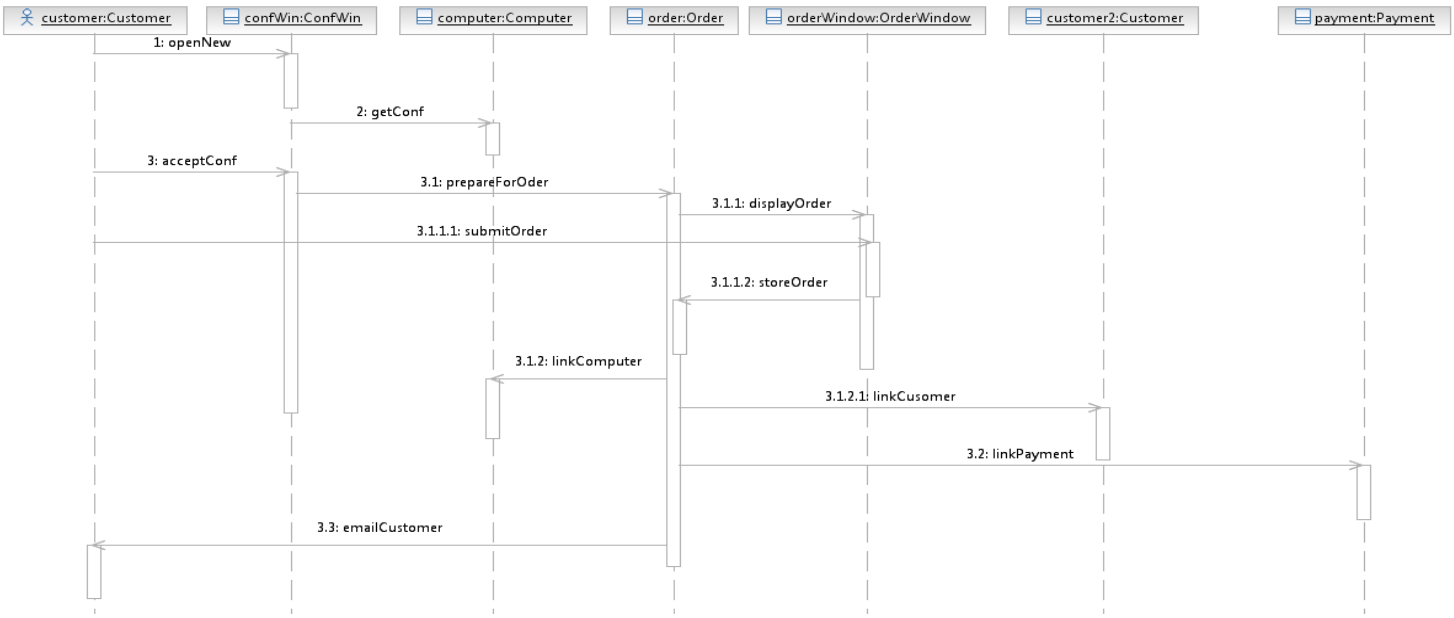

Рисунок 4.8. Диаграмма последовательности для прецедента Заказ выбранной конфигурации (Интернет-магазин)

4.6. Моделирование состояний

Модель взаимодействий позволяет составить детализированную спецификацию прецедента. Модель состояний (statechart model) служит детализированным описанием класса или, более точно, динамических изменений состояний класса. Эти динамические изменения обычно описывают поведение объекта в рамках нескольких прецедентов.

Модель состояний (statechart model) фиксирует возможные состояния, в которых может находиться класс, и эффективно фиксирует “жизненный путь” класса. На протяжении своего жизненного цикла объект остается одним и тем же - его идентичность никогда не изменяется. Однако состояние объекта изменяется.

Диаграмма состояний представляет собой граф состояний (прямоугольников с закругленными углами) и переходов (стрелки), вызванных событиями.

Рассмотрим класс Invoice (Счет-фактура), относящийся к приложению Интернет-магазин. Из модели прецедентов известно, что клиент определяет способ оплаты (карточка или наличные), когда закупочная форма заполнена и отослана производителю. Это приводит к генерации заказа и последующей подготовке счета-фактуры. Но диаграмма прецедентов не проясняет вопрос о том, когда производится фактическая оплата в соответствии со счетом фактурой. Можно предположить, что оплата может осуществляться до или после того как счет-фактура выдана, также можно предположить возможность частичной оплаты.

Из модели класса нам известно, что счет-фактура подготавливается продавцом и , в конечном итоге, передается на склад. Склад отправляет счет-фактуру клиенту одновременно с доставкой клиенту компьютера. Состояние оплаты счета-фактуры отслеживается в системе и счет-фактура снабжена соответствующими комментариями.

Приведенный выше пример схватывает сущность моделирования состояний. Модели состояний строятся для классов, которые характеризуются не просто изменениями состояний, а изменениями состояний, представляющими определенный интерес с точки зрения предметной области. Решение о том, что представляет интерес, а что нет, является прерогативой моделирования бизнес-процессов. Диаграмма состояний представляет собой модель бизнес-правил. В течение некоторого времени бизнес-правила остаются неизменными. Они относительно независимы от конкретных прецедентов. В действительности прецеденты должны соответствовать бизнес-правилам.

На рис. 4.9. показана модель состояний для класса Invoice. Начальным состоянием объекта Invoice является состояние Unpaid (не оплачено). Из состояния Unpaid возможны два перехода. При наступлении события partial payment (частичная оплата) объект Invoice переходит в состояние Partly Paid (частично оплачено). Допускается только одна частичная оплата. Наступление события finalpayment (окончательная оплата), когда объект Invoice находится в состоянии Unpaid или Partly Paid, инициирует переход в состояние Fully Paid (полностью оплачено). Это конечное состояние.

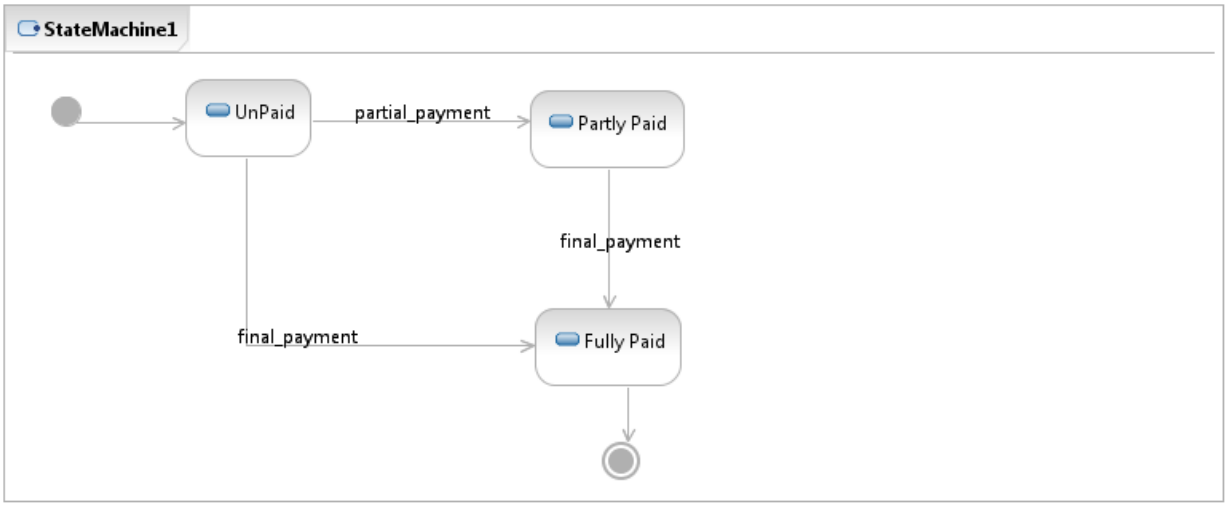

Рисунок 4.9. Состояния и события для класса Invoice (Интернет-магазин)

Диаграмма состояний обычно присоединяется к классу, но, в общем случае, она может присоединяться к другим представлениям, например, прецедентам. Диаграмма состояний, присоединенная к классу, определяет способ реагирования объектов класса на события. Более точно, диаграмма определяет - для каждого состояния объекта - действие (action), выполняемое объектом при получении им сигнала о событии. Один и тот же объект может выполнять различные действия в ответ на одно и то же событие в зависимости от состояния объекта. Выполнение действия, как правило, вызывает изменение состояния.

Полное описание перехода (transition) состоит из трех частей:

```
event (parameters) [guard] / action.
```

Каждая из частей является необязательной. Если строка перехода самоочевидна, допустимо опустить все компоненты**. Событие** — это кратковременное явление, которое воздействует на объект. Оно может обладать **параметрами**, например, `кнопка мыши нажата (правая-кнопка)`. Событие может быть защищено **ограждающим условием** (guard), например, `кнопка мыши нажата (правая-кнопка) [внутри окна]`. Если условие принимает значение “истина”, событие срабатывает и воздействует на объект. **Действие** (action) представляет собой короткое атомическое вычисление, которое выполняется при срабатывании перехода. Действие также может быть связано с состоянием.

В общем случае действие - это отклик объекта на обнаруженное событие. Состояние может содержать более продолжительное вычисление - вид деятельности (activity).

Состояние может состоять из других состояний, которые называются вложенными (nested state). Составное состояние (composite state) является абстрактным - это всего лишь родовое имя для всех вложенных состояний. Переход, который берет начало из-за границы составного состояния, означает, что он может сработать от любого из вложенных состояний. Это делает диаграмму более ясной и лаконичной. Конечно, переход, который берет начало вне границы составного состояния, может сработать от вложенного состояния.

Диаграмма состояний для класса Order показана на рис. 4.10. Начальное состояние объекта класса - New Order (новый заказ). Это одно из вложенных состояний составного состояния Pending (ожидающий [заказ]) - к другим относятся состояния Back Order (невыполненный заказ) и Future Order (будущий заказ). Из каждого из этих трех состояний, вложенных в состояние Pending, возможны два перехода. Переход в состояние Canceled (отменен) защищено условием [canceled]. Должна существовать возможность - без изменения правил моделирования состояний - заменить защитное условие событием cancel. Переход в состояние Ready to Ship (готов к доставке) помечен полным описанием, содержащим событие, защиту и действие.

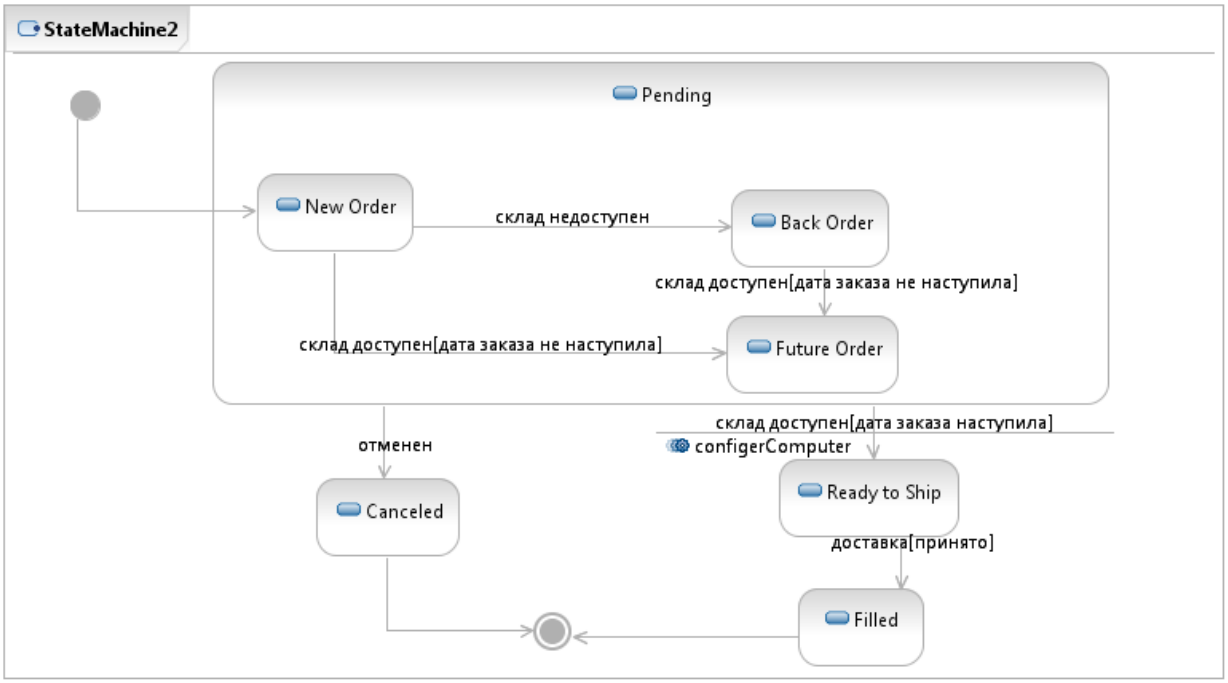

Рисунок 4.10. Диаграмма состояний для класса Order (Интернет-магазин)