2.2. Построение объектной модели

Теперь у нас есть все необходимые понятия, чтобы описать процесс построения объектной модели. Этот процесс включает в себя следующие этапы:

- определение объектов и классов;
- подготовка словаря данных;
- определение зависимостей между объектами;
- определение атрибутов объектов и связей;
- организация и упрощение классов при использовании
- наследования;
- дальнейшее исследование и усовершенствование модели.

2.2.1. Определение классов

Анализ внешних требований к проектируемой прикладной системе позволяет определить объекты и классы объектов, связанные с прикладной проблемой, которую должна решать эта система. Все классы должны быть осмыслены в рассматриваемой прикладной области; классов, связанных с компьютерной реализацией, как например список, стэк и т.п. на этом этапе вводить не следует.

Начать нужно с выделения возможных классов из письменной постановки прикладной задачи (технического задания и другой документации, предоставленной заказчиком). Следует иметь в виду, что это очень сложный и ответственный этап разработки, так как от него во многом зависит дальнейшая судьба проекта.

При определении возможных классов нужно постараться выделить как можно больше классов, выписывая имя каждого класса, который приходит на ум. В частности, каждому существительному, встречающемуся в предварительной постановке задачи, может соответствовать класс. Поэтому при выделении возможных классов каждому такому существительному обычно сопоставляется возможный класс.

Далее список возможных классов должен быть проанализирован с целью исключения из него ненужных классов. Такими классами являются:

1. **избыточные классы:** если два или несколько классов выражают одинаковую информацию, следует сохранить только один из них;
2. **нерелевантные (не имеющие прямого отношения к проблеме) классы:** для каждого имени возможного класса оценивается, насколько он необходим в будущей системе (оценить это часто бывает весьма непросто); нерелевантные классы исключаются;
3. **нечетко определенные (с точки зрения рассматриваемой проблемы) классы** (см. примеры таких классов в п. 2.3.1);
4. **атрибуты:** некоторым существительным больше соответствуют не классы, а атрибуты; такие существительные, как правило, описывают свойства объектов (например, имя, возраст, вес, адрес и т.п.);
5. **операции:** некоторым существительным больше соответствуют не классы, а имена операций (например, телефонный_вызов вряд ли означает какой-либо класс);
6. **роли:** некоторые существительные определяют имена ролей в объектной модели (например, владелец, водитель, начальник, служащий; все эти имена связаны с ролями в различных зависимостях объектов класса человек);

7. **реализационные конструкции:** именам, больше связанным с программированием и компьютерной аппаратурой, не следует на данном этапе сопоставлять классов, так как они не отражают особенностей проектируемой прикладной системы; примеры таких имен: подпрограмма, процесс, алгоритм, прерывание и т.п.

После исключения имен всех ненужных (лишних) возможных классов будет получен предварительный список классов, составляющих проектируемую систему.

2.2.2. Подготовка словаря данных

Отдельные слова имеют слишком много интерпретаций. Поэтому необходимо в самом начале проектирования подготовить словарь данных, содержащий четкие и недвусмысленные определения всех объектов (классов), атрибутов, операций, ролей и других сущностей, рассматриваемых в проекте. Без такого словаря обсуждение проекта с коллегами по разработке и заказчиками системы не имеет смысла, так как каждый может по-своему интерпретировать обсуждаемые термины. Пример такого словаря см. в п. 2.3.2.

2.2.3. Определение зависимостей

На следующем этапе построения объектной модели определяются зависимости между классами. Прежде всего из классов исключаются атрибуты, являющиеся явными ссылками на другие классы; такие атрибуты заменяются зависимостями. Смысл такой замены в том, что зависимости представляют собой абстракцию того же уровня, что и классы, и потому не оказывают непосредственного влияния на будущую реализацию (ссылка на класс лишь один из способов реализации зависимостей).

Аналогично тому, как имена возможных классов получались из существительных, встречающихся в предварительной постановке прикладной задачи, имена возможных зависимостей могут быть получены из глаголов или глагольных оборотов, встречающихся в указанном документе. Так обычно описываются: физическое положение (следует_за, является_частью, содержится_в), направленное действие (приводит_в_движение), общение (разговаривает_с), принадлежность (имеет, является_частью) и т.п. Пример выделения явных и неявных глагольных оборотов из предварительной постановки конкретной прикладной задачи рассмотрен в п. 2.3.3.

Затем следует убрать ненужные или неправильные зависимости, используя следующие критерии:

- зависимости между исключенными классами должны быть исключены, либо переформулированы в терминах оставшихся классов (см. пример в п. 2.3.3);
- нерелевантные зависимости и зависимости, связанные с реализацией, должны быть исключены (см. пример в п. 2.3.3);
- действия: зависимость должна описывать структурные свойства прикладной области, а не малосущественные события (см. примеры в п. 2.3.3);
- тренарные зависимости: большую часть зависимостей между тремя или большим числом классов можно разложить на несколько бинарных зависимостей, используя в случае необходимости квалификаторы (см. примеры в п. 2.3.3); в некоторых (очень редких) случаях такое разложение осуществить не удается; например, тренарная зависимость "Профессор читает курс в аудитории 628" не может быть разложена на бинарные без потери информации;
- производные зависимости: нужно исключать зависимости, которые можно выразить через другие зависимости, так как они избыточны (см. пример в п. 2.3.3); при

исключении избыточных (производных) зависимостей нужно быть особенно осторожным, так как не все дублирующие одна другую зависимости между классами избыточны; в некоторых случаях другие зависимости позволяют установить только существование еще одной производной зависимости, но не позволяют установить кратность этой зависимости; например, в случае, представленном на рисунке 2.36, фирма имеет много служащих и владеет многими компьютерами; каждому служащему предоставлено для персонального использования несколько компьютеров, кроме того, имеются компьютеры общего пользования; кратность зависимости предоставлен_для_использования не может быть выведена из зависимостей служит и владеет; хотя производные зависимости и не добавляют новой информации, они часто бывают удобны; в этих случаях их можно указывать на диаграмме, пометив косой чертой.

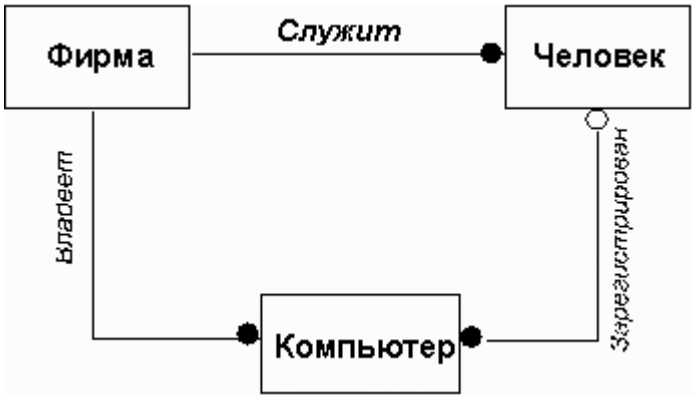

Рис. 2.36. Неизбыточные зависимости

Удалив избыточные зависимости, нужно уточнить семантику оставшихся зависимостей следующим образом:

- **неверно названные зависимости:** их следует переименовать, чтобы смысл их стал понятен (см. пример в п. 2.3.3);
- **имена ролей:** нужно добавить имена ролей там, где это необходимо; имя роли описывает роль, которую играет соответствующий класс в данной зависимости с точки зрения другого класса, участвующего в этой зависимости; если имя роли ясно из имени класса, его можно не указывать (см. пример в п. 2.3.3);
- **квалификаторы:** добавляя квалификаторы там, где это необходимо, мы вносим элементы контекста, что позволяет добиться однозначной идентификации объектов; квалификаторы позволяют также упростить некоторые зависимости, понизив их кратность;
- **кратность:** необходимо добавить обозначения кратности зависимостей; при этом следует помнить, что кратность зависимостей может меняться в процессе дальнейшего анализа требований к системе;
- **неучтенные зависимости** должны быть выявлены и добавлены в модель.

2.2.4. Уточнение атрибутов

На следующем этапе уточняется система атрибутов: корректируются атрибуты классов, вводятся, в случае необходимости, новые атрибуты. Атрибуты выражают свойства объектов рассматриваемого класса, либо определяют их текущее состояние.

Атрибуты обычно соответствуют существительным; например **цвет_автомобиля** (свойство объекта), **позиция_курсора** (состояние объекта). Атрибуты, как правило, слабо влияют на структуру объектной модели.

Не следует стремиться определить как можно больше атрибутов: большое количество атрибутов усложняет модель, затрудняет понимание проблемы. Необходимо вводить только те атрибуты, которые имеют отношение к проектируемой прикладной системе, опуская случайные, малосущественные и производные атрибуты.

Наряду с атрибутами объектов необходимо ввести и атрибуты зависимостей между классами (связей между объектами).

При уточнении атрибутов руководствуются следующими критериями:

- **Замена атрибутов на объекты.** Если наличие некоторой сущности важнее, чем ее значение, то это объект, если важнее значение, то это атрибут: например, начальник - это объект (неважно, кто именно начальник, главное, чтобы кто-то им был), зарплата - это атрибут (ее значение весьма существенно); город - всегда объект, хотя в некоторых случаях может показаться, что это атрибут (например, город как часть адреса фирмы); в тех случаях, когда нужно, чтобы город был атрибутом, следует определить зависимость (скажем, находится) между классами фирма и город.
- **Квалификаторы.** Если значение атрибута зависит от конкретного контекста, его следует сделать квалификатором (см. примеры в п. 2.3.4).
- **Имена.** Именам обычно лучше соответствуют квалификаторы, чем атрибуты объектов; во всех случаях, когда имя позволяет сделать выбор из объектов некоторого множества, его следует сделать квалификатором (см. примеры в п. 2.3.4).
- **Идентификаторы.** Идентификаторы объектов связаны с их реализацией. На ранних стадиях проектирования их не следует рассматривать в качестве атрибутов.
- **Атрибуты связей.** Если некоторое свойство характеризует не объект сам по себе, а его связь с другим объектом (объектами), то это атрибут связи, а не атрибут объекта.
- **Внутренние значения.** Атрибуты, определяющие лишь внутреннее состояние объекта, незаметное вне объекта, следует исключить из рассмотрения.
- **Несущественные детали.** Атрибуты, не влияющие на выполнение большей части операций, рекомендуется опустить.

2.2.5. Организация системы классов, используя наследование

Далее необходимо постараться найти суперклассы для введенных классов. Это полезно, так как проясняет структуру модели и облегчает последующую реализацию. Соответствующий пример рассмотрен в п. 2.3.5.

2.2.6. Дальнейшее исследование и усовершенствование модели

Лишь в очень редких случаях построенная объектная модель сразу же оказывается корректной. Модель должна быть исследована и отлажена. Некоторые ошибки могут быть найдены при исследовании модели без компьютера, другие - при ее интерпретации совместно с динамической и функциональной моделями на компьютере (эти модели строятся после того, как объектная модель уже построена).

Здесь мы рассмотрим приемы бескомпьютерного поиска и исправления ошибок в объектной модели. В их основе лежат внешние признаки, по которым можно находить ошибки в модели; эти признаки могут быть объединены в следующие группы.

Признаки пропущенного объекта (класса):

- несимметричности связей и обобщений (наследований); для исправления ошибки необходимо добавить пропущенные классы;
- несоответствие атрибутов и операций у класса; для исправления ошибки необходимо расщепить класс на несколько других классов, так чтобы атрибуты и операции новых классов соответствовали друг другу;
- обнаружена операция, не имеющая удовлетворительного целевого класса; для исправления ошибки необходимо добавить пропущенный целевой класс;
- обнаружено несколько зависимостей с одинаковыми именами и назначением; для исправления ошибки необходимо сделать обобщение и добавить пропущенный суперкласс.

Признаки ненужного (лишнего) класса:

- нехватка атрибутов, операций и зависимостей у некоторого класса; для исправления ошибки необходимо подумать, не следует ли исключить такой класс.

Признаки пропущенных зависимостей:

- отсутствуют пути доступа к операциям; для исправления ошибки необходимо добавить новые зависимости, обеспечивающие возможности обслуживания соответствующих запросов.

Признаки ненужных (лишних) зависимостей:

- избыточная информация в зависимостях; для исправления ошибки необходимо исключить зависимости, не добавляющие новой информации, или пометить их как производные зависимости;
- не хватает операций, пересекающих зависимость; для исправления ошибки необходимо подумать, не следует ли исключить такую зависимость.

Признаки неправильного размещения зависимостей:

- имена ролей слишком широки или слишком узки для их классов; для исправления ошибки необходимо переместить зависимость вверх или вниз по иерархии классов.

Признаки неправильного размещения атрибутов:

- нет необходимости доступа к объекту по значениям одного из его атрибутов; для исправления ошибки необходимо рассмотреть нужно ли ввести квалифицированную зависимость.

Примеры практического применения описанных признаков см. в п. 2.3.6.

2.3. Пример объектной модели

Рассмотрим процесс построения объектной модели для системы банковского обслуживания (см. п. 1.3) в процессе анализа требований и предварительного проектирования этой системы. Для построения объектной модели рассматриваемой системы нам необходимо выполнить все этапы, перечисленные в п 2.2.

2.3.1. Определение объектов и классов

В пункте 1.3 сформулирована задача и приведена схема сети банковского обслуживания (рисунок 1.3). Анализируя эту постановку задачи, можно выделить возможные классы, сопоставив их существительным, упомянутым в ее предварительной формулировке; получится следующий список возможных имен классов (в алфавитном порядке):

АТМ (банкомат) кассир программное обеспечение
банк кассовый терминал система
банковская сеть квитанция проверка безопасности
данные проводки клиент служба ведения записей
данные счета компьютер банка счет
деньги консорциум цена
доступ пользователь центральный компьютер
карточка проводка

Исследуем этот список, исключая из него имена классов в соответствии с рекомендациями п. 2.2.1:

избыточные классы: ясно, что клиент и пользователь означают одно и то же понятие; для банковской системы более естественно оставить класс клиент;
нерелевантные классы: таким классом является класс цена (он не имеет непосредственного отношения к работе банковской сети);
нечетко определенные классы: такими классами являются служба_ведения_записей и проверка безопасности (эти службы входят в состав проводки), система (в нашем случае непонятно, что это такое), банковская_сеть (вся наша программная система будет обслуживать банковскую сеть);
атрибуты: данные проводки, данные счета, деньги (имеются в виду реальные деньги, выдаваемые клиенту кассиром или банкоматом, либо принимаемые кассиром), квитанция (выдается клиенту вместе с деньгами) более естественно иметь в качестве атрибутов;
реализационные конструкции выражают такие имена как программное_обеспечение и доступ; их тоже следует исключить из списка имен возможных классов.

После исключения всех лишних имен возможных классов получаем следующий список классов, составляющих проектируемую систему банковского обслуживания (эти классы представлены на рисунке 2.5):

АТМ (банкомат) кассовый терминал проводка
банк клиент счет
карточка компьютер банка центральный компьютер
кассир консорциум

2.3.2. Подготовка словаря данных

Приведем часть словаря данных, содержащую определения классов, используемых в проекте.

АТМ (банкомат) - терминал, который дает возможность клиенту осуществлять свою собственную проводку, используя для идентификации свою карточку. АТМ (банкомат) взаимодействует с клиентом, чтобы получить необходимую информацию для проводки,

посылает информацию для проводки центральному_компьютеру, чтобы он проверил ее и в дальнейшем использовал при выполнении проводки и выдает деньги и квитанцию клиенту. Предполагается, что АТМ (банкомату) не требуется работать независимо от сети.

Банк - финансовая организация, которая содержит счета своих клиентов и выпускает карточки, санкционирующие доступ к счетам через сеть АТМ (банкоматов).

Карточка - пластиковая карточка, врученная банком своему клиенту, которая санкционирует доступ к счетам через сеть АТМ (банкоматов). Каждая карточка содержит код банка и номер карточки, закодированные в соответствии с национальными стандартами на банковские карточки. Код_банка однозначно идентифицирует банк внутри консорциума. Номер_карточки определяет счета, к которым карточка имеет доступ. Карточка не обязательно обеспечивает доступ ко всем счетам клиента. Каждой карточкой может владеть только один клиент, но у нее может существовать несколько копий, так что необходимо рассмотреть возможность одновременного использования одной и той же карточки с разных АТМ (банкоматов).

Кассир - служащий банка, который имеет право осуществлять проводки с кассовых_терминалов, а также принимать и выдавать деньги и чеки клиентам. Проводки, деньги и чеки, с которыми работает каждый кассир должны протоколироваться и правильно учитываться.

Кассовый_терминал - терминал, с которого кассир осуществляет проводки для клиентов. Когда кассир принимает и выдает деньги и чеки, кассовый_терминал печатает квитанции. Кассовый_терминал взаимодействует с компьютером_банка, чтобы проверить и выполнить проводку.

Клиент - держатель одного или нескольких счетов в банке. Клиент может состоять из одного или нескольких лиц, или организаций. То же самое лицо, держащее счет и в другом банке рассматривается как другой клиент.

Компьютер_банка - компьютер, принадлежащий банку, который взаимодействует с сетью АТМ (банкоматов) и собственными кассовыми_терминалами банка. Банк может иметь свою внутреннюю компьютерную сеть для обработки счетов, но здесь мы рассматриваем только тот компьютер_банка, который взаимодействует с сетью АТМ.

Консорциум - объединение банков, которое обеспечивает работу сети АТМ (банкоматов). Сеть передает в консорциум проводки банков.

Проводка - единичный интегрированный запрос на выполнение некоторой последовательности операций над счетами одного клиента. Было сделано предположение, что АТМ (банкоматы) только выдают деньги, однако для них не следует исключать возможности печати чеков или приема денег и чеков. Хотелось бы также обеспечить гибкость системы, которая в дальнейшем обеспечит возможность одновременной обработки счетов разных клиентов, хотя пока этого не требуется. Различные операции должны быть правильно сбалансированы.

Счет - единичный банковский счет, над которым выполняются проводки. Счета могут быть различных типов; клиент может иметь несколько счетов.

Центральный_компьютер - компьютер, принадлежащий консорциуму, который распределяет проводки и их результаты между АТМ (банкоматами) и компьютерами_банков. Центральный_компьютер проверяет коды банков, но не выполняет проводок самостоятельно.

2.3.3. Определение зависимостей

Следуя рекомендациям п. 2.2.3, выделяем явные и неявные глагольные обороты из предварительной постановки задачи и рассматриваем их как имена возможных зависимостей. Из постановки задачи о банковской сети (см. п. 1.3) можно извлечь следующие обороты:

Глагольные обороты (явные и неявные):
Банковская сеть *включает* кассиров и АТМ'ы
Консорциум *распределяет* результаты проводок по АТМ
Банк *владеет* компьютером банка
Компьютер банка *поддерживает* счета
Банк *владеет* кассовыми терминалами
Кассовый терминал *взаимодействует* с компьютером банка
Кассир *вводит* проводку над счетом
АТМ'ы *взаимодействуют* с центральным компьютером во время проводки
Центральный компьютер *взаимодействует* с компьютером банка
АТМ *принимает* карточку
АТМ *общается* с пользователем
АТМ *выдает* наличные деньги
АТМ *печатает* квитанции
Система *регулирует* коллективный доступ
Банк *предоставляет* программное обеспечение
Консорциум *состоит* из банков
Консорциум *владеет* центральным компьютером
Система *обеспечивает* протоколирование
Система *обеспечивает* безопасность
Клиенты *имеют* карточки
Карточка *обеспечивает* доступ к счету
В банке *служат* кассиры

Затем исключаем ненужные или неправильные зависимости, используя критерии, сформулированные в п. 2.2.3:

- **зависимости между исключенными классами:** исключаются следующие зависимости: Банковская сеть включает кассиров и АТМ'ы (класс банковская_сеть исключен), АТМ печатает квитанции (класс квитанция исключен), АТМ выдает наличные деньги (класс деньги исключен), Система обеспечивает протоколирование проводок (класс служба_ведения_записей исключен), Система обеспечивает безопасность ведения счетов (класс служба_безопасности исключен), Банки предоставляют программное обеспечение (класс программное_обеспечение исключен);
- **нерелевантные зависимости и зависимости, связанные с реализацией:**
- зависимость "Система регулирует коллективный доступ" исключается как связанная с реализацией; действия описываются такими зависимостями как "АТМ принимает карточку" и "АТМ общается с пользователем"; мы исключаем эти зависимости;

- **тренарные зависимости:** зависимость "Кассир вводит проводку над счетом" раскладывается на две бинарные зависимости "Кассир вводит проводку" и "Проводка относится к счету". Зависимость "АТМ'ы взаимодействуют с центральным компьютером во время проводки" раскладывается на "АТМ'ы взаимодействуют с центральным компьютером" и "Проводка начинается с АТМ";
- **производные зависимости:** зависимость "Консорциум распределяет АТМ'ы" является следствием зависимостей "Консорциум владеет центральным компьютером" и "АТМ'ы взаимодействуют с центральным компьютером".

Удалив избыточные зависимости, получим следующий список зависимостей:
Банк *владеет* компьютером банка
Компьютер банка *поддерживает* счета
Банк *владеет* кассовыми терминалами
Кассовый терминал *взаимодействует* с компьютером банка
Кассир *вводит* проводку
Проводка *относится* к счету
АТМ'ы *взаимодействуют* с центральным компьютером
Проводка *начинается* с АТМ
Центральный компьютер *взаимодействует* с компьютером банка
Консорциум *состоит* из банков
Консорциум *владеет* центральным компьютером
Клиенты *имеют* карточки
Карточка *обеспечивает доступ* к счету
В банке *служат* кассиры

Уточним семантику оставшихся зависимостей следующим образом:

переименуем неверно названные зависимости, чтобы смысл их стал более понятен; так зависимость Компьютер_банка *поддерживает* счета удобнее заменить зависимостью Банк *держит* счета.

имена ролей можно не использовать, так как они ясны из имен классов, участвующих в зависимости, как например, для зависимости АТМ'ы *взаимодействуют* с центральным компьютером;

неучтенные зависимости: Проводка *начинается* с кассового_терминала, Клиенты *имеют* счета, Проводка *регистрируется* карточкой следует добавить в модель.

После уточнения зависимостей можно составить исходную версию объектной диаграммы. Для рассматриваемой задачи она будет иметь вид, представленный на рисунке 2.37.

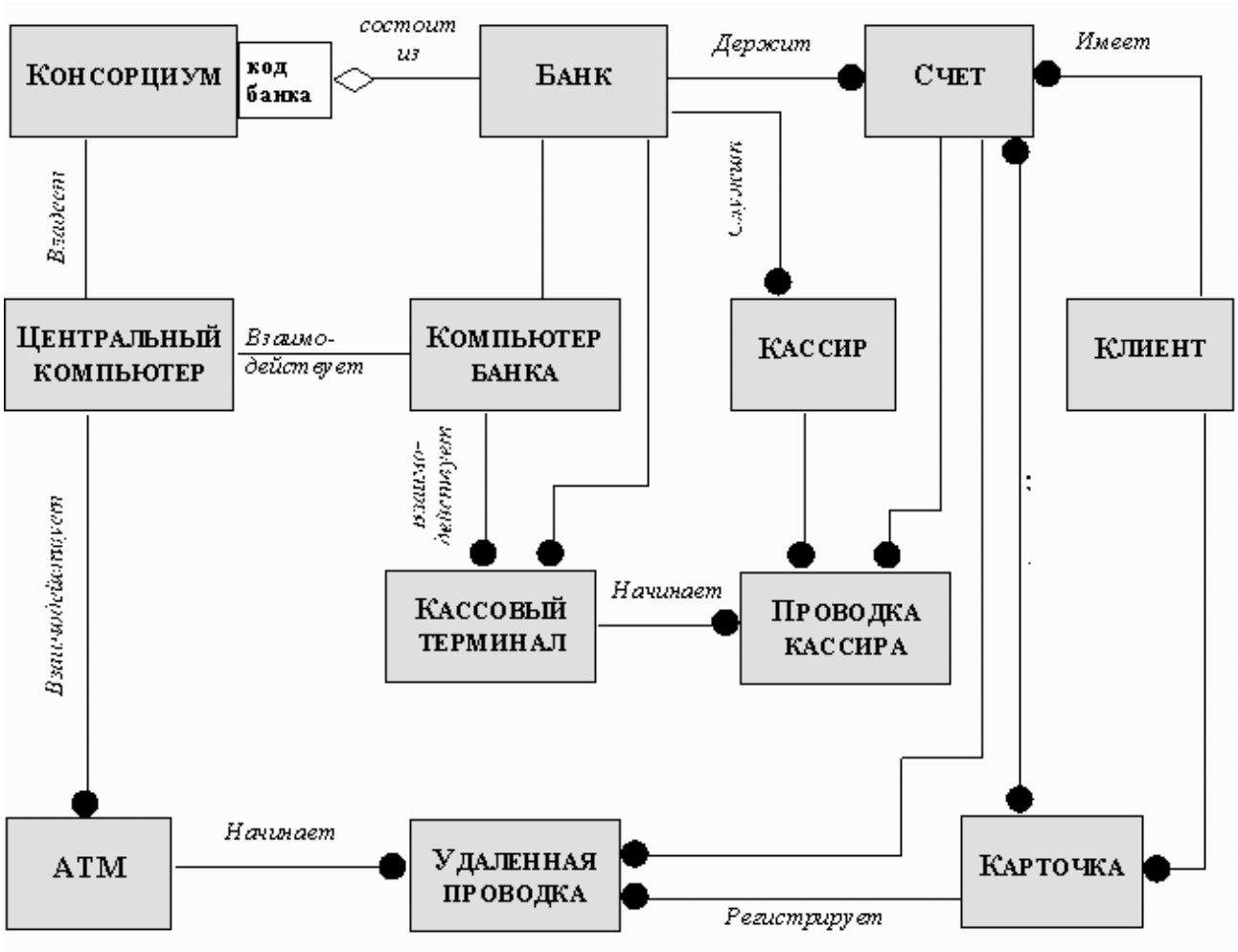

Рис. 2.37. Первая версия объектной диаграммы для банковской сети

2.3.4. Уточнение атрибутов

Применяя критерии, сформулированные в п. 2.2.4, получим:

Карточка содержит код_банка и код_карточки; их можно считать атрибутами объектов класса карточка, но удобнее использовать в качестве квалификаторов, так как код_банка обеспечивает выбор банка, сокращая кратность зависимости консорциум - банк; для аналогичного использования кода_карточки необходимо добавить зависимость Банк выпускает карточки, квалификатором которой будет код_карточки.

После внесения перечисленных изменений диаграмма примет вид, представленный на рисунке 2.38.

2.3.5. Организация системы классов с использованием наследования

В рассматриваемом примере естественно определить суперклассы для объектов, определяющих различные терминалы: кассовый_терминал и АТМ (банкомат), и для объектов, определяющих проводки: проводка_кассира и удаленная_проводка (с банкомата).

Внеся соответствующие изменения, получим объектную диаграмму, представленную на рисунке 2.39.

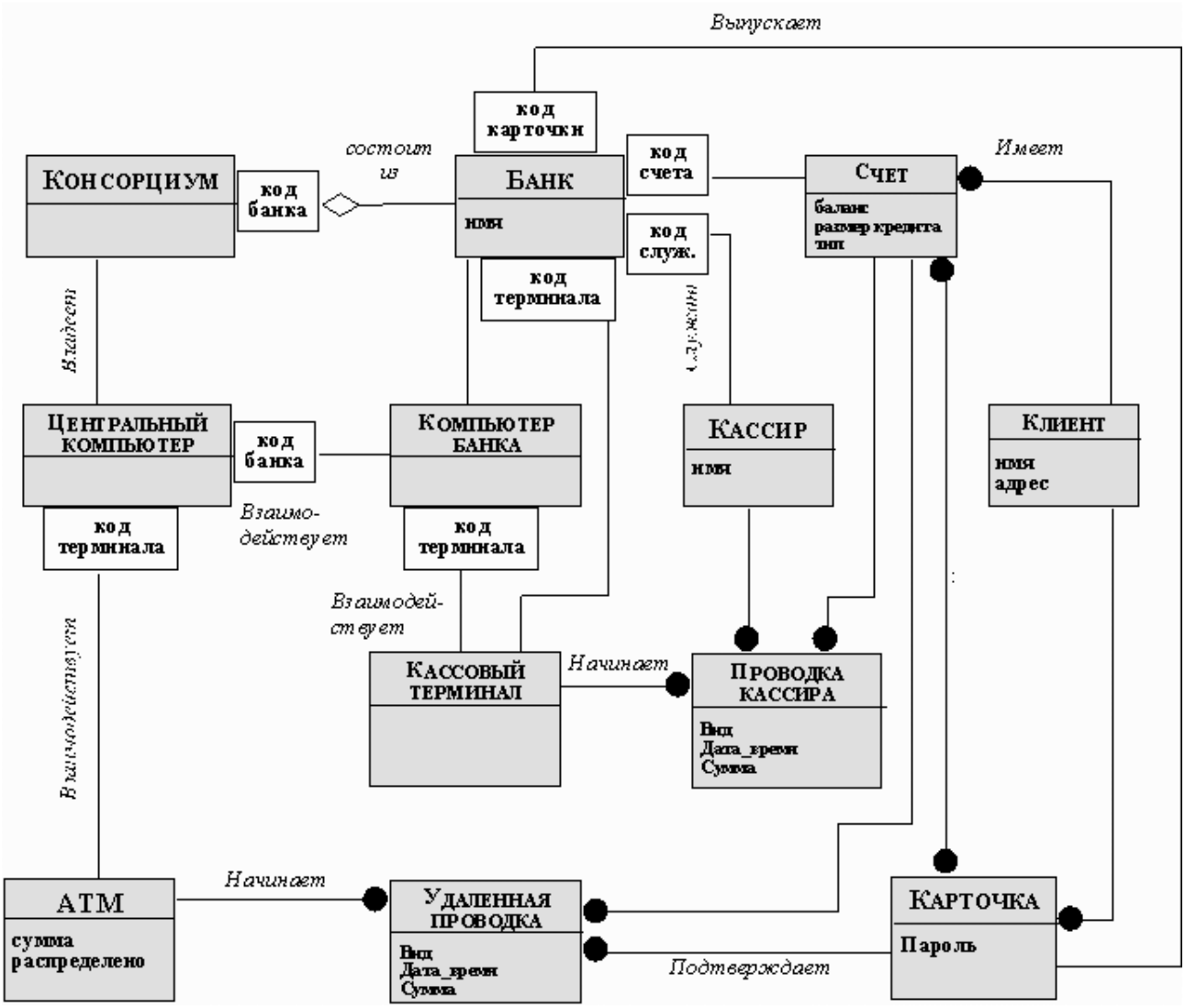

Рис. 2.38. Объектная диаграмма для банковской сети после уточнения атрибутов и добавления квалификаторов

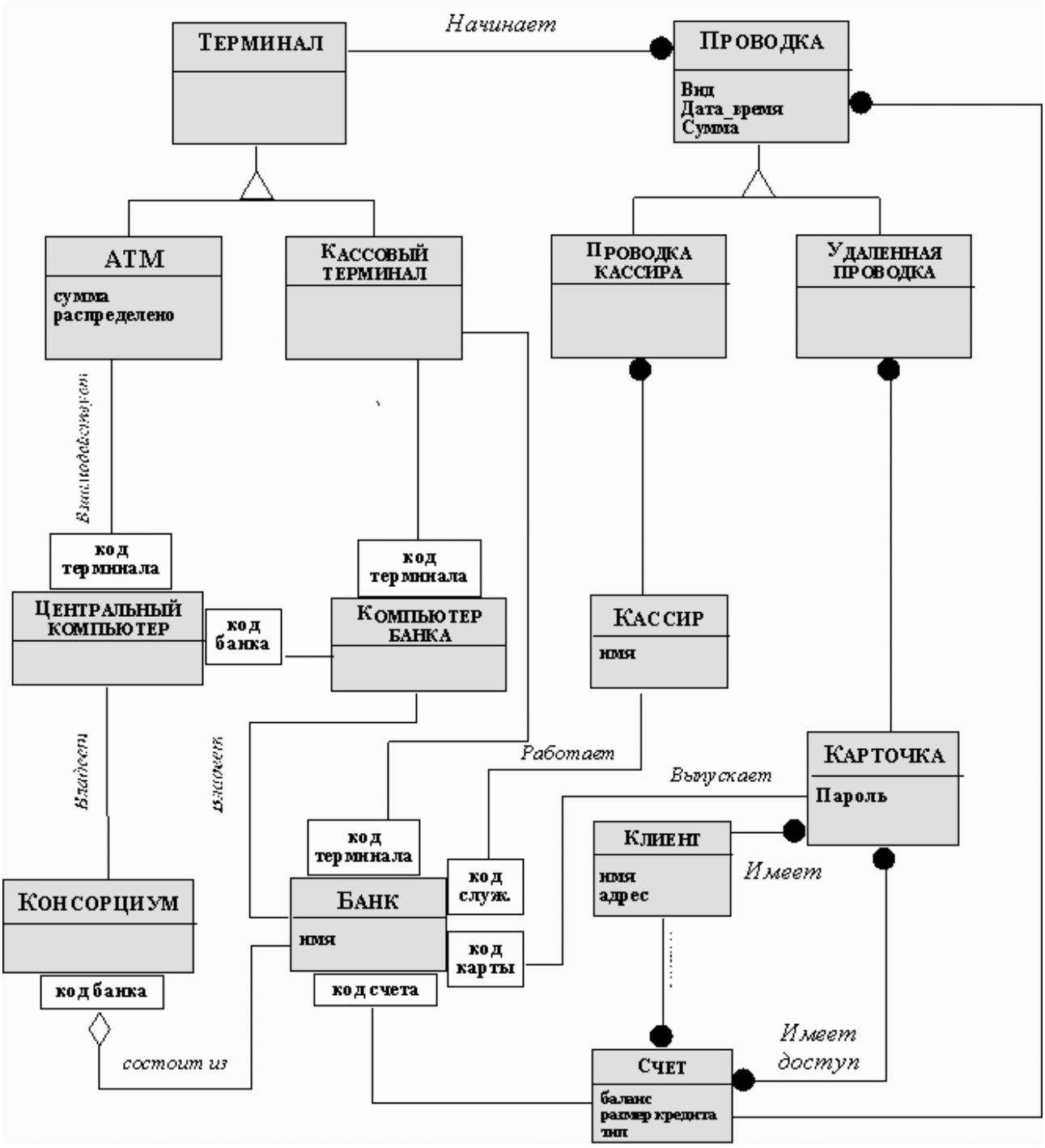

Рис. 2.39. Объектная диаграмма для банковской с учетом наследования

2.3.6. Дальнейшее усовершенствование модели

Карточка выступает в двух сущностях: как регистрационная единица в банке (сберкнижка), обеспечивающая клиенту доступ к его счетам, и как структура данных, с которой работает АТМ. Поэтому удобно расщепить класс карточка на два класса: регистрация_карточки и карточка; первый из этих классов обеспечивает клиенту доступ к его счетам в банке, а второй определяет структуру данных, с которой работает АТМ.

Класс проводка удобно представить как агрегацию классов изменение, так как проводка - это согласованная последовательность внесения изменений в счета и другие банковские документы; при работе с банковскими документами рассматривается три вида изменений: снятие, помещение и запрос.

Класс банк естественно объединить с классом компьютер_банка, а класс консорциум - с классом центральный_компьютер.

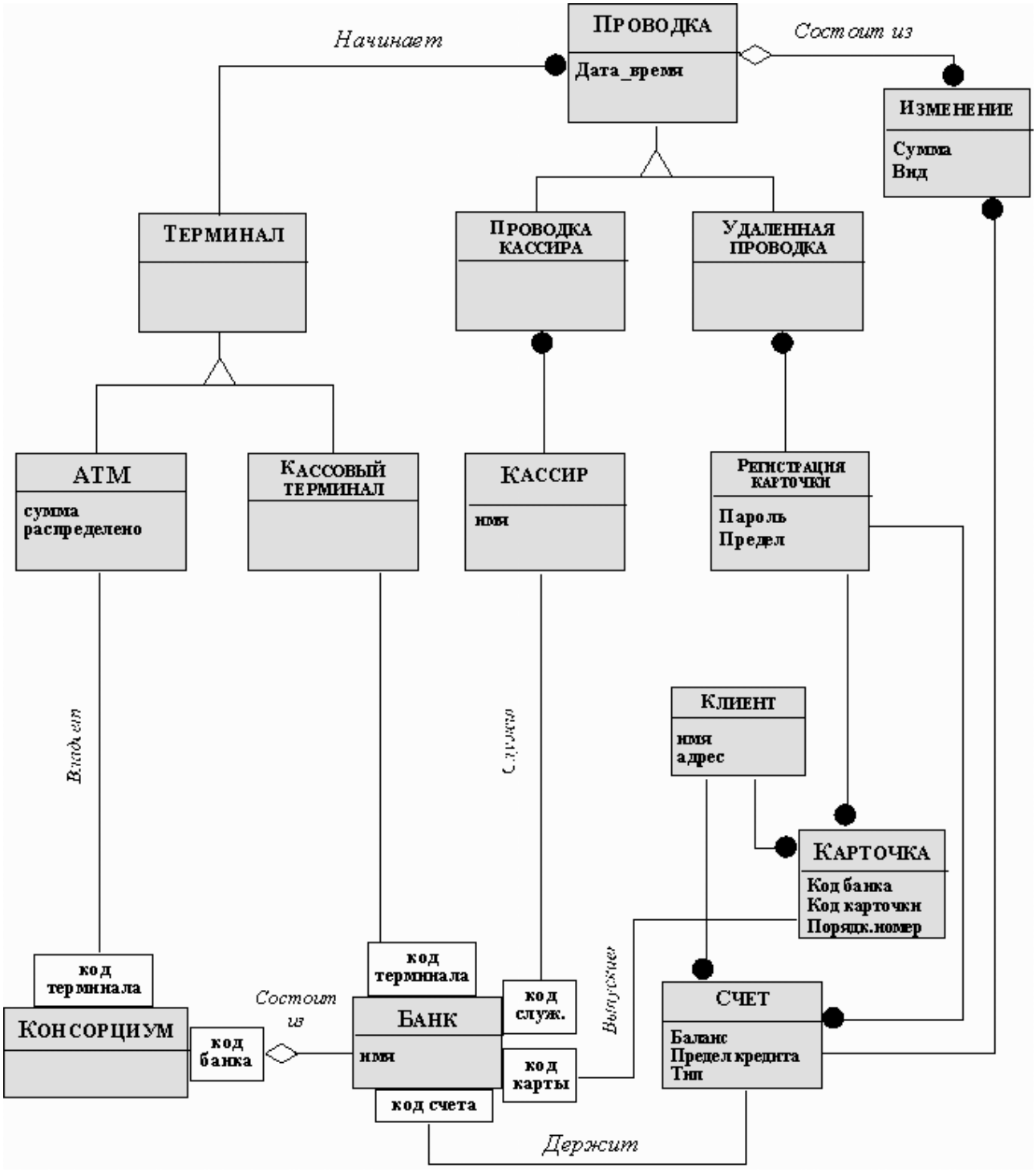

Рис. 2.40. Окончательный вид объектной диаграммы для банковской сети

После внесения перечисленных изменений объектная диаграмма примет вид, представленный на рисунке 2.40. На этом построение объектной модели этапа предварительного проектирования заканчивается. Дальнейшие уточнения объектной модели будут производиться на следующих фазах жизненного цикла системы.